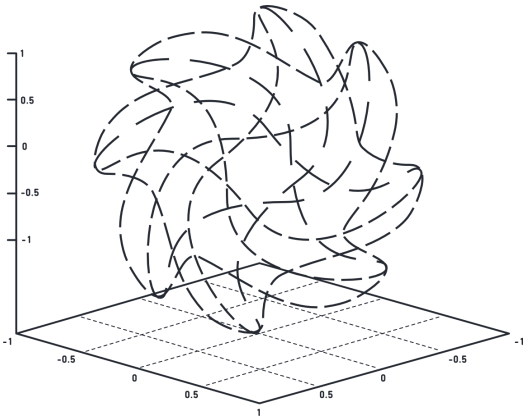


МАТЕРИАЛЫ ЗАДАНИЙ

командной инженерной олимпиады школьников

«Олимпиада Национальной технологической

инициативы»

по профилю

ПРОГРАММНАЯ ИНЖЕНЕРИЯ ФИНАНСОВЫХ ТЕХНОЛОГИЙ

2017/18 учебный год

http://nti-contest.ru

Оглавление

Введение **5**

ПРОФИЛЬ «ПРОГРАММНАЯ ИНЖЕНЕРИЯ ФИНАНСОВЫХ ТЕХНОЛОГИЙ» **16**

I Первый отборочный этап **18**

2 Первая попытка **18**

2.1 Задачи по математике (9 класс) 18

2.2 Задачи по математике (10-11 класс) 20

2.3 Задачи по информатике 22

3 Вторая попытка **32**

3.1 Задачи по математике (9 класс) 32

3.2 Задачи по математике (10-11 класс) 34

3.3 Задачи по информатике 36

4 Третья попытка **44**

4.1 Задачи по математике (9 класс) 44

4.2 Задачи по математике (10-11 класс) 47

4.3 Задачи по информатике 49

5 Критерии отбора победителей и призеров первого этапа **59**

II Второй отборочный этап **60**

6 Задачи второго этапа **60**

6.1 Задачи по математике (9 класс) 60

6.2 Задачи по математике (10-11 класс) 64

6.3 Задачи по информатике 67

7 Критерии определения призеров и победителей второго этапа **126**

III Финальный этап **127**

8 Задачи индивидуального тура **127**

8.1 Задачи по математике (9 класс) 128

8.2 Задачи по математике (10-11 класс) 130

8.3 Задачи по информатике 133

9 Задача командного тура **142**

9.1 Легенда 142

9.2 Набор заданий 144

9.3 Условия проведения 148

9.4 Процедура проведения приемочного тестирования и критерии оценки . 150

9.5 Подробное описание подзадач 152

9.6 Решение 211

10 Критерий определения победителей и призеров заключительного этапа **258**

ВВЕДЕНИЕ

Олимпиада Национальной технологической инициативы[1] (далее – Олимпиада НТИ) – это командная инженерная олимпиада школьников, завершающаяся разработкой действующего устройства, системы устройств или компьютерной программы. Олимпиада является проектом Агентства стратегических инициатив, элементом дорожной карты НТИ «Кружковое движение» и ключевым механизмом вовлечения инженерно-ориентированных школьников в образовательные программы высшего образования, ориентированные на рынки НТИ. Оператором Олимпиады НТИ является некоммерческая организация - Ассоциация участников технологических кружков. Профили Олимпиады НТИ выбраны на основе приоритетов Национальной технологической инициативы: «Автономные транспортные системы», «Большие данные и машинное обучение», «Системы связи и дистанционного зондирования Земли», «Интеллектуальные энергетические системы», «Создание систем протезирования (Нейротехнологии)», «Инженерные биологические системы», «Интеллектуальные робототехнические системы», «Беспилотные авиационные системы», «Ядерные технологии», «Наносистемы и наноинженерия», «Технологии беспроводной связи», «Электронная инженерия: Умный дом», «Передовые производственные технологии», «Виртуальная и дополненная реальность», «Новые материалы и сенсоры», «Водные робототехнические системы» и «Программная инженерия финансовых технологий».

Целевыми победителями Олимпиады НТИ являются школьники, способные реализовывать сложные технические проекты в прорывных областях. Олимпиада должна выделять команды участников с особыми характеристиками мышления, коммуникации и действия, необходимыми для решения задач НТИ. Победители и призеры Олимпиады НТИ должны показывать высокие результаты в области применения предметных знаний в практической работе. Одновременно с этим, система подготовки Олимпиады НТИ должна предоставлять участникам инструменты для подготовки и получения недостающих знаний и практических навыков.

Первый год проведения Олимпиады

Олимпиада НТИ была впервые проведена в 2015/16 учебном году. В отборочных этапах олимпиады приняли участие несколько тысяч школьников, около ста из них были приглашены к участию в заключительном этапе по профилям «Большие данные и машинное обучение», «Системы связи и дистанционного зондирования Земли», «Интеллектуальные энергетические системы», «Автономные транспортные системы». Заключительный этап Олимпиады и торжественные мероприятия проводились в ВДЦ «Орленок».

В 2015-16 учебном году победители и призеры олимпиады могли воспользоваться возможностью добавить дополнительные 10 баллов к сумме баллов за вступительные экзамены, в случае если они поступали в вузы-организаторы Олимпиады НТИ.

[1]Национальная технологическая инициатива (НТИ) — это программа мер, нацеленная на формирование принципиально новых рынков и создание условий для глобального технологического лидерства России к 2035 году. Задача по созданию НТИ поставлена Президентом Российской Федерации 4 декабря 2014 года в Послании к Федеральному собранию.

Второй год проведения Олимпиады

В 2016/17 учебном году Олимпиада проводилась во второй раз по 12 профилям, количество зарегистрированных для участия школьников увеличилось более чем в три раза и достигло 12 тыс., в отборочных этапах приняли активное участие 4 тыс. школьников, на заключительный этап прибыло 306 участников.

Заключительный этап Олимпиады и торжественные мероприятия проводились на площадке ОЦ «Сириус», в лабораториях и помещениях Парка Наук и Искусств. Вечером проходили лекции и неформальные встречи с представителями технологических компаний.

В 2016-17 учебном году четыре профиля Олимпиады НТИ («Автономные транспортные системы», «Большие данные и машинное обучение», «Системы связи и дистанционного зондирования Земли», «Интеллектуальные энергетические системы») входили в «Перечень олимпиад школьников», таким образом победители и призеры смогли воспользоваться льготами при поступлении в вузы России (зависит от правил приема конкретного вуза). Победители и призеры новых профилей также могли воспользоваться бонусами при поступлении в вузы, которые имеют статус «организатор Олимпиады НТИ».

Третий год проведения Олимпиады

В отборе на Олимпиаду 2017/18 учебного года приняло участие более 20 тыс. школьников, подавших более 50 тыс. заявок на различные профили, число которых увеличилось до 17. В финал вышли 578 участников Олимпиады из 51 региона РФ:

Финал стал распределенным и проходил с февраля по апрель 2018 года: Олимпиаду приняли ОЦ «Сириус», МАИ, МИФИ, ТПУ, Университет Иннополис, СПбПУ, ДВФУ, УрФУ. В 2017-18 учебном году девять из 17 профилей Олимпиады НТИ входят в «Перечень олимпиад школьников» (приказ Минобрнауки России от 30.08.2017 № 866) и дают льготы при поступлении в ВУЗы.

Важная составляющая подготовки участников к финалу Олимпиады – открытые для всех желающих хакатоны, вебинары и мастер-классы. Программы этих мероприятий разработаны педагогами профилей Олимпиады НТИ специально для регионов так, чтобы их

можно было провести на минимальном количестве оборудования. Сеть региональных партнеров Олимпиады со статусом Методическая площадка или Площадка подготовки растет с каждым годом, и в 2018 году, на третий год проведения Олимпиады, их количество достигло 74, всего проведенных мероприятий по подготовке (соревнований, хакатонов, сборов) – более 50. Информация о партнерских площадках размещена в специальном разделе официального сайта олимпиады: http://nti-contest.ru/places_to_prepare/.

Организаторы и партнеры Олимпиады НТИ

Оргкомитет олимпиады представлен ректорами крупнейших политехнических и инженерных вузов России, руководителями технологических компаний и представителями государственных органов.

Вузы-соорганизаторы олимпиады:

- ФГБОУ ВО «Московский политехнический университет»;
- ФГАОУ ВО «Санкт-Петербургский политехнический университет Петра Великого»;
- ФГАОУ ВО «Национальный исследовательский Томский политехнический университет».

Технологические партнеры

Олимпиада НТИ проводится при поддержке технологических партнеров, количество которых увеличилось, по сравнению с прошлым годом, среди них: РВК (Российская венчурная компания) и АСИ (Агентство стратегических инициатив по продвижению новых проектов) – в роли со-организаторов и генеральных партнеров: Сбербанк, ПАО «Сухой», ОАК, ФИОП Роснано, сеть детских технопарков «Кванториум», Спутникс, Полюс-НТ, BiTronicsLab, КРОК и др. Полный список организаторов и партнеров олимпиады размещен в соответствующем разделе на официальном сайте: http://nti-contest.ru/about/.

Вузы-организаторы профилей Олимпиады НТИ:

- ФГАОУ ВО «Дальневосточный федеральный университет»;
- АНО ВО «Университет Иннополис»;
- ФГАОУ ВПО «Национальный исследовательский ядерный университет «МИФИ»;
- ФГАОУ ВО «Московский физико-технический институт (государственный университет)»;
- ФГАОУ ВО «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (Университет ИТМО);
- ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)»;
- ФГБОУ ВО «Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева»;
- ФГАОУ ВО «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина.

К работе методической комиссии был привлечен профессорско-преподавательский состав вузов-организаторов и представители реального сектора экономики. Объективную оценку работы осуществляет жюри, представленное основателями технологических компаний, а также представителями вузов-организаторов.

Вузы-организаторы, входящие в Оргкомитет Олимпиады НТИ, ведут непрерывную работу с талантливыми школьниками.

Московский политехнический университет

Московский политехнический университет при активном участии Инженерной школы (факультета) регулярно ведет мероприятия для школьников. Факультет “Инженерная школа” создан в 2016 году в целях развития работы Московского Политеха с детьми старшего школьного возраста и организации участия университета во всероссийских и региональных программах по поддержке талантливых школьников.

Факультет курирует проект «Инженерный класс в московской школе», инициированный в 2015 году Департаментом образования города Москвы в целях повышения количества выпускников московских школ, поступивших в инженерные вузы столицы. В 2017 году под научно-методическим руководством сотрудников факультета открыты инженерные классы в 26 школах города Москвы (более 1500 учащихся 10-11 классов). Преподаватели инженерной школы ведут занятия в технологических кружках на базе Московского Политеха, курсы повышения квалификации для преподавателей московских школ, организуют инженерные соревнования и профориентационные мероприятия: экскурсии на предприятия, встречи с носителями профессионального опыта, инженерные турниры и соревнования.

Для подготовки учащихся к инженерной проектной деятельности и вовлечения их в техническое творчество инженерная школа с октября 2016 г. открыла кружки для старшеклассников (8-11 класс):

• Космическая инженерия;
• Кружок радиоэлектроники и программирования;
• Программирование на С++;
• Аквапонные системы;
• Кружок 3D моделирования и прототипирования.

Важнейшим направлением работы факультета является организация выездных инженерных школ и профильных смен, попасть на которые имеют возможность дети из любых регионов России, проявившие уникальные способности в научно-технической сфере.

Образовательный центр «Сириус» и Московский политехнический университет являются официальными партнерами. Летом 2016 года Московский Политех выступил соорганизатором трех направлений проектной деятельности в ОЦ «Сириус» и осуществил экспертную поддержку деятельности центра по направлениям «Транспорт» и «Космос». В июле 2017 Московский Политех стал соорганизатором направления «Наука», в котором приняли участие 400 учеников 8-10 классов, прошедшие конкурсный отбор. Преподаватели и студенты университета приняли участие в организации направлений «Беспилотный транспорт и логистические системы», «Спутники и пилотируемая космонавтика», «Персонализированная медицина» и «Современная энергетика». В планах факультета - создание инженерных кружков Московского Политеха на базе Сириуса (радиотехника, аэрокосмическая инженерия) и лаборатория беспилотного транспорта.

С 2016 г. факультет организует участие Московского Политеха в ежегодном форуме талантливых детей «Проектория» в гор. Ярославле. Форум проводится под руководством аппарата полномочного представителя Президента РФ в ЦФО и Министерства образования и науки РФ. В форуме принимают участие до 500 школьников со всей страны. В ноябре 2016 г. и сентябре 2017 г. Московский Политех выступил партнером образовательной программы форума в рамках направления «Технологии движения».

С 2016 г. Инженерная школа (факультет) является соорганизатором и партнером инженерно-конструкторских школ «Лифт в будущее» - программа БФ «Система» по поддержке талантливой молодежи. В течение октября 2017 г. вместе с преподавателями Московского Политеха в ВДЦ «Орленок» дети разрабатывали технологические проекты, две из восьми лабораторий курировали сотрудники и студенты университета.

В январе 2018 г. совместно с Центром педагогического мастерства (Департамент образования гор. Москвы) факультет провел зимнюю инженерную школу Московского Политеха для учащихся инженерных классов города Москвы, приняли участие 50 человек.

В марте 2018 г. открылся детский технопарк Центра развития инжиниринга-инженерно-технологический комплекс, на базе которого проводятся углубленные технико-ориентированные курсы дополнительного образования для школьников; является частью Московского политехнического университета, создан при поддержке Департамента науки, промышленной политики и предпринимательства города Москвы. На данный момент запущены 4 образовательных программы: Транспортный дизайн, Введение в автомобилестроение, Беспилотный транспорт и Современная космонавтика.

Вместе с тем Московский Политех принимает участие в организации других олимпиад, входящих в перечень олимпиад школьников Минобрнауки России 2017/2018 учебного года:

1. Объединенная межвузовская математическая олимпиада (ОММО). Проводится для одиннадцатиклассников по инициативе группы московских вузов с 2009 года.

2. Международная олимпиада школьников «Искусство графики». Проводится с целью выявления и привлечения наиболее подготовленных, талантливых и профессионально ориентированных учащихся средних художественных училищ РФ и ближнего зарубежья, школьников, слушателей подготовительных курсов, развитие у обучающихся творческих способностей, содействие профессиональной ориентации школьников.

Московский Политех также участвует (имеет статус организатора или со-организатора) в следующих мероприятиях: инженерно-конструкторское направление предпрофессиональной олимпиады Московской олимпиады школьников 2016-18 гг., предпрофессиональный экзамен для инженерных классов в Москве 2016-2018 гг., инженерное направление Московского конкурса научно-исследовательских и проектных работ учащихся, проектные смены ОЦ «Сириус», турнир двух столиц по робототехнике и т.д.

Санкт-Петербургский политехнический университет Петра Великого

Санкт-Петербургский политехнический университет Петра Великого с целью популяризации инженерных наук и привлечения талантливых абитуриентов проводит ряд масштабных мероприятий:

- летняя и зимние политехнические школы;
- олимпиады для школьников, в которых в прошлом году приняло участие более 1800 школьников;
- организовано взаимодействие с региональными центрами развития творчества одаренных детей;
- в различных школах и кружках, организованных цифровой мастерской Фаблаб Политех ежегодно принимает участие более 5000 школьников из Санкт-Петербурга и Ленинградской области. С 2015 года Фаблаб помогает развивать центры технического творчества и в регионах, наиболее крупный проект представлен в городе Норильск;
- для привлечения и подготовки школьников к поступлению созданы 5 массовых онлайн-курсов, размещенных в настоящее время на портале Лекториум: «Инженерное дело», «Математика», «Физика», «Химия», «Обществознание».

Одним из ключевых событий 2015 года стало проведение Фестиваля научно-технического творчества молодежи на территории Политехнического университета Петра Великого. Мероприятие, ориентированное на школьников 10 и 11 классов, посетило более 15 000 абитуриентов.

Политехнический университет Петра Великого является организатором олимпиад, входящих в перечень Минобрнауки РФ: Межрегиональная олимпиада школьников по математике и криптографии, Политехническая олимпиада школьников, Объединенная межвузовская математическая олимпиада).

Томский политехнический университет

Томский политехнический университет является одним из опорных вузов в программе инновационного развития ПАО «Газпром». В 2009 году ТПУ и ПАО «Газпром» подписали соглашение о сотрудничестве. В Институте природных ресурсов создан и действует полный учебно-научный цикл, включающий обучение (рабочим профессиям, прохождение производственных преддипломных практик, переподготовку и повышение квалификации специалистов компании, создание кадрового задела) и исследования (Инновационный научно-образовательный центр трубопроводного транспорта нефти и газа, в составе которого действует Международная научно-образовательная лаборатория «Нефтегазовая гидродинамика и теплообмен», лаборатория неразрушающего контроля).

Участие со своим треком «Электронная инженерия: Умный дом» в рамках Олимпиады НТИ стало логичным продолжением работы Томского политехнического университета по развитию инженерных навыков у школьников и студентов. На траектории элитного технического образования ТПУ студенты воплощают в жизнь свои самые смелые инженерные идеи. Однако многолетний опыт показывает, что деятельность по развитию интереса к инженерии и закладыванию базовых знаний в предметной области нужно прививать гораздо раньше, поэтому Томский политехнический университет ведет постоянную работу по привлечению в инженерные профессии талантливых школьников с более раннего возраста. Одним из таких мероприятий стал проект «Юный инженер», который проводится совместно с Департаментом образования Томской области и НП “Лифт в будущее”. Суть проекта в том, чтобы студенты ТПУ, уже имеющие опыт в реализации инженерных проектов, готовили и проводили учебные занятия со школьниками г. Томска и Томской области. Цель проекта сделать доступным современное дополнительное образование для детей всего региона. А также развивать практику наставничества, которая реализуется в ТПУ с 2012 г. и в 2018 г. вошла в шорт-лист лучших практик на форуме “Наставник”.

Для знакомства с инженерными профессиями будущего с 2016 г. в ТПУ запущена игра «Агенты будущего» (http://corpus-future.net/) - это образовательная online-система, построенная на технологиях игрового моделирования, геймификации и многопользовательского онлайн-обучения. Игра построена на выполнении квестов (заданий). Каждое задание позволяет изучить аспект или технологии отдельной отрасли инженерии/техники. В ходе прохождения сюжета школьник проходит путь к будущей инженерной специальности в качестве исследователя, технолога, стратега или оператора. Квест-задания проходят в энергоблоке, в виртуальных лабораториях, вычислительном центре и других симуляторах.

Важным этапом в ходе подготовки к Олимпиаде НТИ стала Школа инженерных проектов ТПУ (ШИП), во время которой школьники от 14 до 17 лет создавали макет узла космического аппарата, макет спутника, могли разработать и реализовать отдельные комплексы, которые по алгоритмам, разработанными школьниками, рассчитывают оптимальные характеристики передачи, позволяющие снизить потери электроэнергии.

Целевая аудитория ШИП – учащиеся образовательных учреждений от 14 до 17 лет. Преподаватели ТПУ выезжают к школьникам других городов с проектами, которые в течении недели реализуют с ребятами. Тематики проектов: «Проектирование малых космических аппаратов с использованием 3D моделирования». Результат: макет узла космического аппарата и макет спутника, выполненный посредством 3D печати. «Технологии Smart Grid в электросетях нового поколения». Результат: разработка и реализация отдельных комплексов программно-аппаратных средств, которые по алгоритмам, разработанными школьниками, рассчитывают оптимальные характеристики передачи, позволяющие снизить потери электроэнергии. «Теплотехнологическая переработка биомассы в энергетические продукты и энергию». Результат: собственный вариант теплотехнологической энергоустановки и созданный на её основе экспериментальный стенд. План на 2018/19 уч. год разработка ШИП по тематике «Газовый комплекс».

Со школьниками проводится много других интересных и полезных мероприятий:

«Университетские субботы» - проект, позволяющий школьнику вникнуть в естественные и точные науки (физика, химия, математика, информатика), узнать о потенциале ТПУ, его лабораториях и передовых разработках. Здесь школьникам предоставляются физические задачи с техническим содержанием, которые разрабатываются в колоборации с преподавателями-физиками и преподавателями исследовательских школ (институтов). Решение практических задач для школьников младших классов представлено в виде красочного шоу. Такой подход позволяет школьникам увидеть законы физики в действии.

В ТПУ сформирована единая система довузовской подготовки инженерно-технического образования – подготовительные курсы ТПУ, профильные классы Газпрома, центр занимательных наук «Склад ума».

Школьники имеют возможность работать в KIDsLAB, где проводится обучение в кружках начального технического творчества, авиамодельном, стендового моделизма, робототехники. KIDsLAB - это производственно-образовательная мастерская, где можно изготовить практически все. Основной задачей KIDsLAb является помощь школьникам и студентам первого курса в приобретении опыта изготовления инженерных предметов, внедрения своей разработки - от идеи до создания прототипа, поиске маркетинговых решений и создании инновационных продуктов.

Томский политехнический университет принимает участие в организации олимпиад, входящих в перечень олимпиад школьников Минобрнауки России 2017/2018 учебного года:

1. Интернет-олимпиада школьников по физике, целью олимпиады является повышение качества знаний по физике, повышение мотивации к поступлению в технические вузы. С 30.03.2017 г. олимпиада получила статус международной.
2. Многопрофильная инженерная олимпиада «Звезда». Участие в данной олимпиаде дает возможность школьнику существенно расширить свой кругозор, получить полное представление о тенденциях развития экономики и техники.
3. Межрегиональная олимпиада школьников по математике САММАТ.
4. Олимпиада школьников Сибирского Федерального округа «Будущее Сибири» - мотивирует школьников для поступления в технические вузы и расширяет знания в предметной области.
5. Межрегиональная олимпиада школьников «Высшая проба» дает возможность учащимся попробовать себя в разных областях знаний, как в технических, так и в гуманитарных, что способствует гармоничному развитию личности.

Также ТПУ является соорганизатором таких олимпиад, конкурсов и конференций как:

1. Политехническая олимпиада по математике, физике, химии, информатике, русскому языку, обществознанию, проводимая на территории стран СНГ. Данная олимпиада нацелена на выявление талантливых абитуриентов в странах СНГ, развитие и проверку глубоких знаний по предложенным дисциплинам.
2. Олимпиада ПАО «Газпром», направленная на развитие креативного подхода к решению инженерных задач, проверки знаний и мотивации к изучению газодобывающей отрасли.
3. Олимпиада для детей с ОВЗ. Совместно с ОГБОУ «Школа-интернат для обучающихся, нуждающихся в психолого-педагогической и медико-социальной помощи» в декабре 2017 года проведена VI Межрегиональная дистанционная олимпиада школьников с ограниченными возможностями здоровья, в которой приняли участие 600 школьников из 19 регионов РФ, Казахстана, Беларуси.
4. Всероссийская конференция-конкурс исследовательских работ школьников "Юные исследователи - науке и технике" направлена на выявление и развитие у молодежи творческих способностей и интереса к научно-исследовательской деятельности, создание необходимых условий для поддержки одаренной молодежи, распространение и популяризация научных знаний среди молодежи.
5. Региональные конкурсы «Мой выбор — химия», «От школьной физики – к высоким технологиям», «Царица наук - математика», «Выбор будущего - информатика»

направлены на стимулирование учителей химии, физики, математики и информатики в школах за высокие показатели учеников в сдаче единого государственного экзамена (ЕГЭ) в 11 классах и во Всероссийских олимпиадах. В 2017 году приняли участие 263 учителя.

6. Ежегодно летом с июня по август на базе ТПУ проводятся летние смены. Каждая смена имеет свою тематику. Ребятам предлагают и усилить свои знания по профильным для ТПУ предметам, и принять участие в проектной и исследовательской деятельности, а также начать изучение китайского языка.

Дальневосточный Федеральный Университет

Дальневосточный Федеральный Университет регулярно проводит мероприятия по привлечению абитуриентов, выявлению и поддержке талантливых школьников в области инженерных и естественных наук.

1. Программы дополнительного образования:

Программы «Малых академий» для школьников 9-11 классов. В период с 2013 по 2017 годы было организовано 8 профильных инженерных и естественнонаучных «малых академий» (охват школьников составил более 6000 человек):

- малая инженерная академия;
- школа юного системотехника;
- школа юных биологов и экологов;
- школа юных химиков;
- школа юных математиков;
- школа юных физиков;
- школа юных программистов;
- малая академия биомедицины по профилям: медицинское направление и пищевые биотехнологические и биотехнические направления.

На базе Университета организованы курсы подготовки к ЕГЭ по профильным предметам:

- физика;
- математика;
- биология;
- химия.

Важными мероприятиями для Университета являются краткосрочные интенсивные программы, такие как «Тихоокеанская математическая школа». В марте 2017 года такой интенсив прошли 59 школьников 10-11 классов со всего Приморского края.

2. Олимпиады и конкурсы:

В 2015 - 2016 уч. году в ДВФУ организовано и проведено 33 Олимпиады школьников: в том числе 11 олимпиад, в которых ДВФУ является организатором и 22 олимпиады, в которых университет выступает в роли региональной площадки.

В 2015-16 учебном году Дальневосточный федеральный университет послужил в качестве региональной площадки для проведения Всероссийских олимпиад школьников инженерной и естественно-научной направленности:

- «Полуфинал Всероссийской командной школьной олимпиады по программированию»;
- Всероссийская олимпиада школьников (муниципальный и региональный этапы);
- Межрегиональная Всероссийская олимпиада школьников «Ломоносов» по биологии и математике (2016 год - 48 участников, из них – 1 победитель, 6 призеров из территорий Приморского края, Хабаровского край, Республики Саха (Якутия));
- Всероссийская «Объединенная межвузовская математическая олимпиада школьников» по математике (2016 год – 24 участника из территорий Приморского края, Хабаровского края, Амурской области);
- Многопрофильная инженерная олимпиада школьников «Звезда» по технике и

технологии кораблестроения и морской техники, машиностроению, технике и технологии наземного транспорта, авиационной и ракетно-космической технике, технологии материалов, ядерной энергетике, электронике, радиотехнике и системе связи (2016 год - 250 участников (7 - 11 классы), призеры – 30, победители – 10);

- Всероссийская «Северо-Восточная олимпиада школьников» по математике, информатике и химии (2016 год - 36 участников, из них – 1 победитель, 9 призеров из территорий Приморского края, Хабаровского края, Амурской области).

Дальневосточный федеральный университет также выступил как организатор следующих олимпиад:

- «Океан знаний» по математике, физике, химии, биологии, экологии.
- Турнир юных программистов.

На протяжении 20 лет Университет выступает соорганизатором Регионального этапа Всероссийской олимпиады школьников «Приморский интеллект» организованной на базе ВДЦ «Океан» (500 участников ежегодно).

С 2012 года реализуется уникальная совместная образовательная программа ДВФУ и ВДЦ «Океан» в рамках смены «Российский интеллект» для победителей и призеров региональных этапов Всероссийской олимпиады школьников и победителей/призеров отборочного этапа олимпиады школьников «Океан знаний» 9, 10 и 11 классов (2015 год - 250 учащихся из 22 территорий, 2016 год - 400 учащихся из 21 территории России).

Структура отбора участников Олимпиады НТИ

Соревнование проходит в три этапа. Первый и второй отборочные этапы проходили с октября по декабрь 2017 года в заочной форме на интернет-платформе «Stepik» (http://stepik.org) и в инженерных онлайн-симуляторах.

Отборочные этапы сопровождались различными подготовительными мероприятиями, среди которых были дистанционные мероприятия (вебинары), мероприятия для самостоятельной подготовки (онлайн-курсы), мероприятия направленные на командообразующую деятельность (специальные встречи, интенсивы, очные курсы на площадках по подготовке), мероприятия, направленные на получение практических навыков (интенсивы).

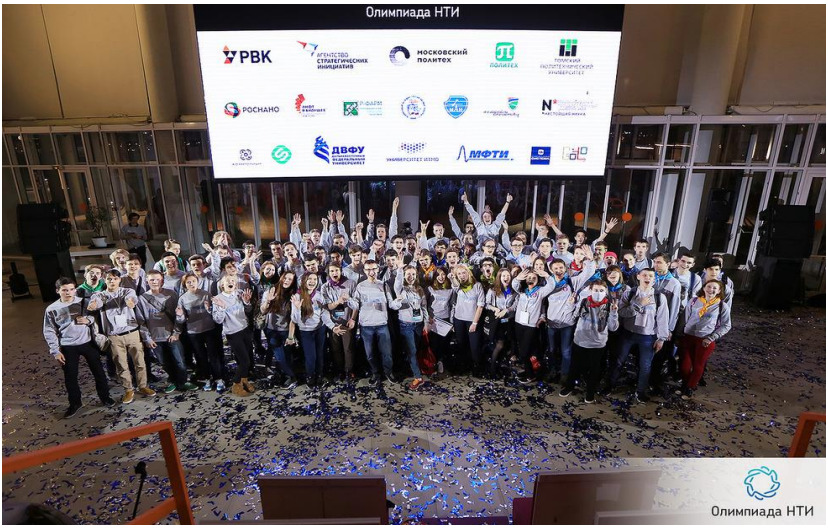
Заключительный этап Олимпиады состоит из двух частей: индивидуальное решение предметных задач по выбранным профилям и командная разработка инженерного решения с испытанием его на стенде. Задание второй части заключительного этапа имеет свою специфику для каждого профиля.

Работа с участниками

Организаторы Олимпиады заинтересованы в дальнейшем сопровождении ее участников. Практика показывает, что школьники – участники Олимпиады НТИ также заинтересованы в дальнейшем сотрудничестве. В организации заключительного этапа Олимпиады НТИ 2016\17 в качестве волонтеров приняли участие победители и призеры Олимпиады НТИ 2015\16, студенты первых курсов из различных регионов России. Участники заключительного этапа 2016\17 из числа учеников одиннадцатого класса также выразили желание принять участие в организации олимпиады и подготовке участников в качестве волонтеров.

В 2017/18 годах число партнерских мероприятий Олимпиады увеличилось: на странице http://nti-contest.ru/participants/posle_finala/ представлен список мероприятий организаторы которых специально приглашают участников Олимпиады и дают им бонусы при конкурсном отборе.

Так, на конкурсном отборе на авиационную смену в ВДЦ «Океан», призёры и победители профилей по «Интеллектуальным робототехническим системам» и «Беспилотным авиационным системам» получают весомые бонусы (70 и 130 баллов соответственно). На авиационную смену в «Артеке» получили приглашение лучшие участники профиля «Беспилотные авиационные системы». Отбор на июльскую проектную смену в ОЦ «Сириус» предполагает дополнительные баллы для призеров и победителей Олимпиады НТИ.



На фото: участники заключительного этапа Олимпиады во время торжественной церемонии закрытия.

Партнерство с инженерными соревнованиями

Оргкомитет Олимпиады НТИ, в свою очередь, ежегодно утверждает перечень инженерных мероприятий и конкурсов, победители которых, могут принять участие в заключительном этапе Олимпиады, минуя отборочные. В 2016/17 и 2017/18 таковыми мероприятиями являлись: IT-хакатон GoTo, инженерно-конструкторские школы «Лифт в будущее», всероссийский форум «Будущие интеллектуальные лидеры России» и World Skills High Tech.

Подготовка участников

Чтобы участники могли восполнить недостаток практических компетенций и изучить оборудование, на котором им предстоит работать на заключительном этапе Олимпиады НТИ, разработчики направлений представляют методические материалы для самостоятельной практики и самоподготовки, проводят вебинары для участников и педагогов с ответами на вопросы и подбирают подготовительные курсы. Все указанные материалы находятся в свободном доступе и размещены на официальном сайте олимпиады, на страницах профилей в разделе «Материалы для участников».

Прошедшая олимпиада является промежуточным итогом работы по реализации дорожной карты НТИ «Кружковое движение»: подготовка к ней велась в фаблабах, ЦМИТах, детских технопарках, на базе активных школ и лицеев, центров дополнительного образования по всей России. Рабочая группа «Кружковое движение» НТИ направлена на развитие технологического сообщества, объединяющего школьников и студентов, ориентированных на инженерную деятельность на рынках НТИ, самодеятельных технических энтузиастов, лидеров технологических кружков, разработчиков педагогических технологий, технологических предпринимателей, популяризаторов науки и технологий.

Популяризация Олимпиады НТИ

Олимпиада НТИ проходит с широким освещением мероприятий в различных средствах массовой информации (телевидение, печатные издания, электронные издания), среди которых как узкопрофильные издания, аудитория которых включает школьников старших классов, учителей, преподавателей в определенной научной среде, так и крупные федеральные издания общего характера. Во время проведения отборочных этапов Олимпиада НТИ освещалась в публикациях специализированных научных и общеобразовательных порталов (Занимательная робототехника, Дневник.ру, НИА Наука, Научная Россия, 5 углов, Нейроновости), официальных региональных порталов (Калининград, Тюмень, Курск, Курган, Тамбов, Мурманск, Новогород, Вологда и т.д.), в печатных изданиях («Качество образования»). Заключительный этап Олимпиады НТИ проходил при участии журналистов таких печатных изданий, как Российская газета, Комсомольская правда, Огонек, Сноб, Известия, Учительская газета; федеральных телевизионных каналов (Первый канал, Россия 24), научно-популярных блогов Мел, Постоянная Планка, Химия-Просто, Первый научный. Широкое освещение мероприятий заключительного этапа имеет своей целью распространение информации среди потенциальных участников Олимпиады НТИ будущего года - учеников 6-10 классов и направлено на привлечение талантливых школьников со всей России. Список лучших материалов об олимпиаде: http://nti-contest.ru/publications/.

ПРОФИЛЬ «ПРОГРАММНАЯ ИНЖЕНЕРИЯ ФИНАНСОВЫХ ТЕХНОЛОГИЙ»

Новый профиль для Олимпиады НТИ 2017-2018 учебного года ”Программная инженерия финансовых технологий” знакомил школьников с технологией блокчейн. Начиная со второго отборочного этапа участники профиля знакомились с алгоритмическими основами технологий распределенного реестра, а также с инструментарием для создания децентрализованных проиложений на базе платформы Ethereum. В результате учащиеся 9-11 классов, прошедшие все этапы олимпи.ады не только демонстрировали понимание основных технологических концепций: цепочка блоков, хэш-функция, цифровая подпись, валидация транзакций и блоков, но и могли самостоятельно реализовывать и отлаживать Ethereum-контракты согласно поставленной задаче. Для того, чтобы сделать это возможным в течение олимпиады проводилась цикл образовательных и отборочных мероприятий.

Первый отборочный дистанционный этап (индивидуальный) определял общий уровень подготовки школьников по предметам математика и информатика. Решая задачи по программированию, школьники должны были продемострировать простейшие навыки составления и отладки программ, обрабатывающих массивы данных, и понимание таких тем, как битовые операции, строковые операции, теория графов. Успешное решение данных задач показывало готовность участников знакомиться с образовательными материалами второго тура. Задачи по математике проверяли у ребят углубленные знания некоторых тем из школьной программы.

Логика составления задач второго отборочного этапа требовла от участников объединения в команды, поскольку их было сложно решить индивидуальному участнику. Из задач было сформировано два блока: математика и программирование. Для решения задач по математике школьники должны были изучить некоторые главы из таких тем, как теория чисел, теория информации и теория вероятности - те знания, которые помогут начать знокомиться с основами криптографии. Перед решением задач по программированию командам было предложен набор видео-лекций ”Введение в блокчейн технологии”, подготовленным Университетом Иннополис. После знакомства с курсом, необходимо было разработать алгоритмы для кодирования/декодирования информации, вычисления хэш-функций, работы с цифровой подписью, построения дерева Меркле и модифицированного префиксного дерева, валидации транзакций и блоков, отправки транзакций в сеть блокчейн и вычисления балансов счетов пользователей сети, а также продемонстрировать базовые навыки в написании и понимании работы Ethereum-контрактов.

После второго этапа команды, прошедшие в финал профиля, участвовали в двух дистанционных хакатонах. В ходе первого хакатона школьники должны были настроить свой собственный узел сети Ethereum и написать программное обеспечение для выполнения функционала простого кошелька. Второй хакатон был направлен на знакомство с понятием незаменяемых токенов (стандарт ERC721) - участникам предлагалось разработать простое децентрализованное приложение для управления

оцифрованными ценностями. Оба хакатона позволили командам получить все нобходимые навыки для решения задачи в заключительном этапе, поскольку полностью повторяли (но в более коротком промежутке времени) процесс проведения заключительного этапа: задания для школьников были сформулированы в виде пользовательстких историй (User Stories), необходимо было использовать систему ведения версий Git, а правильность решения проверять в автоматической системе тестирования.

Командная задача заключительно этапа была логическим заверешением той работы, которую проедлали участники на отборочных этапах. Школьники должны были написать децентрализованное приложение для платформы Ethereum, которое включало бы в себя клиентское приложение (Python-скрипт) и Etherum-контракты, в том числе контракты токенов по стандартам ERC20 и ERC721. Также в заключительный этап входил индивидуальный тур, в ходе которого участники решали задачи по математике и информатике. Задачи по математике подразумевали использование тех же знаний, что были приобретены участниками в ходе второго этапа. Задачи по информатике проверяли в простой форме воспроизводили задачи, которые могут вставать в индустрии разработки программного обеспечения перед разработчиками новых блокчейн платформ.

1. ПЕРВЫЙ ОТБОРОЧНЫЙ ЭТАП

Первый отборочный тур проводится индивидуально в сети Интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Для каждого из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, задачи по информатике общие для всех участников. Решение задач по информатике предполагало написание программ. Участники не были ограничены в выборе языка программирования для решения задач. На решение задач каждого предмета первого отборочного этапа участникам давалось 2 дня. У участников было три временных слота по 2 дня каждый, когда они могли решать задачи по предмету. Решение каждой задачи дает определенное количество баллов. За каждую следующую попытку сдать решение баллы начисляются по формуле $\frac{P}{N}$, где P - максимальное количество очков, которые можно получить за решение задачи, N - количество попыток. Всего на каждую задачу математике давалось по 2 попытки, на каждую задачу по информатике - 4 попытки.

Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 30 баллов.

Первая попытка

2.1. Задачи по математике (9 класс)

Задача 2.1.1. (2 балла)

Найдите сумму всех полных квадратов, которые при делении на 11 в частном дают простое число и в остатке 4.

Решение

$n^2 = 11p + 4 \Longrightarrow 11p = (n-2)(n+2)$. $n - 2 = 1$ не подходит, значит $n - 2 = 11$ и $n + 2 = p$ или $n - 2 = p$ и $n + 2 = 11$. В первом случае $p = 15$ не простое, во втором случае $n = 9$. (в наших рассуждениях n натуральное, p простое число)

Ответ: 81

Задача 2.1.2. (3 балла)

Вася и Петя живут в одном доме у прямой автодороги. Чтобы сесть в школьный автобус, они одновременно вышли из дома, Петя пошел в сторону остановки А навстречу автобусу со скоростью 6 км/час, а Вася в сторону остановки В по направлению движения автобуса со скоростью 4 км/час. Они оба подошли к остановкам как раз к моменту прибытия автобуса. Найдите отношение расстояния пройденного Петей к расстоянию пройденного Васей, если скорость автобуса 60 км/час. Результат округлите до сотых.

Решение

Пусть Петя шел к своей остановке t часов и прошел расстояние $6t$ км. В момент, когда автобус подобрал его в остановке В, Вася находился на расстоянии $10t$. Автобус догоняет Васю со скоростью 56км/час и догонит через $\dfrac{10t}{56}$ часов. Вася к этому моменту пройдет $4t + \dfrac{4 \cdot 10t}{56} = \dfrac{66}{14}t$ километров. Отношение расстояний тогда равно $6t : \dfrac{66}{14}t = \dfrac{14}{11} = 1,(27)$.

Ответ: 1,27

Задача 2.1.3. (3 балла)

В автомат для размена денег загружены монеты в 2, 5 и 10 рублей. Сколько всего способов разменять 100 рублевую купюру в этом автомате?

Решение

Пусть x, y, z количество монет в 2, 5 и 10 рублей соответственно. Тогда $2x+5y+10z = 100$. Видим, что $x = 5k$, $y = 2n$, иначе уравнение не имеет решения в целых числах. Осталось найти количество неотрицательных целых решений уравнения $k + n + z = 10$. А это число сочетаний с повторениями $\overline{C}_3^{10} = C_{12}^{10} = 66$.

Ответ: 66

Задача 2.1.4. (3 балла)

На доске были написаны n первых натуральных чисел $(1, 2, \ldots n)$. Вася стер одно число. Петя заметил, что среднее арифметическое оставшихся стало $\dfrac{45}{4}$. Какое число стер Вася?

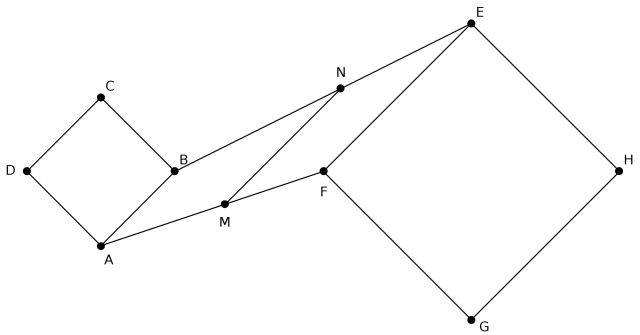
Решение

Из условия ясно, что $n > 1$. Пусть стерли число x. Тогда $x = \frac{n(n+1)}{2} - \frac{45}{4}(n-1) = \frac{2n^2 - 43n + 45}{4} = \frac{(2n-41)(n-1)}{4} + 1$. По условию $1 \leqslant x \leqslant n$. Отсюда $0 \leqslant \frac{(2n-41)(n-1)}{4} \leqslant n-1$, $0 \leqslant 2n - 41 \leqslant 4$. Решением неравенства являются только $n = 21$ или $n = 22$. При $n = 22$ число x не является целым, при $n = 21$ $x = 6$.

Ответ: 6

Задача 2.1.5. (4 балла)

На рисунке внизу $ABCD$ и $EFGH$ квадраты, их площади равны 32 см^2 и 66 см^2 соответственно. Отрезок MN параллелен основаниям трапеции $ABEF$ и делит её площадь пополам. Найдите длину отрезка MN.



Решение

Пусть прямые AB и EF пересекаются в точке K. Тогда $2S_{KMN} = S_{KBE} + SKAF$. Отсюда $2 = \frac{S_{KBE}}{S_{KMN}} + \frac{S_{KAF}}{S_{KMN}} = \frac{BE^2}{MN^2} + \frac{AF^2}{MN^2} = \frac{98}{MN^2}$.

Ответ: 7

2.2. Задачи по математике (10-11 класс)

Задача 2.2.1. (2 балла)

На плоскости отмечены 17 точек, не лежащих на одной прямой (т.е. найдутся хотя бы 3 точки, не лежащие на одной прямой). Через каждые две точки провели прямую. Какое наименьшее количество различных прямых могло получиться?

Решение

Пусть на одной прямой x точек, а на другой — $k-x$ точек (не считая общую). Прямые, проходящие через точки на различных прямых, не совпадают. Поэтому их $x(k-x)$. Эта величина уменьшается когда x стремится к 0 или k. Если точки перенесем на одну из отмеченных прямых, то количество прямых будет наименьшим. Все точки, кроме одной, лежат на одной прямой. Тогда количество раздичных прямых равно 17.

Ответ: 17

Задача 2.2.2. (3 балла)

Для проведения олимпиады преподаватели разбивают 60 школьников следующим образом: список в алфавитном порядке разбивается на 4 части, первая идет в первую аудиторию, вторая — во вторую и т. д. При этом в каждую аудиторию отправляется хотя бы один школьник. Сколькими способами можно произвести распределение?

Решение

В списке надо провести 3 разделительные черты. Эти черты можно проводить в промежутках между фамилиями. Надо выбрать 3 промежутка из 59 и поставить там разделительную черту. Всего способов $C_59^3 = \dfrac{59!}{3!56!}$.

Ответ: 32509

Задача 2.2.3. (3 балла)

Имеется таблица 100×100 натуральных чисел, в которой строки пронумерованы сверху вниз числами от 1 до 100. В k-ой строке таблицы слева направо написаны числа, которые являются арифметической прогрессией с первым членом 1 и разностью k. Какое наибольшее число встречается среди элементов диагонали, идущей из левой нижней клетки в правую верхнюю?

Решение

На отмеченной диагонали окажется $(101-k)$-ый член арифметической прогрессии, находящейся на k-ой строке. Тогда $x_{101-k} = 1+(100-k)k = 2501-(x-50)^2 \leqslant 2501$.

Ответ: 2501

Задача 2.2.4. (3 балла)

Бочка в виде прямого кругового цилиндра полностью заполнена водой. Радиус основания 40 см, а высота 120 см; бочка стоит на горизонтальной поверхности. На

какой угол необходимо наклонить эту бочка, чтобы вылить ровно половину воды. В ответ напишите тангенс этого угла, если необходимо, округлите до сотых.

Решение

Любая плоскость, проходящая через центр цилиндра, делит её на 2 равные части. Поэтому бочку надо наклонить так, чтобы плоскость, проходящая через точку на границе верхнего основания и центр цилиндра, стала горизонтальной. Тогда угол наклона станет равным углу между этой плоскостью и основанием. Тангенс этого угла равен отношению высоты бочки на диаметр основания, т.е. $\frac{120}{80}$.

Ответ: 1,5

Задача 2.2.5. (4 балла)

Найдите наибольшее натуральное значение n такое, что 120! делится без остатка на 12^n.

Решение

В 120! простое число p входит в степени $\left[\frac{120}{p}\right] + \left[\frac{120}{p^2}\right] + \left[\frac{120}{p^3}\right] + \ldots$. Таким образом находим степени 2 и 3 входящие в разложение 120! на простые множители:

$$120! = 2^{60+30+15+7+3+1} \cdot 3^{40+13+4+1} \cdot A = 12^{58} \cdot A,$$

где A не делится на 12.

Ответ: 58

2.3. Задачи по информатике

Задача 2.3.1. (1 балл)

Вам дана шахматная доска размером 8×8, в поле a которого находится король. Определите, за какое минимальное количество ходов король может добраться до клетки b.

Шахматный король может переместиться на любое соседнее поле, с которым есть хотя бы одна общая точка.

Формат входных данных

Первая строка входных данных содержит два числа a_v, a_h — координаты, где изначально находится король ($1 \leqslant a_v, a_h \leqslant 8$). Вторая строка также содержит два числа b_v, b_h — координаты клетки b, в таком же формате.

Формат выходных данных

Выведите одно целое число — ответ на задачу.

Примеры

Пример №1

Стандартный ввод
2 3 5 8

Стандартный вывод
5

Решение

Для минимизации количества ходов выгодно перемещать короля по диагонали, пока не дойдем до строки b_v или столбца b_h. Это $min(|b_v - a_v|, |b_h - a_h|)$ ходов. Оставшуюся часть доходим ходами по строке/столбцу, их всего $max(|b_v - a_v|, |b_h - a_h|) - min(|b_v - a_v|, |b_h - a_h|)$. Суммируем два ответа и получаем, что ответ равен $max(|b_v - a_v|, |b_h - a_h|)$.

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
    int av, ah, bv, bh;
    cin >> av >> ah >> bv >> bh;
    cout << max(abs(av - bv), abs(ah - bh)) << '\n';
}
```

Задача 2.3.2. (2 балла)

Фёдор перешел в 11 класс, и подготовка к ЕГЭ — его приоритетная цель в этом году.

Сегодня он узнал, что такое маска подсети, IP-адрес и адрес сети. Адрес сети получается как применение *поразрядной конъюнкции* (логическая операция И) к IP-адресу и маске подсети.

Поразрядная конъюнкция — применение к соответствующим битам операции логического И. Другими словами, если соответствующие биты равны 1, результирующий двоичный разряд будет равен 1, иначе 0.

Применение поразрядной конъюнкции к маске подсети и IP-адресу выглядит так:

```
11111111.11111111.11111111.00000000 (255.255.255.0)
&
11000000.10101000.00000000.00000001 (192.168.0.1)
-----------------------------------
11000000.10101000.00000000.00000000 (192.168.0.0) <--- Адрес сети
```

Вам дана маска подсети и n IP-адресов. Ваша задача определить сколько различных адресов сети получится.

Формат входных данных

В первой строке вам дана маска сети. Во второй строке дано числа n — количество IP-адресов ($1 \leqslant n \leqslant 1000$). Следующие n строк содержат IP-адреса.

Формат выходных данных

Выведите количество различных адресов сети.

Примеры

Пример №1

Стандартный ввод
255.255.255.0 3 192.168.0.1 192.168.0.28 192.168.1.1

Стандартный вывод
2

Решение

Давайте представим маску и IP-адреса как простые 32 битные числа, чтобы было удобно с ними работать.

Теперь получим все возможные адреса сети путем $\widehat{\Pi}$оразрядная конъюнкция всех IP-адресов с маской. Дальше нужно убрать одинаковый адреса сети. Давайте отсортируем все получившееся адреса сети. Дальше не трудно посчитать сколько будет различных, так как одинаковые теперь стоять друг за другом.

Пример программы

Ниже представлено решение на языке C++

```
1  #include <iostream>
2  #include <vector>
3  #include <set>
4
5  using namespace std;
6
7  int address(int p1, int p2, int p3, int p4) {
8          int data = p1;
9          data <<= 8;
10         data |= p2;
11         data <<= 8;
```

```
12          data |= p3;
13          data <<= 8;
14          data |= p4;
15          return data;
16  }
17
18  int read_ad() {
19          int m1, m2, m3, m4;
20          char c;
21          cin >> m1 >> c >> m2 >> c >> m3 >> c >> m4;
22          return address(m1, m2, m3, m4);
23  }
24
25  set <int> s;
26
27  int main() {
28          int mask = read_ad();
29          int n;
30          cin >> n;
31          for (int i = 0; i < n; i++) {
32                  int x = read_ad();
33                  s.insert(mask & x);
34          }
35          cout << s.size();
36          return 0;
37  }
```

Задача 2.3.3. (3 балла)

Юный поэт Марк хочет выпустить свой первый сборник стихотворений. Как вы знаете, рифма — одна из важных деталей любого стихотворения. И Марк хочет, чтобы во всех его стихах сохранялась рифма; но из-за недостатка опыта он еще не научился определять, имеет ли его стихотворение рифму. И вам необходимо ему помочь.

Стихотворение Марка представляет из себя n непересекающихся четверостиший. Четверостишие — 4 строки, состоящие из латинских букв и пробелов. Словом считается последовательность букв, не имеющая пробелов. Два слова имеют рифму, если не менее двух их последних букв совпадают. Две строки имеют рифму, если их последние слова рифмуются. Существует 3 способа задания рифмы в четверостишье:

1. Если независимо рифмуются строки номеров (1, 2) и (3, 4).
2. Если независимо рифмуются строки номеров (1, 3) и (2, 4).
3. Если независимо рифмуются строки номеров (1, 4) и (2, 3).

И наконец, стихотворение рифмуется, если все его четверостишья рифмуются одним и тем же способом. Получив стихотворение Марка, вам необходимо ответить, имеет ли оно рифму.

Формат входных данных

Первая строка входных данных содержит целое число $k = 4 * n$ — количество строк в стихотворении Марка ($4 \leqslant k \leqslant 1000$).

Далее следует стихотворение, в виде k строк, состоящих из латинских букв и

пробелов (ни одна строка не начинается и не оканчивается пробелом). Длина каждой строки не превышает 1000 символов.

Формат выходных данных

Необходимо вывести *Yes*, если стихотворение рифмуется, а иначе — *No*.

Примеры

Пример №1

Стандартный ввод

```
4
If bees stay at home
Rain will soon come
If they fly away
Fine will be the day
```

Стандартный вывод

```
Yes
```

Пример №2

Стандартный ввод

```
4
One two three
Let me see
I think about u
I like u
```

Стандартный вывод

```
No
```

Пример №3

Стандартный ввод

```
8
I want a tree
For shade and rest
I want a tree
Where birds can nest
So I planted
A green tree
You must see
Rest guaranteed
```

Стандартный вывод

```
No
```

Решение

Для того, чтобы понять рифмуются ли две строки, необходимо всего лишь сравнить на равенство по два последних символа строки(не должно быть пробелов). Теперь можно для каждого типа рифмы проверить, поддерживается ли он в каждом четверостишии стихотворения.

Пример программы

Ниже представлено решение на языке Python3

```
k = int(input())
k = k // 4

def is_ryf(s1, s2):
        if len(s1[-1]) < 2 or len(s2[-1]) < 2:
                return False

        if (s1[-1][-2:] == s2[-1][-2:]):
                return True
        return False

t1 = True
t2 = True
t3 = True

for i in range(0, k):
        s1 = input().split()
        s2 = input().split()
        s3 = input().split()
        s4 = input().split()

        if not(is_ryf(s1, s2) and is_ryf(s3, s4) and t1):
                t1 = False
        if not(is_ryf(s1, s3) and is_ryf(s2, s4) and t2):
                t2 = False
        if not(is_ryf(s1, s4) and is_ryf(s2, s3) and t3):
                t3 = False

if t1 or t2 or t3:
        print("Yes")
else:
        print("No")
```

Задача 2.3.4. (4 балла)

Вам дан невырожденный треугольник и точка. Ваша задача состоит в том, чтобы определить периметр видимой части треугольника из данной точки.

Гарантируется, что точка находится строго вне треугольника.

Формат входных данных

В первой строке вам дано 6 чисел: $(x_1; y_1)$, $(x_2; y_2)$ и $(x_3; y_3)$ — точки образующие треугольник соответственно.

На следующей строке дано 2 числа — координаты точки $(x; y)$.

Все координаты точек целые и не превосходят по модулю 10^5.

Формат выходных данных

Вывести одно число: периметр видимой части треугольника из точки $(x; y)$.

Ответ будет засчитан, если он отличается от правильного не более чем на 10^{-6} относительной или абсолютной погрешности.

Примеры

Пример №1

Стандартный ввод
0 1 1 0 1 1 2 2
Стандартный вывод
2.0000000000

Пример №2

Стандартный ввод
0 1 1 0 1 1 2 1
Стандартный вывод
1.0000000000

Пример №3

Стандартный ввод
0 0 0 1 1 0 1 1
Стандартный вывод
1.4142135624

Решение

Если мы зафиксируем какой-нибудь отрезок треугольника, то когда его будет видно ? Логично, что его будет видно, если точка, из которой мы смотрим, находится по другую сторону прямой, образованной этим отрезком, для третьей точки треугольника.

Осталось научиться проверять, лежат ли две точки по разные стороны от прямой. Это можно делать с помощью длины векторного произведения. Для удобства введем обозначения: t_1, t_2, t_3 — точки треугольника, p — точка, из которой смотрим. Возьмем длину векторного произведения c_1 для векторов из точки t_1 в точку t_2 и из t_1 в точку p. Аналогично c_2 для векторов из точки t_1 в точку t_2 и из t_1 в точку t_3. Если знаки c_1 и c_2 отличаются, то этот отрезок виден из точки, и нужно прибавить к ответу длину этого отрезка.

Пример программы

Ниже представлено решение на языке C++

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>

using namespace std;

typedef long long ll;

struct Point {
        int x, y;
        Point() {}
        Point(int x, int y) : x(x), y(y) {}
};

Point operator-(Point &p1, Point &p2) {
        return Point(p2.x - p1.x, p2.y - p1.y);
}

istream & operator >>(istream &is, Point &p) {
        is >> p.x >> p.y;
        return is;
}

ostream & operator <<(ostream &os, Point &p) {
        os << p.x << ' ' << p.y;
        return os;
}

ll crossproduct(Point const &p1, Point const &p2) {
        return (ll)p1.x * p2.y - p2.x * p1.y;
}

int side(Point &p1, Point &p2, Point &p) {
        ll cross = crossproduct(p2 - p1, p - p1);
        if (cross == 0) return 0;
        if (cross > 0) return 1;
        else return -1;
}

double dis(Point &p1, Point &p2) {
        return sqrt(((double)p2.x - p1.x) * (p2.x - p1.x) +
                    ((double)p2.y - p1.y) * (p2.y - p1.y));
}

int main() {
        vector <Point> tri(3);
        for (auto &i : tri) cin >> i;
        Point p;
        cin >> p;
        double ans = 0;
        for (int i = 0; i < 3; i++) {
        if (side(tri[i], tri[(i + 1) % 3], p) *
            side(tri[i], tri[(i + 1) % 3], tri[(i + 2) % 3]) == -1)
                ans += dis(tri[i], tri[(i + 1) % 3]);
        }
```

```
        cout << fixed << setprecision(15) << ans;
    return 0;
}
```

Задача 2.3.5. (5 баллов)

Программист Искандер уже 14-е сутки подряд пишет программу, и вот, наконец-то, весь проект был готов. Однако, скомпилировав и запустив программу, результат выполнения оказался не таким, как ожидал Искандер...

Искандер — опытный программист, и для удобства он разделил программу на n исходных файлов, каждый из которых мог использовать некоторые из остальных исходных файлов (для простоты он пронумеровал файлы целыми числами от 1 до n). После завершения выполнения программы Искандер тут же осознал, в каких k исходных файлах допущены ошибки. Исправив всё, что нужно, теперь осталась только перекомпиляция, но в повторной компиляции нуждались не только исправленные исходные файлы, но и те файлы, которые непосредственно или через другие файлы используют какие-либо файлы из изменённых.

Искандер — ленивый программист, поэтому он хочет перекомпилировать как можно меньше исходных файлов. Помогите ему посчитать количество таких файлов.

Вспоминая, что Искандер – опытный программист, можно гарантированно сказать, что он не допустил ситуации, когда один из исходных файлов использует самого себя (непосредственно или через другие файлы).

Формат входных данных

В первой строке задано целое число n ($1 \leqslant n \leqslant 5000$).

Далее идут n строк. Каждая следующая i-я строка начинается с числа p – количество исходных файлов, которые используются в i-м файле ($0 \leqslant p \leqslant n-1$). Затем следуют p различных целых чисел – номера исходных файлов, которые используются в i-м файле. Все числа положительные и не превосходят n. Суммарное количество p по всем строкам не превышает 10^5.

Затем задано целое число k ($1 \leqslant k \leqslant n$). Следующая строка содержит k различных целых чисел – номера файлов, которые исправил Искандер. Все числа положительные и не превосходят n.

Формат выходных данных

Выведите одно число — минимальное количество файлов, требующих перекомпиляции.

Примеры

Пример №1

Стандартный ввод
5 2 2 3 0 1 4 0 0 2 2 4

Стандартный вывод
4

Решение

Представим зависимости файлов между собой как ориентированный граф. Вершина с номером i будет соответствовать файлу с таким же номером. Если в текущем файле i используется файл j, то добавляем ребро из вершины с номером j в вершину с номером i. Теперь рассмотрим, какие из файлов надо перекомпелировать: запустим обход в глубину от каждой из k вершин, соответствующих изменённым файлам, и будем помечать все вершины на пути. Сделаем небольшую оптимизацию: не будем запускать обход в глубину из уже помеченных вершин. Таким образом, каждая вершина будет посещена не более 1 раза. Ответом на задачу является количество помеченных в графе вершин.

Пример программы

Ниже представлено решение на языке Python3

```
used = [];
v = [];

def dfs(x):
    used[x] = True
    for i in range(len(v[x])):
        to = int(v[x][i])
        if used[to] == False:
            dfs(to)

n = int(input())

v = [[] for i in range(n)]
used = [False] * n

for i in range(n) :
    p = input().split();
    for j in range(len(p)) :
        p[j] = int(p[j]) - 1
    for j in range(1, len(p)):
        v[p[j]].append(i);

k = int(input())

if k > 0:
    edit = input().split()
```

```
    for i in range(k):
        edit[i] = int(edit[i]) - 1

    for i in range(k):
        dfs(edit[i])

ans = 0;

for i in range(n):
    if used[i] == True:
        ans = ans + 1;

print(ans);
```

Вторая попытка

3.1. Задачи по математике (9 класс)

Задача 3.1.1. (2 балла)

С числом можно проделавать 2 типа операции: операция A умножает число на 2, операция B вычитает 3. Как из 11 при помощи этих операций получить 25 за наименьшее количество ходов? В ответ напишите последовательность операций (например последовательность операций AAABBBBBBBBBBBBBBBBBBBB за 24 хода достигает результата).

Решение

Обозначим M_n множество чисел, из которых можно получить 25 за n ходов. Тогда $M_1 = \{28\}$, $M_2 = \{14, 31\}$, $M_3 = \{7, 17, 34\}$, ... На 7-ом шаге встречаем 11.

Ответ: BBABAAB

Задача 3.1.2. (4 балла)

При каком наименьшем значении параметра k корни уравнения

$$x^2 + (k-1)x - k = 0$$

удовлетворяют условию $x_1^2 + x_2^2 = 5$.

Решение

$x_1^2 + x_2^2 = (x_1 + x_2)^2 - 2x_1x_2 = (k-1)^2 + 2k = 5$. Откуда $k = \pm 2$.

Ответ: -2

Задача 3.1.3. (3 балла)

В выпуклом пятиугольнике $ABCDE$ известно, что $AB = AE = 3$, $CD = 2$, $BC = 0,8$, $DE = 1,2$, $\angle ABC = \angle DEA = 90^\circ$. Найдите площадь этого пятиугольника.

Решение

Разделим пятиугольник на 3 треугольника ABC, ADE и ACD. Площади первых двух треугольников находяться по уже известным данным, поскольку треугольники прямоугольные, и известно две их стороны.

Стороные AC и AD треугольников прямоугльных треугольников ABC и ADE находятся также по уже известным данным. Зная все три стороны треугольника ACD, можно найти его площадь.

Останется только сложить площади всех трех треугольников.

Ответ: 6

Задача 3.1.4. (3 балла)

Натуральное число n называется «приятным», если оно удовлетворяет следующим условиям:

- состоит из 4 цифр;
- первая цифра равна третьей;
- вторая цифра равна четвертой;
- n^2 делится на произведение цифр n.

Найдите сумму всех «приятных» чисел.

Решение

$n = \overline{abab} = \overline{ab} \cdot 101$, $n^2 \vdots (a \cdot b \cdot a \cdot b) \Longrightarrow n \vdots ab$. н.о.д.$(101, ab) = 1 \Longrightarrow \overline{ab} \vdots ab$. Отсюда получаем, что $(10a + b)$ делится на a, т.е. $b \vdots a$. Небольшим перебором находим все значения $\overline{ab}$: 11, 12, 15, 24, 36. Сумма всех «приятных» чисел равна $(11 + 12 + 15 + 24 + 36) \cdot 101$.

Ответ: 9898

Задача 3.1.5. (4 балла)

Найдите количество решений в целых числах уравнения

$$\frac{1}{x}+\frac{1}{y}=\frac{1}{2017}.$$

Решение

Уравнение приводится к виду $y = 2017 + \dfrac{2017^2}{x-2017}$. Тогда $x - 2017$ является целым делителем 2017^2. Таких значений шесть: ± 1, ± 2017, $\pm 2017^2$. В каждом случае получаем пару целых решений.

Ответ: 6

3.2. Задачи по математике (10-11 класс)

Задача 3.2.1. (2 балла)

Найдите наименьшее число, которое можно представить в виде суммы квадратов натуральных чисел двумя различными способами.

Решение

$65 = 4^2 + 7^2 = 1^2 + 8^2$. Перебирая все суммы квадратов, меньших 64, выясняем минимальность.

Ответ: 65

Задача 3.2.2. (3 балла)

Найдите значение выражения

$$[2\sqrt{1\cdot 2\cdot 3+2\cdot 3\cdot 4+\ldots+2016\cdot 2017\cdot 2018}].$$

($[x]$ — целая часть числа x.)

Решение

$1\cdot 2\cdot 3+2\cdot 3\cdot 4+\ldots+2016\cdot 2017\cdot 2018=\frac{1}{4}\cdot 2016\cdot 2017\cdot 2018\cdot 2019=4141845698256$. Остальное вычисляем на калькуляторе.

Ответ: 4070304

Задача 3.2.3. (3 балла)

На координатной плоскости отмечены точки $A(2;-1)$, $B(3;1)$ и $C(2017;4034)$. Найдите площадь треугольника ABC.

Решение

Пусть $O(0;0)$. Тогда $\overrightarrow{AB}\{1;2\} \upuparrows \overrightarrow{OC}\{2017;4034\}$. Поэтому площадь $\triangle ABC$ равна площади $\triangle OAB$.

Ответ: 2,5

Задача 3.2.4. (3 балла)

Обозначим $\tau(n)$ количество различных натуральных делителей числа n, а $\sigma(n)$ сумму различных натуральных делителей числа n. Найдите $\tau(\sigma(12)!)$.

Решение

$\sigma(12) = (1+2+4)\cdot(1+3) = 28$, $28! = 2^{25}\cdot 3^{13}\cdot 5^{6}\cdot 7^{4}\cdot 11^{2}\cdot 13^{2}\cdot 17\cdot 19\cdot 23$, $\tau(28!) = 26\cdot 14\cdot 7\cdot 5\cdot 3\cdot 3\cdot 2\cdot 2\cdot 2$.

Ответ: 917280

Задача 3.2.5. (4 балла)

Найдите все значения параметра p, при которых уравнение

$$27x^3 + (26-2p)x^2 + 16p^3 = 0$$

имеет три различных корня, образующих арифметическую прогрессию. В ответ напишите сумму полученных значений.

Решение

По теореме Виета для кубического многочлена:

$$\begin{cases} x_1 + x_2 + x_3 = -\dfrac{26-2p}{27} \\ x_1x_2 + x_2x_3 + x_3x_1 = 0 \\ x_1x_2x_3 = -\dfrac{16p^3}{27} \end{cases}$$

В эту систему добавляем условие арифметической прогрессии $x_1+x_3 = 2x_2$ и находим единственное решение.

Ответ: -0,5

3.3. Задачи по информатике

Задача 3.3.1. (1 балл)

Никита работает на автостоянке. В его обязанности входит запись номеров въезжающих машин. Это довольно скучное занятие и поэтому Никита решил оптимизировать этот процесс. Он хочет, чтобы компьютер обрабатывал изображение с камеры перед въездом в автостоянку и записывал номера. Никита уже написал софт, который обнаруживает на изображении последовательности из шести символов; осталось только проверять, является ли данная последовательность номером. Но из-за того, что Никита постоянно отвлекается на запись номеров, он просит вас о помощи.

Автомобильный номер – строка из шести символов. Первый символ – заглавная латинская буква, далее следует 3 цифры, и после — две заглавные латинские буквы. Например, строка "P142EQ" является номером. Вам будет дана строка, состоящая из шести символов, необходимо ответить, является ли строка автомобильным номером.

Формат входных данных

В единственной строке находится строка из шести символов, состоящая из цифр и заглавных латинских букв.

Формат выходных данных

Если строка является автомобильным номером, то необходимо вывести *Yes*, в ином случае — *No*.

Примеры

Пример №1

Стандартный ввод
`K040LE`

Стандартный вывод
`Yes`

Пример №2

Стандартный ввод
`M3239L`

Стандартный вывод
`No`

Решение

Для того, чтобы решить задачу необходимо проверить, что на первой и двух последних позициях строки находятся заглавные латинские буквы ($'A' \leqslant c \leqslant' Z'$); а между буквами находились три числа ($'0' \leqslant c \leqslant' 9'$).Для того, чтобы решить задачу необходимо проверить, что на первой и двух последних позициях строки находятся заглавные латинские буквы ($'A' \leqslant c \leqslant' Z'$).

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>
#include <string>

using namespace std;

bool is_dig(char c) {
        if ('0' <= c && c <= '9') {
                return true;
        }
        return false;
}

bool is_let(char c) {
        if ('A' <= c && c <= 'Z') {
                return true;
        }
        return false;
}

int main() {
        string s;
        cin >> s;
        if (is_let(s[0]) && is_dig(s[1]) && is_dig(s[2])
           && is_dig(s[3]) && is_let(s[4]) && is_let(s[5])) {
                cout << "Yes\n";
        } else {
                cout << "No\n";
        }
}
```

Задача 3.3.2. (1 балл)

Студент Денис часто прогуливает свои пары в университете. Сегодня, когда Денис пришел в деканат, он узнал, что нужно за каждую пропущенную пару написать объяснительную.

Денису выдали один бланк для написания объяснительной. Ему нужно сделать еще n копий данного бланка. Для этого в его распоряжении есть два ксерокса. Первый ксерокс тратит на копирование одного листа одну секунду, а второй — две секунды. Копию можно делать как с оригинала, так и с копии. Денис может использовать оба ксерокса одновременно. Определите, какое минимальное время необходимо Денису для получения n копий объяснительной.

Формат входных данных

В единственной строке дано число n — необходимое число копий ($1 \leqslant n \leqslant 1000$).

Формат выходных данных

Выведите одно число — минимальное время в секундах, необходимое для получения n копий.

Примеры

Пример №1

Стандартный ввод
2

Стандартный вывод
2

Пример №2

Стандартный ввод
5

Стандартный вывод
4t

Решение

Изначально нужно скопировать один экземпляр, чтобы одновременно копировать на обоих ксероксах. После этого для печати трех экземпляров необходимо 2 минуты. Получается, что для печати $n-1$ копий необходимо затратить $(n-1)//3*2$ минут в случае, если $n-1$ делится на 3. В случае остатка от деления на 3, равному k, необходимо дополнительно потратить k минут. Таким образом, итоговая формула выглядит, как $1+(n-1)//3*2+(n-1)\%3$.

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>

using namespace std;

int main() {
        int n;
        cin >> n;
        cout << 1 + (n - 1) / 3 * 2 + (n - 1) % 3<< "\n";
}
```

Задача 3.3.3. (3 балла)

Сегодня Ринат на уроке узнал про новую битовую операцию – XOR двух чисел. Напомним, что XOR или Исключающим ИЛИ, называется бинарная операция, которая применяется к каждой паре битов, стоящих на одинаковых позициях в двоичных представлениях чисел так, что, если биты равны, то в результате на этой позиции стоит 0, если же биты различны, то 1. Например, 3 XOR 5 = 6, потому что $3_{10} = 011_2$, $5_{10} = 101_2$, поэтому после применения операции второй и третий бит становятся равными 1, а первый бит – 0, таким образом получается $110_2 = 6_{10}$.

Учитель дал Ринату массив различных чисел и попросил написать массив, в котором все числа различны и нет ни одного числа из массива, который дал учитель. При этом, чтобы XOR всех чисел в массиве Рината был равен XOR всех чисел в массиве учителя.

Формат входных данных

В первой строке дано число n — количество элементов в массиве учителя ($1 \leqslant n \leqslant 10^5$).

Во второй сроке дано n чисел a_i — элементы массива ($0 \leqslant a_i \leqslant 2^{30}$).

Гарантируется, что XOR всех a_i не равен 0.

Формат выходных данных

Выведите элемента вашего массива, чтобы все числа в вашем массиве и массиве учителя отличались.

Все числа вашего массива b_i должны находиться в промежутке ($0 \leqslant b_i \leqslant 2^{30}$).

Количество элементов не должно превосходить 10^5.

Примеры

Пример №1

Стандартный ввод
1 27

Стандартный вывод
21 14

Пример №2

Стандартный ввод
2 5 6

Стандартный вывод
7 4

Пример №3

Стандартный ввод
3 1 2 4

Стандартный вывод
5 3 9 8

Пример №4

Стандартный ввод
3 3 5 1

Стандартный вывод
10 9 6 2

Решение

Стоит подумать о том, какая длина массива нам нужна. Давайте подумаем, можно ли это сделать с одним элементом ? Возьмем $x = \mathbf{XOR}\ a_i$. Тогда наш ответ и есть этот x, но он может быть в массиве учителя.

Можно ли это сделать из двух элементов ? Давайте переберем число i. Тогда вторым числом должно быть i **XOR** x. Если число i и число i **XOR** x не были в массиве учителя, то мы нашли ответ.

Давайте подумает за сколько итераций завершится цикл, если просто перебирать i увеличивая на 1. А цикл завершится не больше, чем за $2 \cdot n + 1$ итераций. Потому что все числа разбиваются на пары i и i **XOR** x. Всего n элементов могут покрыть не более n пар, поэтому $2 \cdot n$ чисел i могут не подойти.

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>
#include <vector>
#include <set>


using namespace std;


set <int> s;

bool contains(int n) {
    return s.find(n) != s.end();
}

int main() {
        int n;
        cin >> n;
        int now = 0;
        vector <int> v(n);
        for (auto &i : v) {
            cin >> i;
            s.insert(i);
            now ^= i;
        }
        for (int i = 0; ; i++) {
            if (i != (i ^ now) && !contains(i) && !contains(i ^ now)) {
                cout << i << ' ' << (i ^ now) << '\n';
                return 0;
            }
        }
        return 0;
}
```

Задача 3.3.4. (5 баллов)

Азат уже очень давно отдыхает на островах. Сегодня он решил, что хочет перебраться на другой остров. Все острова пронумерованы целыми числами. Какие-то из островов соединены мостами. Но вот незадача, какие-то из островов ушли под воду и попасть на них никак нельзя. Азат сейчас находится на острове с номером v и хочет попасть на остров с номером u. Ваша задача проверить, сможет ли он это сделать.

Формат входных данных

В первой строке содержится 4 числа: n, m, v, и u ($1 \leqslant n \leqslant 1000, 0 \leqslant m \leqslant 5000, 1 \leqslant v, u \leqslant n$). Количество островов, количество мостов, номер острова, на котором находится Азат, и номер острова, на который он хочет попасть.

Следующие m строк описывают мосты. Каждая строка содержит 2 числа: a, b ($1 \leqslant a, b \leqslant n$). Мост между островами с номерами a и b соответственно. По мосту можно попасть с острова a на b, так и с острова b на a.

Следующая строка содержит одно число k ($0 \leqslant k \leqslant n$). Количество островов, которые затонули.

Следующая строка содержит k чисел c_i ($1 \leqslant c_i \leqslant n$) — номера затонувших островов. Гарантируется, что острова u и v не затоплены. Если k равно 0, то данная строка отсутствует.

Формат выходных данных

Если можно добраться от острова v до острова u, не посещая затопленные острова, то необходимо вывести *YES*, в ином случае — *NO*.

Примеры

Пример №1

Стандартный ввод

```
4 4 1 4
1 2
2 3
3 4
1 3
1
2
```

Стандартный вывод

```
YES
```

Пример №2

Стандартный ввод
5 6 1 5 1 2 2 3 3 4 4 5 1 3 2 4 2 2 3

Стандартный вывод
NO

Решение

Давайте применим известный алгоритм поиска в глубину(DFS). Будем хранить массив $used[i]$, где будем стоять 1, если мы были на острове с номером i, или он затоплен, и 0, если мы там не были.

Изначально пометим все затопленные острова в массиве $used$ как 1. Потом запустим DFS от острова, в котором стоит Азат и будем пытаться попасть на остров, с которым мы связаны мостом, но мы не были на нем, и он не затоплен. Если таки образом мы сможем попасть на остров u, то ответ YES, иначе NO.

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>
#include <vector>

using namespace std;

int const MAX = (int)1e5;

bool used[MAX];

void dfs(int v, vector < vector <int> > &vv) {
        used[v] = true;
        for (auto i : vv[v]) {
                if (used[i]) continue;
                dfs(i, vv);
        }
}

int main() {
        int n, m, a, b;
        cin >> n >> m >> a >> b;
        --a; --b;
        vector < vector <int> > vv(n);
        for (int i = 0; i < m; i++) {
                int f, t;
                cin >> f >> t;
                --f; --t;
                vv[f].push_back(t);
```

```
28                  vv[t].push_back(f);
29          }
30          int k;
31          cin >> k;
32          for (int i = 0; i < k; i++) {
33                  int x;
34                  cin >> x;
35                  used[x - 1] = true;
36          }
37          dfs(a, vv);
38          used[b] ? cout << "YES\n" : cout << "NO\n";
39          return 0;
40  }
```

Задача 3.3.5. (5 баллов)

На прямой отмечено n точек. Нужно соединить некоторые точки отрезками одинаковой длины, чтобы можно было по отрезкам дойти от первой точки до последней, причём можно использовать не более k отрезков. Среди всех таких длин отрезков требуется вывести минимальную.

Формат входных данных

В первой строке заданы целые числа n и k ($2 \leqslant n \leqslant 2500$, $1 \leqslant k \leqslant n-1$).

На следующей строке в возрастающем порядке заданы n различных неотрицательных целых чисел, каждое из которых не превосходит 100 000.

Формат выходных данных

Выведите одно целое число — ответ на задачу.

Примеры

Пример №1

Стандартный ввод
5 3 1 2 3 4 5

Стандартный вывод
2

Пример №2

Стандартный ввод
9 3 1 3 4 5 7 8 9 10 13

Стандартный вывод
4

Решение

Перебираем все возможные длины отрезка: отрезок длиной равной расстоянию от первой точки до второй, от первой точки до третьей и т.д. Для каждого такого отрезка пробуем соединить некоторые точки: начинаем с первой точки и проверяем, существует ли точка на позиции, равной сумме координаты текущей точки и длины рассматриваемого отрезка. Если такой точки не существует, то соединить точки отрезком текущей длины невозможно, иначе переходим к точке, существование которой мы проверяли. Если таким образом мы смогли дойти до последней точки, и при этом было использовано не более k отрезков, то текущая длина и является ответом.

Пример программы

Ниже представлено решение на языке Python3

```
A = input().split();
n = int(A[0]);
k = int(A[1]);

x = input().split();

for i in range(n) :
    x[i] = int(x[i]);

for i in range(1, n) :
    len = int(x[i]) - int(x[0]);
    cnt = 0;
    now = x[0];
    while (now != x[n - 1]) :
        now += len;
        if now not in x:
            cnt = -1;
            break;
        cnt = cnt + 1;
    if cnt != -1 and cnt <= k :
        print(len);
        break;
```

Третья попытка

4.1. Задачи по математике (9 класс)

Задача 4.1.1. (2 балла)

Имеется 4 попарно различных по весу камней и двухчашечные весы без гирь. За одно взвешивание можно положить по камню на каждую из чаш и узнать какой из

камней более тяжелый. За какое наименьшее число взвешиваний можно упорядочить камни по возрастанию?

Решение

Пусть имеются камни весами a, b, c, d. Если взвешивать все пары камней, то получится 6 взвешиваний. Надо предсказать результат одного взвешивания и нам не придется его совершать. Взвешиваем a с b и c с d. Пусть a и c окажутся более тяжелыми. Еще за одно взвешивание из них выберем более тяжелый. Тогда этот камень не надо взвешивать с более легким из другой пары камней (очевидно он тяжелей).

Ответ: 5

Задача 4.1.2. (2 балла)

Чему равен наибольший общий делитель пары чисел 7808809 и 7182391?

Решение

Используем алгоритм Евклида.

Ответ: 2677

Задача 4.1.3. (3 балла)

Три генератора тактовой частоты начинают вырабатывать импульсы одновременно. Интервалы между импульсами составляют 4/3 секунды, 5/3 секунды и 2 секунды соответственно. Совпавшие во времени импульсы накладываются (обрабатываются компьютером за один). Сколько тактов будет обработано за 1 минуту? (Первый импульс также считается.)

Решение

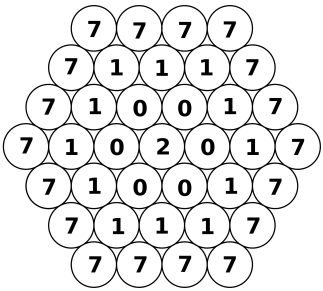
Первый генератор выработает 45 импульсов, второй — 36, третий — 30. Но мы посчитали совпадающие несколько раз: попарно совпадающие по два раза, все три совпадения по три раза. У первого и второго генератора совпадение импульсов каждые 20/3 секунды, у первого и третьего — каждые 4 секунды, у второго и третьего — каждые 10 секунд. Тогда попарное совпадение импульсов $9 + 15 + 6$ раз. В эту сумму тройное совпадение вошло 3 раза. Все три импульса совпадают $60 : \frac{60}{3} = 3$ раза. Общее количество импульсов за минуту равно:

$$45 + 36 + 30 - (9 + 15 + 6) + 3 = 84.$$

Ответ: 85

Задача 4.1.4. (4 балла)

Требуется выбрать 4 круга с цифрами 2, 0, 1 и 7 на рисунке ниже так, чтобы круг с цифрой 2 касался круга с 0, а круг с 0 касался круга с 1, круг с 1 касался круга с 7. Сколько способов это сделать?



Решение

Рассмотрим круги с 1. Если брать 1 из углового круга, то к нему от 2 ведет только 1 путь и от него можно пройти к 7 ровно 3 способами. Таких кругов с 1 ровно 6, поэтому всего путей через угловые 1 ровно 18. Если же брать круги с 1 на середине стороны шестиугольника, то к нему от 2 можно попасть через 2 нулика, а от него к 7 можно пройти тоже 2 способами. Таких путей всего $6 \cdot 2 \cdot 2 = 24$.

Ответ: 42

Задача 4.1.5. (4 балла)

Из одной точки окружности проведены две хорды длиной 9 и 17. Найдите радиус окружности, если расстояние между серединами хорд равно 5.

Решение

Расстояние между другими концами этих хорд равно 10 (по теореме о средней линии). Теперь надо выяснить радиус окружности, описанной вокруг треугольника со сторонами 9, 10, 17. Его можно выяснить по формулам $R = \dfrac{abc}{4S}$, и $S = \sqrt{p(p-a)(p-b)(p-c)}$.

Ответ: 10,625

4.2. Задачи по математике (10-11 класс)

Задача 4.2.1. (2 балла)

Имеется 5 попарно различных по весу камней и двухчашечные весы без гирь. За одно взвешивание можно положить по камню на каждую из чаш и узнать какой из камней более тяжелый. За какое наименьшее число взвешиваний можно упорядочить камни по возрастанию?

Решение

Пусть веса наших камней равны a, b, c, d, e. Первыми двумя взвешиваниями сравниваем пары (a, b) и (c, d). Не умаляя общности можно считать, чтобы $a > b$ и $c > d$. Третьим взвешиванием сравниваем пару (a, c). Не умаляя общности можно считать, что $a > c > d$ и $a > b$. За следующие 2 взвешивания вставляем камень e в цепочку $a > c > d$. А так как мы еще знаем, что $a > b$, то за последние 2 взвешивания вставляем b в нашу полученную цепочку.

Ответ: 7

Задача 4.2.2. (2 балла)

При каком наибольшем n на шахматной доске можно расставить n чёрных и n белых королей так, чтобы чёрные не били белых, а белые — чёрных?

Решение

Группа белых королей от группы черных королей должна быть отделена полосой пустых клеток. Если полоса в 8 пустых клеток, то в одной из частей не более 24 королей. Для полосы в 10 клеток пример строится несложно:

б	б	б	б		ч	ч	ч
б	б	б	б		ч	ч	ч
б	б	б	б		ч	ч	ч
б	б	б			ч	ч	ч
б	б	б			ч	ч	ч
б	б	б		ч	ч	ч	ч
б	б	б		ч	ч	ч	ч
б	б	б		ч	ч	ч	ч

Ответ: 27

Задача 4.2.3. (3 балла)

К числу 2017 припишите слева и справа по одной цифре так, чтобы полученное шестизначное число делилось на 31. Какие числа получились? В ответ напишите сумму таких чисел.

Решение

$\overline{a2017b} = 100000a + 20170 + b \equiv 25a + b - 11 \equiv 0 \pmod{31}$. Откуда $b \equiv 6a + 11 \pmod{31}$. Подставляя все цифры от 1 до 9 вместо a находим b. Нам подойдут $a = 4$, $b = 4$ и $a = 9$, $b = 3$.

Ответ: 1340347

Задача 4.2.4. (4 балла)

Рассматриваются всевозможные расстояния от точки $A(28; 13)$ до точек параболы $y = x^2$. Найдите среди них кратчайшее. В ответ запишите квадрат этого расстояния.

Решение

Пусть $M(x; x^2)$. Введем функцию $f(x) = AM^2 = (x - 28)^2 + (x^2 - 13)^2 = x^4 - 25x^2 - 56x + 28^2 + 13^2$. Требуется найти наименьшее значение $f(x)$. Она достигается в точках минимума. $f'(x) = 4x^3 - 50x - 56 = 2(x - 4)(2x^2 + 8x + 7)$ имеет единственную точку минимума $x = 4$. $f(4) = 576 + 9 = 585$.

Ответ: 585

Задача 4.2.5. (4 балла)

В окружности проведены хорды AC и BD, пересекающиеся в точке E, причем касательная к окружности, проходящая через точку A, параллельна BD. Известно, что $CD : ED = 4 : 3$, $S_{\triangle ABE} = 72$. Найдите площадь треугольника ABC.

Решение

Прямая, проходящая через A перпендикулярно касательной, проходит через центр окружности. Она же перпендикулярна хорде BD, следовательно, делит её пополам. Треугольник ABD равнобедренный. Используя теорему о вписанном угле выясняем справедливость следующих равенств:

$$\angle ACD = \angle ABD = \angle ADB = \angle ACB.$$

Из этих равенств очевидно следуют подобия треугольников:

$$\triangle ABC \sim \triangle AEB \sim \triangle DCE.$$

Откуда получаем $\dfrac{AB}{AE} = \dfrac{CD}{DE} = \dfrac{4}{3}$.

$$\frac{S_{\triangle ABC}}{S_{\triangle AEB}} = \left(\frac{AB}{AE}\right)^2 = \frac{16}{9}.$$

Ответ: 128

4.3. Задачи по информатике

Задача 4.3.1. (1 балл)

Юному Тимуру нравится рисовать четырехугольники с заданными длинами сторон. Длины для сторон он придумывает сам, и после этого начинает рисовать. Недавно на уроке математики он узнал, что не всегда из придуманных сторон можно получить четырехугольник. И теперь, чтобы не терять времени, Тимур хочет быть уверен, что по его задуманным длинам можно построить четырехугольник. За помощью в этом вопросе он обращается к вам.

Тимур даст вам длины четырех сторон. Необходимо определить, возможно ли построит из заданных сторон четырехугольник.

Формат входных данных

В единственной строке входных данных находится четыре целых положительных числа, разделенных пробелом. Каждое число не превосходит 10000.

Формат выходных данных

Если из заданных длин сторон можно построить четырехугольник, то необходимо вывести *Yes*, в ином случае — *No*.

Примеры

Пример №1

Стандартный ввод
`2 1 3 2`

Стандартный вывод
`Yes`

Пример №2

Стандартный ввод
`4 10 3 2`

Стандартный вывод
`No`

Решение

Необходимое и достаточное условие для построение четырехугольника с заданными сторонами – длина максимальной стороны должна быть строго меньше суммы трех остальных. Например, если a - максимальная сторона, то условие выглядит, как $a < b + c + d$. (b, c, d – остальные стороны четырехугольника).

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
        vector <int> lines(4);
        for (int i = 0; i < 4; i++) {
                cin >> lines[i];
        }
        sort(lines.begin(), lines.end());
        if (lines[0] + lines[1] + lines[2] > lines[3]) {
                cout << "Yes\n";
        } else {
                cout << "No\n";
        }
}
```

Задача 4.3.2. (2 балла)

Из переписки двух друзей:

- Вася, привет! Не мог бы ты мне помочь?
- Привет, Настя! Да, конечно.
- Я готовлюсь к одной олимпиаде по программированию и решила немного подготовиться. Не мог бы ты дать мне какую-нибудь задачку?
- Без проблем! Вот условия: дана строка S_0. Каждая следующая строка получается добавлением в конец к предыдущей строке самого часто встречающего символа этой строки. Надо найти строку S_k. Ответов может быть несколько, поэтому надо вывести лексикографически минимальную строку из всех возможных.
- Хорошо, спасибо большое!

Помогите Насте решить задачу, которую ей дал Вася.

Формат входных данных

В первой строке заданы целые положительные числа n и k ($1 \leqslant n \leqslant 1000$, $1 \leqslant k \leqslant 1000$).

На следующей строке задана строка S_0, состоящая из n строчных букв латинского алфавита.

Формат выходных данных

Выведите лексикографически минимальную строку S_k.

Комментарии

Одна строка лексикографически меньше другой, если существует такая позиция m, что символ первой строки на позиции m меньше символа второй строки на той же позиции, а первые $m-1$ символы двух строк совпадают.

Примеры

Пример №1

Стандартный ввод
5 1 abcdc

Стандартный вывод
abcdcc

Пример №2

Стандартный ввод
4 2 baab

Стандартный вывод
baabaa

Решение

Очевидно, что если мы в конец строки добавим самый часто встречающийся символ в нём, то и в следующих строках он также будет самым часто встречающимся. Поэтому ответом является исходная строка S_0, к которой в конец k раз приписан самый часто встречающийся символ в нём. Чтобы получить лексикографически минимальную строку в ответе, нужно среди самых часто встречающихся символов в S_0 выбрать тот символ, который раньше всего идёт в алфавите.

Пример программы

Ниже представлено решение на языке Python3

```
A = input().split();
n = int(A[0]);
k = int(A[1]);

s = input();

cnt = [0] * 26;

for i in range(n) :
    cnt[ord(s[i]) - 97] = cnt[ord(s[i]) - 97] + 1;

mx = 0;

```

```
for i in range(26) :
    if (cnt[i] > cnt[mx]) :
        mx = i;

for i in range(k) :
    s = s + chr(97 + mx);

print(s);
```

Задача 4.3.3. (3 балла)

Фермер по имени Артем сегодня решил озеленить свой сад. Для удобства представим, что сад Артема представляет собой декартову плоскость, где на целых координатах он посадил розы.

Чтобы цветок расцвел необходимо его поливать, поэтому он поставил две поливалки, которые поливают все розы в определенном радиусе.

После оказалось, что некоторые розы поливаются с обоих устройств. Артему стало интересно, сколько таких роз, но так как поле очень большое, то он попросил вас помочь ему.

Формат входных данных

Первая строка содержит три целых целых числа $x1$, $y1$, $r1$ ($-10^6 \leqslant x1, y1 \leqslant 10^6, 0 \leqslant r1 \leqslant 10^6$) — координаты центра первой поливалки и её радиус действия.

Во второй в аналогичном формате координаты и радиус второй поливалки.

Формат выходных данных

Выведите количество роз, которые поливаются с двух поливалок.

Примеры

Пример №1

Стандартный ввод
0 0 5 2 2 3

Стандартный вывод
26

Решение

Переберем одну из координат. Найдем целые координаты, которые принадлежат первой окружности на координате, которую мы перебираем. Обозначим их за l_1, r_1. Аналогично сделаем для второй окружности и обозначим за l_2, r_2. Прибавим к ответу количество целых координат в пересечении этих отрезков l_1, r_1 и l_2, r_2.

Чтобы найти координаты можно воспользоваться бинарным поиском или же решить уравнение окружности, где будет одна неизвестная, однако нужно будет хорошо округлить до целых координат. От этого и будет зависеть сложность вашего решения. O(N) — для уравнение и O(NlogN) — для бинпоиска, где N — разность между максимальной и минимальной координатой.

Пример программы

Ниже представлено решение на языке Python3

```python
from math import sqrt

def calc(m, x0, x, r):
    return m * m <= r * r - (x - x0) * (x - x0)


def get_y(x0, y0, r, x):
    _l = -1
    _r = r + 2
    while _r - _l > 1:
        m = (_l + _r) // 2
        if m * m <= r * r - (x - x0) * (x - x0):
            _l = m
        else:
            _r = m
    return _l

x1, y1, r1 = map(int, input().split())
x2, y2, r2 = map(int, input().split())
ans = 0
for x in range(x1-r1, x1+r1+1):
    if x > x2 + r2 or x < x2 - r2:
        continue

    y1l = int(y1 - sqrt(r1 ** 2 - (x1 - x) ** 2)) - 10
    while ((y1l - y1) ** 2 + (x1 - x) ** 2) > r1 ** 2:
        y1l += 1

    y1h = int(y1 + sqrt(r1 ** 2 - (x1 - x) ** 2)) + 10
    while ((y1h - y1) ** 2 + (x1 - x) ** 2) > r1 ** 2:
        y1h -= 1

    y2l = int(y2 - sqrt(r2 ** 2 - (x2 - x) ** 2)) - 10
    while ((y2l - y2) ** 2 + (x2 - x) ** 2) > r2 ** 2:
        y2l += 1

    y2h = int(y2 + sqrt(r2 ** 2 - (x2 - x) ** 2)) + 10
    while ((y2h - y2) ** 2 + (x2 - x) ** 2) > r2 ** 2:
        y2h -= 1

    if not (y1h < y2l or y2h < y1l):
        # print((y1l, y1h), (y2l, y2h))
        # print(x, abs(max(y1l, y2l) - min(y1h, y2h)) + 1)
        ans += abs(max(y1l, y2l) - min(y1h, y2h)) + 1

print(ans)
```

Задача 4.3.4. (4 балла)

Гена очень послушный и умный маленький мальчик. Как и все дети он любит играть в кубики, но часто ему в голову приходят сложные математические формулы или непростые задачи. Вот и в этот раз, когда он раскладывал кубики, то придумал очень интересную задачу.

Он начал выкладывать кубики в ряд длинной n. Причем высота на i-ом месте ряда составляет a_i кубиков. Тут он и задался вопросом: какое минимальное количество кубиков надо переложить, чтобы все высоты стали равны. Кубики можно перекладывать только на соседние места, то есть, если мы взяли кубик с ряда номер j, то можно положить его на ряд номер $j+1$ или $j-1$, если такие существуют.

Формат входных данных

Первая строка содержит одно целое число n ($1 \leqslant n \leqslant 10^5$) — количетсво рядов из кубиков, которые выложил Гена.

Во второй строке дано n чисел $a_i (0 \leqslant a_i \leqslant 10^6)$ — количество кубиков в i-м номере ряда.

Формат выходных данных

Выведите минимально количество перекладываний, которое необходимо сделать Гене, чтобы все высоты стали равны, если Гена не сможет выстроить такой ряд из кубиков, выведите -1.

Примеры

Пример №1

Стандартный ввод
4 2 1 2 3

Стандартный вывод
2

Пример №2

Стандартный ввод
5 2 3 4 5 6

Стандартный вывод
10

Решение

Посчитаем количество кубиков в каждом ряду: $h = \frac{1}{n} \cdot \sum_{i=1}^{n} a_i$. Пусть $df = 0$ — количество лишних кубиков на префиксе(отрицательное число, если кубиков не хватает). Теперь рассматриваем каждый ряд от 1 до n, $df = df + (a_i - h)$ — новая разница

кубиков. В каждой итерации к ответу прибавляется $|df|$ — если у нас переизбыток кубиков, в любом случае они перейдут дальше, иначе кубики с конца должны будут перейти в начало.

Пример программы

Ниже представлено решение на языке C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n;
    long long sum = 0;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        sum += a[i];
    }
    if (sum % n != 0) {
        cout << -1 << '\n';
        return 0;
    }
    long long k = sum / n;
    long long ans = 0;
    int j = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] >= k) {
            continue;
        }
        while (a[i] != k) {
            while (j < n && a[j] <= k) {
                j++;
            }
            int t = min(k - a[i], a[j] - k);
            a[j] -= t;
            a[i] += t;
            ans += t * abs(i - j);
        }
    }
    cout << ans << '\n';
}
```

Задача 4.3.5. (5 баллов)

В крупной компании по созданию программного обеспечения находится n серверов. Они соединяются сетью с помощью m проводов. Провод соединяет два сервера между собой.

Сервера A и B находятся в одной локальной сети, если сигнал от сервера A может по рабочим проводам дойти до сервера B, возможно проходя при этом через промежуточные сервера. Если сервер может соединиться только с собой, то считается, что он сам по себе представляет локальную сеть.

В датацентре компании появились грызуны, которые начали перегрызать провода. Пока ваш напарник поехал за отпугивателями грызунов, вам поручили посчитать полученный ущерб компании. Вам нужно ответить, сколько всего локальных сетей в компании возникало после выведения каждого провода из строя.

Формат входных данных

В первой строке вводится целое число n ($2 \leqslant n \leqslant 3 \cdot 10^5$) - количество серверов в компании.

Во второй строке вводится целое число m ($1 \leqslant m \leqslant 3 \cdot 10^5$) - количество проводов.

В следующих m строках вводятся пары различных чисел a, b ($1 \leqslant a, b \leqslant n$) номера серверов, которые соединяет i-ый провод.

В следующей строке вводится число q ($1 \leqslant q \leqslant m$) количество оборванных проводов.

В следующей строке вводится q различных чисел – номера оборванных кабелей. Все номера различны и идут в хронологическом порядке.

Формат выходных данных

Выведите q чисел, количество различных локальных сетей после выведения из строя следующего провода.

Комментарии

Первый пример: после удаления первого провода все компьютеры все еще находятся в одной сети. После удаления второго провода, сеть разбивается на две части: компьютеры 1,3 и компьютер 2.

Примеры

Пример №1

Стандартный ввод
3 3 1 2 2 3 1 3 2 1 2

Стандартный вывод
1 2

Пример №2

Стандартный ввод
4 3 1 2 1 4 4 2 1 3

Стандартный вывод
2

Решение

Давайте решать обратную задачу: провода не обрывают, а чинят(идем по запросам в обратном порядке). Так как запросов порядка 10^5, надо уметь быстро объединять две локальные сети. Для этого можно использовать структуру данных система непересекающихся множеств, которая позволяет объединить два множество за время $O(\log n)$. Пусть изначально ни один компьютер не соединен с другим, то есть всего n локальных сетей. Процесс соединения двух компьютеров: если два компьютера лежат в разных локальных сетях(разные множества), то количество сетей уменьшится и мы объединим два множества, иначе оно останется таким же.

Возьмем все целые провода(которые не подвергались хакерской атаке) и объединим ими наши компьютеры. Количество локальных сетей после объединения будет равно ответу после последней хакерской атаки. Далее идем в обратном порядке по атакам и объединяем сети.

Пример программы

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;
typedef  long long ll;
typedef unsigned long long ull;
int inf_int=1e8;
ll inf_ll=1e18;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
const double pi=3.1415926535898;

const int MAXN=3e5+10;

int parent[MAXN];
int rank[MAXN];
int find_parent(int v) {
    if(v==parent[v]) {
        return v;
    }
    return parent[v]=find_parent(parent[v]);
}

bool union_set(int a,int b) {
```

```
    a=find_parent(a);
    b=find_parent(b);
    if(a!=b) {
        if(rank[a]<rank[b])
            swap(a,b);
        else if(rank[a]==rank[b])
            ++rank[a];
        parent[b]=a;
        return true;
    }
    return false;
}
int x[MAXN],y[MAXN];
char used[MAXN];
int a[MAXN];
int ans[MAXN];
void solve()
{
    int n,m,q;
    scanf("%d",&n);
    scanf("%d",&m);
    for(int i=1;i<=m;++i) {
        scanf("%d %d",&x[i],&y[i]);
    }
    scanf("%d",&q);
    for(int i=1;i<=q;++i) {
        scanf("%d",&a[i]);
        used[a[i]]=1;
    }
    for(int i=1;i<=n;++i)
        parent[i]=i;

    int cur=n;
    for(int i=1;i<=m;++i) {
        if(!used[i]) {
            cur-=union_set(x[i],y[i]);
        }
    }
    for(int i=q;i>=1;--i) {
        ans[i]=cur;
        cur-=union_set(x[a[i]],y[a[i]]);
    }
    for(int i=1;i<=q;++i) {
        printf("%d ",ans[i]);
    }


}

int main()
{
        int t=1;
        while(t--)
           solve();
        return 0;
}
```

Критерии отбора победителей и призеров первого этапа

Количество баллов, набранных при решении задач одной попытки, суммируется. Если участник решал задачи, как в первой, так и во второй попытке, то выбирается попытка с большей суммой баллов. Призерам первого отборочного этапа необходимо было набрать 16 баллов (для 9 класса) и 16 баллов (для 10-11 класса). Победители первого отборочного этапа должны были набрать 29 баллов.

2. ВТОРОЙ ОТБОРОЧНЫЙ ЭТАП

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго этапа составляет 37 дней. Задачи условно разделены на задачи по математике и информатике, но носят междисциплинарный характер и помогают отработать те навыки, которые потребуются для решения командной задачи заключительного этапа. Для каждой из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, задачи по информатике общие — для всех участников. Участники не были ограничены в выборе языка программирования для решения задач.

Объем и сложность задач этого этапа подобраны таким образом, чтобы решение всех задач одним человеком было маловероятно. Это призвано обеспечить включение командной работы и распределения обязанностей. Решение каждой задачи дает определенное количество баллов. Баллы зачисляются в полном объеме за правильное решение задачи. В данном этапе можно получить суммарно от 0 до 99 баллов.

Все условия задач по математике доступны участникам с первого дня второго отборочного этапа. Задачи по программированию выкладывались двумя партиями: в начале второго этапа и через три недели после начала. Команды могут выполнять задачи в любом порядке. Задачи допускают неограниченное число попыток сдать решение.

Задачи второго этапа

6.1. Задачи по математике (9 класс)

Задача 6.1.1. Простой криптоанализ (3 балла)

Используя частотный криптоанализ, расшифруйте следующее сообщение

$$MUMUHUBKSAORUSQKIUEVJUDJXUVHUGK$$

$$UDSOCUJXETDUUTIBEDWUHSYFXUHJUNJ$$

в преположении, что шифрование происходило при помощи «шифра Цезаря» (каждая буква латинского алфавита сдвигается на некоторое число d по модулю 26). Например, если $d = 16$, то сообщение

$$PAYMENOW \longrightarrow 15\ 0\ 24\ 12\ 4\ 13\ 14\ 22 \longrightarrow$$

$$\longrightarrow 5\ 16\ 14\ 2\ 20\ 3\ 4\ 12 \longrightarrow FQOCUDEM$$

Рекомендуем использовать вычислительную технику.

Решение

Находим самый часто встречающийся символ в сообщении: латинская буква U встречается 15 раз. Вероятнее всего ей должна изначально соответствовать латинская буква E. Следовательно, сдвиг d равен 10. После сдвига всех символов на d по модулю 26 получаем осмысленное сообщение.

Ответ: WEWERELUCKYBECAUSEOFTENTHEFREQUENCYMETHOD NEEDSLONGERCIPHERTEXT

Задача 6.1.2. Теоремы Ферма и Эйлера (5 баллов)

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Если найдется хотя бы одно натуральное число a, для которого $a^{n-1} \not\equiv 1 \pmod{n}$, то n — составное число.

б) Натуральное число n свободно от квадратов (если $n \vdots p$, то $n \not\vdots p^2$). Тогда уравнение $n = 7 \cdot \varphi(n)$ имеет ровно одно натуральное решение, где φ — функция Эйлера.

в) Существует лишь конечное число натуральных n, что $5^n - 1$ делится на 2^{100}.

г) $\varphi(n) \vdots \varphi(m)$ тогда и только тогда, когда $n \vdots m$.

д) Если для любого натурального числа a выполняется сравнение $a^n \equiv a \pmod{n}$, то n — простое число.

е) Известно, что натуральное число $n = p \cdot q$, где p и q различные простые числа. Тогда $p, q = \frac{\gamma \pm \sqrt{\gamma^2 - 4n}}{2}$, где $\gamma = n - \varphi(n) + 1$.

ж) Пусть p — нечетное простое число и $p - 1 = m \cdot 2^h$, где m — нечетное число, h — натуральное. Тогда при a взаимно простом с p выполняется

$$a^m \equiv 1 \pmod{p} \text{ или } \exists t \ (0 \leqslant t < h): \ a^{m2^t} \equiv -1 \pmod{p}.$$

з) Для любых натуральных m и n верно $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.

Ответ: 10000110

Задача 6.1.3. Тест Ферма по основанию 2 (6 баллов)

Пусть n – нечетное число. Если число 2^{n-1} при делении на n дает отстаток отличный от 1, то тест говорит, что n – составное число. К сожалению, в обратную сторону этот тест не работает (если остаток 1, то n может быть составным). Выясните сколько ложных результатов дает тест в промежутке $1 < n \leqslant 10^8$. Рекомендуем использовать вычислительную технику.

Решение

Ответ получается из непосредственного перебора всех чисел в промежутке $1 < n \leqslant 10^8$ (на компьютере). На каждом шаге бинарного возведения в степень храним только остаток по модулю n.

Ответ: 2057

Задача 6.1.4. Теория вероятности (5 баллов)

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Пусть $A = \{$при броске двух игральных кубиков сумма значений равно 6$\}$. Тогда $P(A) = 1/6$.

б) Из колоды игральных карт (52 карты) выбирается одна случайная карта. Пусть

$$A = \{\text{карта бубен}\} \text{ и } B = \{\text{карта: валет, дама, король или туз}\}.$$

Тогда $P(A \cap B) = 3/52$.

в) Среди 100 учеников 30 занимаются олимпиадной математикой, 20 — олимпиадным программированием, а 10 — и тем и другим. Тогда вероятность того, что случайно выбранный ученик занимается олимпиадной математикой или олимпиадным программированием равна 2/5.

г) Дан конечный набор попарно непересекающихся событий $\{A_i\}_{i=1}^n$ и некоторое событие B. Если $P(A_i) > 0$ для любого i, то верна формула

$$P(B) = \sum_{i=1}^{n} P(A_i)P(B|A_i).$$

д) Существует набор событий $A_1, A_2, \ldots, A_n$ такой, что верно

$$P(A_1 \cup A_2 \cup \ldots \cup A_n) = P(A_1) \cdot P(A_2|A_1) \cdot \ldots \cdot P(A_n|A_1 \cap A_2 \cap \ldots \cap A_{n-1})$$

е) В семье три ребенка. Рассмотрим события

$$A = \{\text{есть мальчик и девочка}\} \text{ и } B = \{\text{есть хотя бы один мальчик}\}.$$

Тогда события A и B зависимы.

ж) События A и B независимы тогда и только тогда, когда независимы события $\overline{A}$ и $\overline{B}$.

з) Для того, чтобы события A, B и C были попарно независимыми необходимо и достаточно

$$P(A \cap B \cap C) = P(A) \cdot P(B) \cdot P(C)$$

Ответ: 01101010

Задача 6.1.5. Теория вероятности (5 баллов)

Известно, что 80% всех мобильных телефонов производится в Китае, 15% — в США и 5% — в Японии. Оказалось, что с вероятностью 0.04 телефон, произведенный в Китае, с дефектом; с вероятностью 0.06 — в США и с вероятностью 0.09 — в Японии. Посчитайте вероятность того, что случайно выбранный телефон, который оказался с дефектом, произведен в США. Ответ округлите до тысячных.

Решение

Вероятность дефекта при покупке телефона равна $0.8 \cdot 0.04 + 0.15 \cdot 0.06 + 0.05 \cdot 0.09 = 0.0455$. Вероятность того, что случайно выбранный телефон, оказавшийся с дефектом, произведен в США равна доле брака мобильников из США среди всего брака

$$\frac{0.15 \cdot 0.06}{0.8 \cdot 0.04 + 0.15 \cdot 0.06 + 0.05 \cdot 0.09} \approx 0.198.$$

Ответ: 0.198

Задача 6.1.6. Modular Exponentiation (6 баллов)

Сообщение

$$29273,\ 320901,\ 52885,\ 107960,\ 12432,\ 169798$$

было зашифровано при помощи криптографического алгоритма *modular exponentiation* с публичным ключем $(350377, 8461)$ (первое число модуль, а второе — основание). Расшифруйте сообщение (расставьте правильно пробелы). Рекомендуем использовать вычислительную технику.

Решение

Число 350377 изначально является простым, поэтому $\varphi(350377) = 350376$. Следовательно,

$$8461^{-1} \equiv 117565 \pmod{350376}.$$

Возведем каждое число из зашифрованного сообщения в степень 117565 по модулю 350377 и получим следующую последовательность чисел

$$190010,\ 041204,\ 191424,\ 142017,\ 110400,\ 030417.$$

Разбиваем каждое сообщение на блоки по 2 цифры и получаем порядковые номера латинских букв. В итоге после декодирования получаем

$$TAK,\ EME,\ TOY,\ OUR,\ LEA,\ DER.$$

Наконец, правильно расставляем пробелы.

Ответ: TAKE ME TO YOUR LEADER

6.2. Задачи по математике (10-11 класс)

Задача 6.2.1. Безопасный пароль (3 балла)

В научно-исследовательском институте чародейства и волшебства (НИИЧАВО) используется устройство для проверки безопасности пароля. Паролем может быть любая непустая последовательность символов из множества $\{a, b, c\}$. Будем обозначать такие наборы большими латинскими буквами. Устройство преобразует введенный в него пароль X в другую последовательность символов $Y = f(X)$. Само устройство f держится в секрете, но зато известно, что оно обладает тремя свойствами

(1) $f(aX) = X$

(2) $f(bX) = f(X)af(X)$

(3) последовательность символов $f(cX)$ получается из последовательности $f(X)$ выписыванием в обратном порядке.

Пароль признается безопасным, если $f(X) = X$. Например, пароль bab – безопасный ($f(bab) = f(ab)af(ab) = bab$). Найдите хотя бы один безопасный пароль, состоящий из 11 символов.

Решение

Все ответы получаются из простого рекурсивного алгоритма (на компьютере). Также подобрать хотя бы один ответ можно руками.

Ответ: Любой из: bccccabcccc, cbcccacbccc, ccbccaccbcc, cccbcaccccbc, ccccbaccccb

Задача 6.2.2. Теоремы Ферма и Эйлера (5 баллов)

Натуральное число n называется псевдопростым по основанию a, если a и n взаимно просты и $a^{n-1} \equiv 1 \pmod{n}$.

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Если найдется хотя бы одно натуральное число a, для которого $a^{n-1} \not\equiv 1 \pmod{n}$, то n — составное число.

б) Уравнение $n = 7 \cdot \varphi(n)$ имеет ровно одно натуральное решение, где φ — функция Эйлера.

в) Существует лишь конечное число натуральных n, что $13^n - 1$ делится на $2^{100} \cdot 3^{200} \cdot 5^{300}$.

г) Пусть $a > 1$ и n — взаимно простые числа. Натуральное число m удовлетворяет сравнению $a^m \equiv 1 \pmod{n}$ тогда и только тогда, когда $m \vdots \varphi(n)$.

д) Если для любого натурального числа a выполняется сравнение $a^n \equiv a \pmod{n}$, то n — простое число.

е) Известно, что натуральное число $n = p \cdot q$, где p и q различные простые числа.

Тогда $p, q = \frac{\gamma \pm \sqrt{\gamma^2 - 4n}}{2}$, где $\gamma = n - \varphi(n) + 1$.

ж) Пусть p — простое число и $p - 1 = m \cdot 2^h$, где m — нечетное число, h — натуральное. Тогда при a взаимно простом с p выполняется

$$a^m \equiv 1 \pmod p \text{ или } \exists t\ (0 \leqslant t < h):\ a^{m2^t} \equiv -1 \pmod p.$$

з) Если n – псевдопростое число одновременно по основаниям a и ab, то оно псевдопростое и по основанию b.

Ответ: 10000111

Задача 6.2.3. Тест Ферма по основаниям 2, 3 и 5 (5 баллов)

Натуральное число n такое, что НОД$(n,\ 30) = 1$. Если хотя бы одно число из 2^{n-1}, 3^{n-1} и 5^{n-1} при делении на n дает отстаток отличный от 1, то тест говорит, что n – составное число. К сожалению, в обратную сторону этот тест не работает (если все три числа дают остаток 1, то n может быть составным). Выясните сколько ложных результатов дает тест в промежутке $1 < n \leqslant 10^8$. Рекомендуем использовать вычислительную технику.

Решение

Ответ получается из непосредственного перебора всех чисел в промежутке $1 < n \leqslant 10^8$ (на компьютере). На каждом шаге бинарного возведения в степень храним только остаток по модулю n.

Ответ: 257

Задача 6.2.4. Теория информации: энтропия Шеннона (5 баллов)

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Энтропия Шеннона принимает минимальное значение на равномерном распределении (для любого i верно $p_i = \frac{1}{n}$) среди всех возможных распределений на таком же числе исходов.

б) $H(\alpha \mid \beta) = H((\alpha, \beta)) - H(\beta)$.

в) Для любого вероятностного распределения $H(\alpha) \geqslant 0$.

г) $H((\alpha, \beta)) \leqslant H(\alpha)$ для любых двух вероятностных распределений α и β.

д) $H((\alpha, \beta)) = H(\alpha) \cdot H(\beta)$ тогда и только тогда, когда α и β — независимые вероятностные распределения.

е) $H((\alpha, \beta, \gamma)) \leqslant H(\alpha) + H(\beta) + H(\gamma)$ для любых трех вероятностных распределений α, β и γ.

ж) $I(\alpha : \beta) = 0$ тогда и только тогда, когда α и β — независимые вероятностные распределения.

з) Существует вероятностное распределение α такое, что $p_1 = p_2 = p_3 = \frac{1}{100}$ и $H(\alpha) = 0$.

Ответ: 01100110

Задача 6.2.5. Теория вероятности (6 баллов)

На множестве $\{0,1\}^3$ задано совместное распределение (α, β, γ). Причем оказалось, что распределение равномерно на множестве значений $\alpha + \beta + \gamma \equiv 0 \pmod 2$ (во всех остальных случаях вероятность равна 0). Вычислите величину

$$I(\alpha : \beta : \gamma) = H(\alpha) + H(\beta) + H(\gamma) - H((\alpha, \beta)) - H((\beta, \gamma)) - H((\gamma, \alpha)) + H((\alpha, \beta, \gamma)).$$

Решение

Возможны варианты $(\alpha, \beta, \gamma) = \{(0,0,0),\ (1,1,0),\ (1,0,1),\ (0,1,1)\}$. Следовательно, вероятность каждого исхода равна $1/4$.

Заметим, что $P(\alpha = 1) = P(\alpha = 0) = 1/2$, следовательно $H(\alpha) = \log_2 2$. Также $P(\alpha = i, \beta = j) = 1/4$, следовательно $H(\alpha, \beta) = \log_2 4$. И наконец, мы знаем, что вероятность каждого исхода $P(\alpha = i, \beta = j, \gamma = k) = 1/4$, где $i + j + k \equiv 0 \pmod 2$. Поэтому $H(\alpha, \beta, \gamma) = 4 \cdot 1/4 \cdot \log_2 4 = \log_2 4$.

В итоге получаем: $I(\alpha : \beta : \gamma) = 3 \cdot \log_2 2 - 3 \cdot \log_2 4 + \log_2 4 = -1$.

Ответ: -1

Задача 6.2.6. Алгоритм RSA (6 баллов)

Сообщение

$$2163315218,\ 5629365529,\ 9458153711,\ 1721672107,\ 10120672595$$

было зашифровано при помощи криптографического алгоритма RSA с публичным ключом $(11170948057, 11987)$ (первое число модуль, а второе — основание). Расшифруйте сообщение (расставьте правильно пробелы и удалите лишние символы). Рекомендуем использовать вычислительную технику.

Решение

Для того, чтобы расшифровать сообщение, мы должны разложить на множители $11170948057 = 139273 \cdot 80209$. Следовательно,

$$\varphi(11170948057) = 139272 \cdot 80208 = 11170728576$$

и

$$11987^{-1} \equiv 7586627291 \pmod{11170728576}.$$

Возведем каждое число из зашифрованного сообщения в степень 7586627291 по модулю 7586627291 и получим следующую последовательность чисел

$$0408060719,\ 1402111402,\ 1000130300,\ 1111081822,\ 0411112525.$$

Разбиваем каждое сообщение на блоки по 2 цифры и получаем порядковые номера латинских букв. В итоге после декодирования получаем

$$EIGHT,\ OCLOC,\ KANDA,\ LLISW,\ ELLZZ.$$

Наконец, правильно расставляем пробелы и откидываем лишние символы ZZ.

Ответ: EIGHT OCLOCK AND ALL IS WELL

6.3. Задачи по информатике

Задача 6.3.1. Little-Endian, Big-Endian (1 балл)

В современном мире любые данные в памяти компьютера и при их передаче по каналам связи представлены в виде последовательности байтов. В разных системах и протоколах байты принято упорядочивать по-разному: от старшего к младшему (big-endian) или от младшего к старшему (little-endian).

Например:

$$\begin{aligned} 2017_{10} &= 11111100001_2 \\ Big-endian: \quad 2017_{10} &= 00000111 \quad 11100001_2 \\ Little-endian: \quad 2017_{10} &= 11100001 \quad 00000111_2 \end{aligned} \tag{6.1}$$

В мире блокчейн разницу между кодированием чисел можно сразу увидеть: Bitcoin использует последовательность little-endian, Ethereum использует последовательность big-endian.

Напишите программу, которая преобразует число к формату с little-endian с учетом того для гипотетической вычислительной системы, где используются байти состоящие только из 7 бит.

Формат входных данных

На вход подается целое число N ($1 \leqslant N \leqslant 2^{28} - 1$).

Формат выходных данных

Необходимо вывести последовательность из 28 символов 0 или 1 (биты) – представление входного числа в формате little-endian.

Примеры

Пример №1

Стандартный ввод
2

Стандартный вывод
0000010000000000000000000000

Пример №2

Стандартный ввод
268435305

Стандартный вывод
1101001111111011111111111111

Решение

По условию задачи мы знаем, что длина двоичного представления числа, подающегося на вход не превышает 28 символов. Переведем данное число в двоичную систему счисления и добавим недостающие нули в начало получившегося числа.

Осталось разделить полученную строку на 4 подстроки по 7 элементов и вывести их в обратном порядке.

Пример программы

Ниже представлено решение на языке Python3

```
1  n = int(input())
2  n = bin(n)[2:]
3  addition = "0" * (28-len(n))
4  n = addition + n
5  for i in range(21, -1, -7):
6      print(n[i:i+7], end = "")
```

Задача 6.3.2. Кодирование Base58 (3 балла)

Base58 кодирование используется в Bitcoin для получение человекочитаемого адреса, который используется для получения крпитовалюты. Для этого число, определяющее адрес, переводится в строку, для кодирования которой используется алфавит состоящий из 58 символов, включая строчные и заглавные буквы латинского алфавита и цифры, исключая неоднозначные в некоторых шрифтах символы, такие как 0 (ноль), O (заглавная o), I (заглавная i) и l (строчная L).

Напишите программу, которая бы определяла символ алфавита, который используется в конкретном адресе Bitcoin чаще других символов.

Формат входных данных

Входной файл содержит 10 строк, где каждая строка – последовательность из нулей и единиц - двоичное представление адреса Bitcoin. Длина каждой строки - 176 символов.

Формат выходных данных

Укажите символ, который используется чаще всего в Base58 представлениях указанных адресов. Если самый частый символ - единица, то укажите следующий по частоте символ. Гарантируется, что других символов с такой же частотой встречаемости нет.

Примеры

Пример №1

Стандартный ввод
000000000000000010001011011001010000110101110100001111010110101100001101 1
000000111110110000010011010010010110101010001101101100000111010011100100 0
011111111100000100000110001011
000000000000000010000010100000001001101010001100101101101100110010000101 1
000111111010010111001101101101111101100111111011111001101011100010110011 0
111011111010111000011110111011
000000000000000010001101011000111101000110011101101011111111000001000111 1
100011010101110011011000101000011111010000001101000000001100001110101101 1
101000010101111010001011110111
000000000000000010101011000101101110111001000010100101000110001011011011 0
100100111011110110001110001110101001110101101011111101010111110011100110
111011101100100110110100011001
000000000000000001001111000110110100110111101010000100010100101011100011 0
000111100001110100011100000001101001110100000001100100110010101000011100 0
100110011100101011100111101010
000000000000000011001010111100010011010000000010101000001000110110011111 0 11
111010011110101010111100111110110111100110111000111011010000101100010000 1
111011001010001011111111100111
000000000000000000010010101101001000100000000100001010011101010010000000 0 00
100100011110101000101001110000010110110000000101111000010011001010100101 1 11
101110010010001000110100100110
000000000000000001010011110011000010101000011000011111100110011110010000 1
101110110101011011001011000000011010100000000011001111100100111010000011 01
001001010000010011011000111000
000000000000000011110110000001011110110000001101000000011001110101111111 0
100001110011011011110100111001111110011001110010110011011110011001011111
101010011011011101100010010001
000000000000000001111111100110010000001100001011010100000101001110111101
011010010000111101010010000100111000000011110110100100011110000001101101 1
010110110010011110011001000110

Стандартный вывод
d

Комментарии

Используйте библиотеку Python `bitcoin` для генерации base58 адреса Bitcoin из его численного представления.

Решение

Для решения данной задачи воспользуемся функцией *hex_to_b58check* из библиотеки Python `bitcoin`.

Согласно документации Bitcoin, адрес генерируется с помощью base58 кодирования, применяемого к хэшу публичного ключа. Необходимый хэш получается путем последовательного применения функций sha256 и ripemd160 к публичному ключу аккаунта и состоит ровно из 20 байт (`https://github.com/bitcoinbook/bitcoinbook/blob/second_edition/ch04.asciidoc`).

Таким образом, для кодирования необходимо перевести полученное на вход число в шестнадцатеричную систему счисления и убедиться, что оно содержит ровно 40 символов. В случае необходимости мы просто добавляем незначащие нули в начало хэша.

Проходясь по каждому адресу, находим наиболее часто встречающийся символ и в конце выводим ответ.

Пример программы

Ниже представлено решение на языке Python3

```
import bitcoin
n = 10
dict = {}
for i in range(n):
    f = input()
    f = hex(int(f, 2))[2:]
    addition = 40 - len(f)
    f = '0'*addition + f
    f = bitcoin.hex_to_b58check(f)
    for j in f:
        if j in dict:
            dict[j] += 1
        else:
            dict[j] = 1

a = sorted(list(dict.items()), key = lambda x: x[1], reverse=True)
if a[0][0] != '1':
    print(a[0][0])
else:
    print(a[1][0])
```

Задача 6.3.3. Сериализация данных в Ethereum (3 балла)

Человек привык работать с символьным отображением информации, цифровые устройства хранят, обрабатывают и пересылают любую информацию в виде чисел, а если быть более точным, в виде двоичных битов. Разные структуры данных по разному выглядят для человека: числа - последовательности цифр, строки - последовательности символов, списки - последовательности символов со специальным форматированием. Цифровому устройству нужно использовать специальные алгоритмы, для того, чтобы сохранить данные структуры в памяти или передать по коммуникационному каналу. Процесс перевода структур данных в формат, используемый для хранения и пересылки, называется *сериализацией*, обратный процесс называется *десериализацией*.

Программное обеспечение Ethereum для сериализации и десериализации данных использует метод RLP (Recursive Length Prefix, `https://github.com/ethereum/wiki/wiki/RLP`.

Например,

Изначальные данные	Сериализированные данные в шестнадцатиричном формате
42	2a
31415	827ab7
'ethereum'	88657468657265756d
['a task', 4, 'you']	cc8661207461736b0483796f75

Напишите программу, которая меняет порядок элементов на обратный в сериализованной с использованием RLP кодирования структуре данных по следующему правилу:

- Если данные - число, то в нем меняется порядок с big-endian на little-endian.
- Если данные - строка, то ее элементы представляются в обратном порядке.
- Если данные - список, то его элементы представляются в обратном порядке.

Формат входных данных

На вход подается одна строка - сериализованные данные в шестнадцатиричном виде. Длина строки - не более 65535 символов.

Формат выходных данных

Необходимо вывести сериализованные с использованием RLP кодирования данные в шестнадцатиричном виде.

Примеры

Пример №1

Стандартный ввод
`81df`

Стандартный вывод
`81df`

Пример №2

Стандартный ввод
`83017d45`

Стандартный вывод
`83457d01`

Пример №3

Стандартный ввод
`87426974436f696e`

Стандартный вывод
`876e696f43746942`

Пример №4

Стандартный ввод
`cb8277658673616c75746575`

Стандартный вывод
`cb758673616c757465827765`

Решение

Функция RLP-кодирования может принимать в качестве аргумента элементы, которые мы назовем 'item', где item определен следующим образом:

1. Строка (т.е. массив байтов) является item-ом
2. Список item-ов является item-ом

Теперь рассмотрим каким образом происходит само кодирование:

- Для одного байта, значение которого находится на отрезке [0, 127] (или [0x00, 0x7f]), данный байт возвращается без каких-либо изменений.
- Для строки, длиной от 0 до 55 байт, RLP-кодирование включает байт, который определяет длину строки и вычисляется как длина строки (количество байтов в массиве) +128 (0x80). Таким образом первый байт лежит на отрезке [128, 183] ([0x80, 0xb7]). Следом за ним идет сама строка.
- Для строки, длиной свыше 55 байт, в начало добавляется первый дополнительный байт, который находится как минимально необходимое для кодирования длины строки количество байтов +183 (0xb7) и лежит на отрезке [184, 191] ([0xb8, 0xbf]). Следом за ним идет некоторое количество байт, в которых представлена длина строки с порядком байтов - "BigEndian". Затем же идет сама строка.

  Попробуем разобраться, что означает минимально необходимое для кодирования длины строки количество байтов. Предположим что наша строка состоит из 1024 байт. Тогда её длина будет равна 1024. Легко заметить, что число 1024 соответствует числу '400' в шестнадцатеричной системе счисления и для её кодирования достаточно 2 байта. Таким образом первый байт для данной строки будет равен 183+2 или 185 (0xb9), а следом за ним будут находиться 2 байта, равные 'x04' и 'x00' соответсвенно.

Для списков все выглядит очень похоже:

- Если итоговая длина списка не превышает 55 байтов, то RLP-кодирование состоит из байта, равного длине списка +192 (0xc0), следом за которым идет сам список. Соответственно, первый байт будет находиться в пределах отрезка [192, 247] ([0xc0, 0xf7]).

  Однако тут необходимо уточнить, каким образом находится итоговая длина списка. Покажем это на примере. Предположим, что мы хотим закодировать список, состоящий из двух элементов - ['cat', 'dog']. При этом и 'cat' и 'dog' должны быть уже закодированы в виде массива байтов. Сначала, мы должны преобразовать каждый элемент списка в его RLP-представлении. Тогда наш массив можно будет представить в виде [0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g']. Таким образом, его длина будет равна 8 байтам и после кодирования мы должны получить: [0xc8, 0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g'].

- Если же итоговая длина списка больше 55 байт, то RLP-кодирование состоит из байта, равного минимальной длине в байтах итоговой длины списка + 247 и лежащего на отрезке [248, 255] ([0xf8, 0xff]), следом за которым идет минимальное количество байт, которые представляют итоговую длину списка, следом за которыми идет сам список.

Теперь обратимся к условию. Заметим, что для нас не имеет значения число или строка были закодированы, так как и в том и в другом случае, нам необходимо переставить байты местами. Таким образом, по первому байту возможно сразу определить, чем являются полученные данные - строкой или списком, и каким образом необходимо переставлять их элементы. Остается лишь аккуратно разобрать все возможные случаи.

Пример программы

Ниже представлено решение на языке Python3

```python
def RLPDecode(s: str):
    result = ''
    temp = s[:2]
    if 0 <= int(temp, 16) <= 191:
        result += RLPDecodeString(s)
    else:
        result += RLPDecodeArray(s)
    return result

def RLPDecodeString(s: str):
    result = ''
    temp = s[:2]
    result += temp
    if 128 <= int(temp, 16) <= 183:
        for i in range(len(s)-2, 1, -2):
            result += s[i] + s[i+1]
    elif 184 <= int(temp, 16) <= 191:
        tsize = int(temp, 16) - 183
        for i in range(1, int(temp, 16) - 183 + 1):
            temp = s[2*i:2*i+2]
            result += temp
        for i in range(len(s) - 2, 1+tsize*2, -2):
            result += s[i] + s[i+1]
    return result

def RLPDecodeArray(s: str):
    result = ''
    elements = []
    temp = s[:2]
    result += temp
    if 192 <= int(temp, 16) <= 247:
        i = 2
        while i < len(s):
            temp = s[i:i+2]
            if 0 <= int(temp, 16) <= 191:
                tempPair = RLPDepodeStringFromArray(s[i:])
                elements.append(tempPair[0])
                i += tempPair[1]
            elif int(temp, 16) <= 255:
                tempPair = RLPDecodeArrayFromArray(s[i:])
```

```
41                 elements.append(tempPair[0])
42                 i += tempPair[1]
43     elif int(temp, 16) <= 255:
44         tsize = int(temp, 16) - 247
45         for i in range(1, int(temp, 16) - 247 + 1):
46             temp = s[2*i:2*i+2]
47             result += temp
48         i = tsize*2 + 2
49         while i < len(s):
50             temp = s[i:i+2]
51             if 0 <= int(temp, 16) <= 191:
52                 tempPair = RLPDepodeStringFromArray(s[i:])
53                 elements.append(tempPair[0])
54                 i += tempPair[1]
55             elif int(temp, 16) <= 255:
56                 tempPair = RLPDecodeArrayFromArray(s[i:])
57                 elements.append(tempPair[0])
58                 i += tempPair[1]
59     for i in range(len(elements)-1, -1, -1):
60         result += elements[i]
61     return result
62
63 # Находит закодированную строку
64 def RLPDepodeStringFromArray(s: str):
65     result = ''
66     tempSize = 0
67     temp = s[:2]
68     if 0 <= int(temp, 16) <= 127:
69         result += temp
70         tempSize = 2
71     elif int(temp, 16) <= 183:
72         size = int(temp, 16) - 128
73         result += s[:2+size*2]
74         tempSize = 2+size*2
75     elif int(temp, 16) <= 191:
76         tsize = int(temp, 16) - 183
77         temp = s[2:2*tsize+2]
78         size = int(temp, 16)
79         result += s[:size*2+tsize*2+2]
80         tempSize = size*2+tsize*2+2
81     return (result, tempSize)
82
83 # Находит закодированный массив
84 def RLPDecodeArrayFromArray(s: str):
85     result = ''
86     tempSize = 0
87     temp = s[:2]
88     if 192 <= int(temp, 16) <= 247:
89         size = int(temp, 16) - 192
90         result += s[:2+size*2]
91         tempSize = 2+size*2
92     elif int(temp, 16) <= 255:
93         tsize = int(temp, 16) - 247
94         temp = s[2:2*tsize+2]
95         size = int(temp, 16)
96         result += s[:size*2+tsize*2+2]
97         tempSize = size*2+tsize*2+2
98     return (result, tempSize)
99
100 f = open('input.txt', 'r')
```

```
out = open('output.txt', 'w')
for i in f.readlines():
    i = i.strip()
    out.write(RLPDecode(i))
    out.write('\n')
```

Задача 6.3.4. Вычисление контрольной суммы (2 балла)

Контрольная сумма используется для проверки целостности данных при их передаче или хранении. Также контрольные суммы могут использоваться для быстрого сравнения двух наборов данных на неэквивалентность: при правильном выборе алгоритма, вычисляющего контрольную сумму, с большой вероятностью различные наборы данных будут иметь неравные контрольные суммы.

Один из самых простых алгоритмов вычисления контрольной суммы - сложение по модулю 2 (XOR, $\oplus$) для всех блоков, на которые разбиваются входные данные.

$$CRC = w_1 \oplus w_2 \oplus w_3 \oplus \cdots \oplus w_n, \tag{6.2}$$

где w_1, w_2, $\ldots w_n$ - блоки одинакового размера, на которые разбиты входные данные.

Пример для контрольной суммы, размером в 16 бит: если входные данные - строка символов "0xyz", то они разбиваются на блоки "0x" и "yz". Тогда контрольная сумма будет равна 4902 в шестнадцатиричной системе счисления.

Метод продемонстрированный выше, можно также отнести к функциям хэширования (`https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5`).

Напишите программу для нахождения такой последовательности бит в тридцатидвухбитном слове, чтобы контрольная сумма для входных данных сконкатенированных с полученным словом совпадала с заданной.

Для вычисления контрольной суммы используйте алгоритм предложенный выше.

Формат входных данных

Входной файл содержит только 2 строки.

Первая строка содержит восемь символов - запись требуемой контрольной суммы в шестнадцатиричной системе счислениях.

Вторая строка содержит последовательность символов, включающую цифры, латинские строчные и заглавные буквы, для которых нужно вычислить контрольную сумму. Длина строки не больше 65535 символов.

Формат выходных данных

8 символов - контрольная сумма в шестнадцатиричной системе счисления.

Примеры

Пример №1

Стандартный ввод
45a060c5 1234diGi
Стандартный вывод
10fb1498

Пример №2

Стандартный ввод
43852d88 sYmb0ls
Стандартный вывод
9fb03375

Комментарии

Пояснение к примеру №1: последовательность `'1234diGi'` можно представить в виде последовательности байт `b'\x31\x32\x33\x34\x64\x69\x47\x69'`, тогда конкатенируя ее с последовательностью `b'\x10\xfb\x14\x98'`, получится `b'\x31\x32\x33\x34\x64\x69\x47\x69\x10\xfb\x14\x98'`, по которой уже необходимо считать контрольную сумму.

Решение

Для начала заметим, что операция сложения по модулю 2 обладает таким замечательным свойством как обратимость, то есть $(a \oplus b) \oplus b = a$. Таким образом, для любого набора из трех битов, один из которых является результатом сложением двух других, не имеет значения какие из двух битов складывать. В результате мы всегда будем получать значение третьего бита.

Далее нам необходимо понять на блоки каких размеров надо делить данную на вход строку. В этом нам помогает итоговая контрольная сумма, которую надо получить. Её запись в шестнадцатеричной системе счисления состоит из 8 символов, следовательно один блок должен состоять из 4 байтов (или 4 символов).

Для начала разберемся со случаем, когда количество символов во входной строке кратно 4. Тогда разобьем данное слово на равные части и сконкатенируем их между собой. Затем воспользуемся указанным выше свойством обратимости и сконкатенируем полученный до этого результат с контрольной суммой. Полученное значение и будет ответом.

Но остается вопрос, что делать в случае, когда количество символов в строке, для которой необходимо вычислить контрольную сумму не кратно 4. Для того, чтобы ответить на него, обратимся ко второму примеру и составим таблицу, состоящую из двоичного представления полученных данных.

Все строки в данной таблице разделены и состоят из 8 ячеек, представляющих собой 1 байт данных. В последней ячейке каждой строки указано, что было закодировано в этой строке.

Исходя из этой таблицы, можно понять, как происходит сложение в случае, если строка содержит количество символов не кратное 4. В этом случае первые несколько байт в ответе, который надо вывести, необходимо конкатенировать с окончанием

0	1	1	1	0	0	1	1	0	1	0	1	1	0	0	1	0	1	1	0	1	1	0	1	0	1	0	0	0	1	0	"sYmb" - (1)
0	0	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	1	"0ls" + "9f" - (2)
0	1	0	0	0	0	1	1	0	0	1	1	0	1	0	1	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	xor((1), (2)) - (3)
1	0	0	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	0	1	0	1	"00" + "b03375" - (4)
0	1	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0	0	xor((3), (4))

Рис. 6.1: Бинарное представление примера

данной строки. А их место в конечной строке должны занять нули. То есть, если остаток от деления количества символов в строке на 4 равен, к примеру, двум, то к данной строке необходимо прибавить еще первые 2 байта из ответа, а затем полученный результат необходимо сложить с последними двумя байтами ответа. Так, во втором примере, конкатенация должна происходить в следующем порядке (последняя строка выражает контрольную сумму):

73	59	6d	62
30	6c	73	9f
0	b0	33	75
43	85	2d	88

Рис. 6.2: Шестнадцатеричное представление примера

Осталось понять, как найти этот самый ответ, из которого и будут добавляться недостающие байты.

Для удобства будем называть первой строку, которая получается конкатенацией всех подстрок из 4 символов, не считая последнюю подстроку, в которой количество символов меньше 4. Второй будем называть ту самую строку, в которой не достает символов, а третьей - последнюю строку, с которой происходит сложение. Ответом будем называть результат, который мы должны вывести.

Из примера становится ясно, что, для успешного решения задачи, все тесты при сложении должны давать в первых n битах (n - остаток от деления количества символов на 4) правильный ответ сразу же, так как независимо от ответа в третьей строке на этих местах будут нули. При этом, кажется, совершенно неважно, что будет в первых байтах, которые заменят недостающие во второй строке символы, так как мы всегда можем подобрать для них последовательность байт в третьей строке, с которой они будут складываться. Однако тут важно заметить, что будет в случае, если n равно 1. Тогда ответственность за правильный второй байт во второй строке лежит именно на нас. Поэтому будем добавлять во вторую строку такие байты, чтобы при конкатенации с первой строкой она сразу давала необходимый результат, а в третьей строке в этом случае можно было оставить нули.

Чтобы получить такие байты снова воспользуемся свойством операции XOR. Заполним недостающее количество байт во второй строке нулями и сложим полученную строку сначала с первой строкой, а затем и с результатом, который нам необходимо получить. Полученная последовательность байт - это ответ, где первые n байт расположены в конце, а оставшиеся - в начале.

Пример программы

Ниже представлено решение на языке Python3

```python
def string_xor(s, code: str):
    result = 0
    while len(s) >= 4:
        result ^= int(s[:4].encode().hex(), 16)
        s = s[4:]
    if len(s) == 0:
        ans = hex((result ^ int(code, 16)))[2:]
        ans = '0' * (8-len(ans)) + ans
    else:
        result ^= int(code, 16)
        tempStr = (s.encode() + b'\x00'*(4 - len(s)%4)).hex()
        ans = hex(result ^ int(tempStr, 16))[2:]
        ans = '0' * (8 - len(ans)) + ans
        ans = ans[2*(len(s)%4):] + \
              ans[8-2*(len(s)%4):2*(len(s)%4)] + ans[:8-2*(len(s)%4)]

    return ans

code = input()
s = input()
print(string_xor(s, code))
```

Задача 6.3.5. Хэширование SHA-3 (2 балла)

Существует достаточно много методов хэширования данных. Какие-то из методов отличаются скоростью работы, другие - обладают свойством криптостойкости.

Хэш-функция будет считаться криптостойкой, если она - наобратима, т.е. когда с помощью преобразований результата ее работы нельзя восстановить изначальную последовательность данных. Еще одно свойство криптостойких хэш-функций - устойчивость к коллизиям, когда вычислительно невозможно найти два разных набора данных для которых хэщ-функция будет выдавать одинаковый результат.

Те криптостойкие хэш-функции, которые используются в криптографии, называют криптографическими функциями.

В блокчейн Ethereum базовой криптографической хэш-функцией является алгоритм SHA-3(*keccak*) с размером выходных значений 256 бит.

Пример: результат работы хэш-функции SHA-3 для строки *'Ethereum is a distributed database'*, записанный в шестнадцатиричном виде будет `80529d6b03ffe0bcc4064ac573a512cb114cf4d8033913178e329b6dcda208a7`.

Как уже было сказано выше, единственный способ узнать, какие данные были поданы на вход криптографической хэш-функции, - это выполнить их полный перебор. Напишите программу, которая определяет точную дату рождения школьника, если известны результат хэш-функции SHA-3 и год, когда он родился.

Формат входных данных

Входной файл содержит 2 строки.

Первая строка целое число - год рождения школьника.

Вторая строка содержит результат хэш-функции SHA-3, записанный в шестнадцатиричной системе счисления. Значение, которое передается в хэш-функцию,

определяется как количество секунд (целое), прошедшее с 1 января 1970 Всемирного координированного времени (UTC) до 0 часов 0 минут соответствующего дня рождения.

Формат выходных данных

Строка в формате `DD.MM`, где

1. DD - двузначное число, обозначающее день месяца;
2. MM - двузначное число, обозначающее месяц.

Примеры

Пример №1

Стандартный ввод
2003 82717807fc9b67b1510f9dc37425bb2453a66ea41a9c297b0c9148816d449c6b

Стандартный вывод
21.10

Пример №2

Стандартный ввод
2004 a62bd35682c84f6ea77fbf057ab30c9a3525cb4e2ddd5f8d9211c8b3e1b95d8d

Стандартный вывод
29.02

Комментарии

Для решения задачи используйте функцию `keccak_256` из Python библиотеки `pysha3`.

Обратите внимание, что функции хэширования принимают на вход набор байт. Следовательно, перед тем как вычислить хэш-функцию от целого числа, данное число нужно представить в виде набора байт. В Ethereum целые числа конвертируются как 256битный код с порядком байт big-endian.

Решение

Для решения задачи, помимо указанной в примечании библиотеки `pysha3`, будем использовать встроенный модуль `datetime` и функцию *timestamp*, которая вычисляет количество секунд, прошедшее с 1 января 1970 Всемирного координированного времени, автоматически устанавливая временную зону +0 относительно UTC.

Однако уже для первого же теста ответ, полученный при помощи функции *timestamp* не совпадает с тем, который дан в условии. Проблема в летнем времени, которое сдвигает время на один час назад или вперед, в зависимости от даты. Для того, чтобы

обойти эту проблему, при переборе будем учитывать этот сдвиг, то есть проверять не только для 0 часов текущего дня, но и для 23 часов предыдущего, считая датой рождения в этом случае день, который наступит спустя час.

Кроме того, как было написано в задании, важно учитывать, что на вход функция *keccak_256* принимает набор из 32 байт.

Пример программы

Ниже представлено решение на языке Python3

```
from _pysha3 import keccak_256
import datetime

def increaseTime(curr_datetime: datetime):
    if curr_datetime.hour == 23:
        curr_datetime += datetime.timedelta(hours=1)
    else:
        curr_datetime += datetime.timedelta(hours=23)
    return curr_datetime

year = int(input())
code = input()
year, month, day = year-1, 12, 31
ans = ""
dfTime = datetime.datetime(year, month, day, 23, 0, 0)

while ans != code:
    dfTime = increaseTime(dfTime)
    seconds = dfTime.timestamp()
    tempTime = int.to_bytes(int(seconds), 32, byteorder='big')
    ans = keccak_256(tempTime).hexdigest()
if dfTime.hour == 23:
    dfTime = increaseTime(dfTime)

print('{num:02d}'.format(num=dfTime.day),'{num:02d}'.format(num=dfTime.month), sep='.')
```

Задача 6.3.6. Получение публичного ключа по приватному (3 балла)

В блокчейн Ethereum для идентификации пользователей и доказательства владения используется криптография с открытым ключом, основанная на математических операциях на эллиптической кривой.

С каждым пользователем сети блокчейн ассоциирован его приватный ключ - случайная последовательность длиной 256 бит. Очевидно, что эту последовательность также можно представить в виде некоторого большого числа K.

Полные узлы сети Ethereum, майнеры, а также обычные пользователи всегда могут проверить, действительно ли какой-то конкретный пользователь является источником транзакции, зная его публичный ключ.

Публичный ключ генерируется на основе приватного. С точки зрения эллиптической криптографии, публичный ключ – координата точки на эллиптической кривой, полученная в качестве умножения некоторой базовой точки G, на значение равное приватный ключ K.

$P = G \cdot K$

Поскольку результатом данной функции являются координаты, то для представления их в виде публичного ключа, нужно выравнять их до 32байтных слов и сконкатенировать друг с другом.

Напишите программу, которая из приватного ключа генерирует публичный, используя криптографию на эллиптических кривых. Имейте в виду, что перед генерацией публичного ключа, который будет использоваться в блокчейн Ethereum, соответствующий приватный ключ нужно привести к набору из 32 байт (даже если он, как число, требует меньше разрядов).

Формат входных данных

Входная строка содержит последовательность символов – приватный ключ, записанный в шестнадцатеричном формате.

Формат выходных данных

Одна строка – публичный ключ, соответствующий приватному, записанный в шестнадцатеричном формате.

Примеры

Пример №1

Стандартный ввод
`c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470`

Стандартный вывод
`a63a07e888061ec9e8b64a3dc2937805c76089af36459305920373cd98a9f4ce15` `c27dbbe60928161eb62ae19f94ea48f399ce85e6db698520f3bcd4a9257157`

Пример №2

Стандартный ввод
`10ca3eff73ebec87d2394fc58560afeab86dac7a21f5e402ea0a55e5c8a6758f`

Стандартный вывод
`765ee8b856d2ec6a1ea400c0dea57095fd096a45fe6735fe199738c4ec4909c6c2` `bda34586353edd45419d715925875ebb9f371c3eeb38ed3a3615b93e20dc1a`

Комментарии

В примере выходных данных публичный ключ представлен в виде двух разделенных строк из-за форматирования. В реальных входных данных публичный ключ должен быть выведен в одной строке.

Для решения задачи используйте модуль `secp256k1` библиотеки Python `py_ecc`.

Решение

Для решения задачи воспользуемся функцией `privtopub` из модуля `secp256k1`.

Напишем функцию, которая будет из приватного ключа получать публичный ключ, записанный в шестнадцатеричном формате. Для этого:

1. Сначала приведем приватный ключ к набору из 32 байт.
2. Затем при помощи библиотеки `py_ecc` получим необходимые координаты.
3. Выровняем полученные координаты до 32 байт и сконкатенируем их друг с другом.

Пример программы

Ниже представлено решение на языке Python3

```python
from py_ecc import secp256k1

def privToPub(priv):
    priv = int(priv, 16).to_bytes(32, 'big')
    res = secp256k1.privtopub(priv)
    a = res[0].to_bytes(32, 'big')
    b = res[1].to_bytes(32, 'big')
    pub = int.from_bytes(a + b, 'big')
    return '0' * (128-len(hex(pub)[2:])) + hex(pub)[2:]

inp = open('input.txt', 'r')
out = open('output.txt', 'w')

for i in inp.readlines():
    i = i.strip()
    out.write(privToPub(i) + '\n')

inp.close()
out.close()
```

Задача 6.3.7. Получение адреса Ethereum (4 балла)

Между адресом пользователя в блокчейн Ethereum и его публичным ключом устанавливается непосредственная связь. Это сделано для того, чтобы не тратить дополнительное место (64 байта) в каждой транзакции для хранения публичного ключа.

Адрес пользователя - последовательность из 20 байт, которые получены из публичного ключа следующим преобразованием:

$addr = right(keccak256(pubkey), 20)$

Чаще всего, адрес пользователя можно увидеть записанным в виде шестнадцатеричного числа. Чтобы защитить адрес от случайных ошибок при переписывании с какого-то бумажного носителя (например, кто-то может свой адрес указать на визитке), была предложена система для интеграции кода проверки (checksum) в шестнадцатеричную запись адреса без изменения самого адреса: *EIP-55 «Mixed-case checksum address encoding»*. Суть системы в том, что в зависимости от значений

определенных битов хэш-сумме адреса некоторые строчные латинские буквы в его шестнадцатеричном отображении заменяются на заглавные.

Например,

`5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed`

Напишите программу, которая из приватного ключа генерирует адрес пользователя Ethereum и выводит позиции в шестнадцатеричной записи адреса, в которых строчная буква должна быть заменена на заглавную согласно *EIP-55*.

Формат входных данных

Входная строка содержит последовательность символов – приватный ключ, записанный в шестнадцатеричном формате.

Формат выходных данных

Две строки, первая - адрес без кода проверки, вторая – последовательность чисел, разделенных пробелом. Самый левый знак в шестнадцатеричной записи имеет позицию 0.

Примеры

Пример №1

Стандартный ввод
`c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470`

Стандартный вывод
`9cce34f7ab185c7aba1b7c8140d620b4bda941d6` `6 9 15 16 17 21 30 32 33 34`

Пример №2

Стандартный ввод
`10ca3eff73ebec87d2394fc58560afeab86dac7a21f5e402ea0a55e5c8a6758f`

Стандартный вывод
`eaee08c91b13d8d7a82850b714c7c68826e9cfae` `0 1 9 16 26 34 36 37 38`

Решение

Для получения адреса из публичного ключа воспользуемся модулем `keccak_256` из библиотеки `sha3`. Получение публичного ключа из приватного уже было разобрано в соответствующей задаче.

Во-первых, необходимо заметить, что на вход *keccak_256* получает последовательность байт. Кроме того, после кодирования данная функция возвращает массив из 32 байтов. Таким образом, для получения адреса надо обрезать первые 12 байт,

полученных в результате кодирования публичного ключа, что соответствует 24 символам в шестнадцатеричной записи.

Теперь рассмотрим алгоритм кода проверки *EIP-55*. Основная идея заключается в том, что адрес в шестнадцатеричной записи сравнивается с двоичным представлением результата применения функции *keccak_256* к данному адресу. То есть, если в шестнадцатеричной записи адреса стоит буква, а в двоичной записи закодированного адреса этой букве соответствует *1*, то данная буква должна быть заменена на заглавную. Если же данной букве соответствует *0* бит, то она не изменяется. Точно так же остаются без изменений и цифры, независимо от того, какой бит им соответствует. Кроме того, один символ в шестнадцатеричной системе счисления соответствует 4 битам в двоичном представлении, однако нам нужен самый первый из них.

Пример программы

Ниже представлено решение на языке Python3

```python
from py_ecc import secp256k1
from sha3 import keccak_256

def privToPub(priv):
    priv = int(priv, 16).to_bytes(32, 'big')
    res = secp256k1.privtopub(priv)
    x = res[0].to_bytes(32, 'big')
    y = res[1].to_bytes(32, 'big')
    return x+y

def pubToAddr(pub):
    return keccak_256(pub).hexdigest()[24:]

def privToAddr(priv):
    return pubToAddr(privToPub(priv))

def checkAddr(addr):
    hashAddr = bin(int(keccak_256(addr.encode()).hexdigest(), 16))[2:]
    hashAddr = '0' * (256 - len(hashAddr)) + hashAddr
    res = ''
    for i in range(len(addr)):
        if addr[i].isalpha() and hashAddr[4*i] == '1':
            res += str(i) + ' '
    return res

inp = open('input.txt', 'r')
out = open('output.txt', 'w')

for i in inp.readlines():
    i = i.strip()
    addr = privToAddr(i)
    out.write(addr + '\n')
    res = checkAddr(addr)
    out.write(res + '\n')

inp.close()
out.close()
```

Задача 6.3.8. Красивый адрес (5 баллов)

В сообществе криптовалют сформировалось несколько стартапов, которые предлагают пользователям сгенерировать "красивые"адреса.

Например, сколько может стоить вот такой Ethereum адрес?

`faceb00cab185c7aba1b7c8140d620b4bda941d6`

Генерация таких адресов без раскрытия своего приватного ключа стала возможна благодаря криптографии на эллиптических кривых.

Если зависимость адреса от приватного ключа можно выразить в виде:

$P = G \cdot K$

$addr = right(keccak256(P), 20)$

то можно написать также

$P_{faceb00c} = G \cdot (K + K_{faceb00c}) = G \cdot K + G \cdot K_{faceb00c} = P + G \cdot K_{faceb00c}$

Следовательно, пользователь, который хочет красивый адрес, не обязательно должен раскрывать свой приватный ключ. Вместо этого он может послать стартапу только свой публичный ключ и получить от него специальное дополнение, позволяющее получать требуемый красивый адрес.

Напишите программу, которая для известного публичного ключа формирует дополнительный приватный ключ, позволяющий получить красивый адрес.

Формат входных данных

В качестве входных данных будут переданы две строки. Первая строка содержит последовательность символов – публичный ключ, записанный в шестнадцатеричном формате. Вторая строка содержит четыре шестнадцатеричных знака - эти знаки должны стоять в красивом адресе в первых четырех символах.

Формат выходных данных

Одна строка - дополнительный приватный ключ, записанный в шестнадцатеричном формате.

Примеры

Пример №1

Стандартный ввод

`580e9e8c13aa4da3040962f57273ed5f1759c4edb32c6705310362bbf52f2c36`
`80c811172f177b2c3ed1d53ac7c521b8b1813284fbe0279557401bfe20d256da`
`face`

Стандартный вывод
`0000000000000000000000000000000000000000000000000000000000000331c`

Пример №2

Стандартный ввод
45b23c86707f54d7f168f70d62248a0f4ca463cc9c57f7b2865759a4b6a8f9c7 cc2bb0a90ea6c8a0eeace75281c5fc09470996e93eed17cfc3e21a1a061ff45d b00c
Стандартный вывод
00000000000000000000000000000000000000000000000000000000000007e1

Комментарии

В примере входных данных публичный ключ представлен в виде двух разделенных строк из-за форматирования. В реальных входных данных публичный ключ будет записан в одной строке.

Для решения задачи можно скопировать содержимое модуля `secp256k1` Python библиотеки `py_ecc` в код своей программы.

Решение

Пожалуй, наиболее трудным в этой задаче было понять, каким образом необходимо складывать публичные ключи. Обычное сложение тут не подходит, как не подходит и конкатенация массивов байт. Дело в том, что для получения публичного ключа используются эллиптические кривые, и для решения задачи необходимо было использовать функционал, реализующий математику на этих кривых.

В самом условии дана подсказка, в которой советуется использовать модуль `secp256k1` Python библиотеки `py_ecc`. К сожалению, документация к данной библиотеке не представлена, но если зайти в нее, то можно найти целый набор функций, которые реализуют различные операции на эллиптической кривой, используемой в блокчейн ethereum.

Для данной задачи, в частности, можно использовать функцию *add*, которая принимает на вход две упорядоченные пары с хранящимися в них целочисленными координатами двух точек, которые необходимо сложить. На выходе же она возвращает координаты точки, получившейся в результате сложения.

Таким образом, для использования данной функции необходимо перевести публичный ключ в упорядоченную пару из двух целых чисел, которые представляют собой координаты для соответствующей точки на эллиптической кривой. Для хранения координат можно использовать структуру данных - кортеж.

Осталось только подобрать такой приватный ключ, чтобы при сложении с публичным ключом мы получали адрес, начинающийся с искомых символов. Единственный способ сделать это - это полный перебор всех 2^{256} вариантов. К счастью, ответ всегда достаточно близок, чтобы его можно было вычислить.

Пример программы

Ниже представлено решение на языке Python3

```
from py_ecc import secp256k1
from sha3 import keccak_256
from py_ecc.secp256k1.secp256k1 import add
```

```python

# return public key in byteArray
def privToPub(priv):
    priv = int(priv, 16).to_bytes(32, 'big')
    res = secp256k1.privtopub(priv)
    x = res[0].to_bytes(32, 'big')
    y = res[1].to_bytes(32, 'big')
    return x+y

# return address in hex
def pubToAddr(pub):
    return keccak_256(pub).hexdigest()[24:]

# param s: 64 length byteArray
def prepareToAdd(s):
    return (int.from_bytes(s[:32], 'big'), int.from_bytes(s[32:], 'big'))

# return public key in byteArray
def vanityAdd(x, y):
    res = add(x, y)
    return res[0].to_bytes(32, 'big') + res[1].to_bytes(32, 'big')

pub = input()
key = input()

pub = int(pub, 16).to_bytes(64, 'big')
pub = prepareToAdd(pub)

for i in range(2**256):
    i = hex(i)[2:]
    privAdd = privToPub(i)
    privAdd = prepareToAdd(privAdd)
    a = pubToAddr(vanityAdd(pub, privAdd))
    if a[:4] == key:
        print('0' * (64-len(i)) + i)
        break
```

Задача 6.3.9. Цифровая подпись документа (2 балла)

Цифровая подпись – это цифровой аналог рукописной подписи. У цифровой подписи должны быть два свойства, присущи подписи рукописной:

- Во-первых, только конкретный человек может сделать свою подпись и все могут проверить, является ли подпись действительной;
- Во-вторых, подпись, оставленная на документе, не может быть использована для того, чтобы подписать другой документ.

Криптография с публичным ключом на математике эллиптических кривых позволяет реализовать цифровую подпись для доказательства владения данными.

В общем случае, говорят о двух функциях:

$sig = sign(K, message)$ - возвращает цифровую подпись для данного сообщения.

и

$isValid = verify(P, message, sign)$ - проверяет, что данное сообщение было подписано таким ключом, которому соответствует данный публичный ключ.

Видно, что для создания подписи нужен приватный ключ, а для проверки подписи достаточно лишь публичного ключа.

Следовательно:

- подделка цифровой подписи для конкретного сообщения невозможна, поскольку использование публичного ключ, соответствующего данному приватному ключу, продемонстрирует несоответствие подписи
- подмена сообщения невозможно, поскольку текущая цифровая подпись не будет соответствовать новому сообщению

В криптографии эллиптических кривых функция получения цифровой подписи возвращает два числа: r, s. В блокчейн Ethereum к цифровой подписи также добавляется еще одно число v. Оно позволяет упаковать цифровую подпись в структуру размером 65 байт, а также несет информацию о том, на каких узлах сети Ethereum данная цифровая подпись может проверяться.

Обратите внимание, что в функциях генерации и проверки цифровой подписи используется не само сообщение, а его хэш. В блокчейн Ethereum для получения хэша используется алгоритм *keccak*.

Напишите программу, которая для приватного адреса и заданного сообщения генерировала бы цифровую подпись. Для получения хэша сообщения используйте алгоритм *keccak*.

Формат входных данных

В качестве входных данных будут переданы две строки. Первая строка содержит последовательность символов – приватный ключ, записанный в шестнадцатеричном формате. Вторая строка содержит набор символов длиной не больше 255.

Формат выходных данных

Три строки. В первой v-компонента цифровой подписи, записанная в шестнадцатеричном формате (два символа). Во второй и третьей строках r- и s-компоненты, записанные в шестнадцатеричном формате.

Примеры

Пример №1

Стандартный ввод
`f56226b5d46a4c3307ab0283932095716ca5f1ba8d88b19f8ccf4227c09a1d9e` `Ethereum uses public key cryptography for authentication.`

Стандартный вывод

`1b`
`eec36bb56f15d2954d573ffad9e7a9e751479310971ed641787986f8aed7b5b8`
`40505b53f3f976f114cf585139dbf67be79538eb006f6bb7d2ca4bf10cd4b6f2`

Пример №2

Стандартный ввод
e1b791559d56f9a081fb45c8b1a1bdc36c10c0673e00d95497c13ec48b9a5ab8 Digital signatures employ asymmetric cryptography.

Стандартный вывод
1b 6cb8c7d270d8d254cd864e185a54ae90cb84669d012fcf9b4e39bfe245e40acc 67d7630b02c5a672eb5b071b296429ae8a7835b3e9fe022a82974493a6e891db

Решение

Для решения будем использовать функцию *ecdsa_raw_sign* из модуля `secp256k1` библиотеки `py_ecc`.

На вход данная функция принимает два набора байт: данные, закодированные при помощи алгоритма *keccak*, и приватный ключ, - и возвращает три целых числа: v, s и r.

Однако тут важно учесть сколько байт должны быть каждое из данных компонентов.

Согласно спецификации ethereum (`https://ethereum.github.io/yellowpaper/paper.pdf`, *Appendix F*), компонента v является 1-байтным элементом, а компоненты s и r - 32-байтными. Таким образом, перед выводом ответа, обязательно нужно добавить недостающее количество нулей в шестнадцатеричную запись компонентов.

Пример программы

Ниже представлено решение на языке Python3

```python
from py_ecc.secp256k1 import ecdsa_raw_sign
from sha3 import keccak_256

inp = open('input.txt', 'r')
out = open('output.txt', 'w')

n = 5

inp = inp.read()
inp = inp.split('\n')

for i in range(0, n*2, 2):
    priv = int(inp[i], 16).to_bytes(32, 'big')
    data = keccak_256(inp[i+1].encode()).digest()
    ans = ecdsa_raw_sign(data, priv)
    res = ''
    for j in range(3):
        if j > 0:
            res += '0' * (64 - len(hex(ans[j])[2:])) + hex(ans[j])[2:] + '\n'
        else:
            res += '0' * (2 - len(hex(ans[j])[2:])) + hex(ans[j])[2:] + '\n'
    out.write(res)
```

Задача 6.3.10. Проверка подписи в транзакции (5 баллов)

Перед тем, как транзакция Ethereum будет отправлена в сеть, она подписывается на узле с использованием одного из приватных ключей, доступных на данном узле.

Транзакция подписывается не вся, а только следующие данные в ней:

```
nonce
gasprice
gas
to
value
data
```

Перед тем, как быть подписаны, эти данные сериализуются - представляются в виде набора байт с помощью RLP (Recursive Length Prefix) кодирования.

Напишите программу, которая бы проверяла транзакцию на предмет подделки.

Формат входных данных

В качестве входных данных будут передано 8 транзакции в виде JSON-записей их содержимого.

Формат выходных данных

Одна строка - число от 0 до 255, которое, если его записать в двоичном виде, кодировало бы состояние транзакций: если в нулевом бите — 0, то первая в списке транзакция подделана, если — 1, то верификация первой транзакции прошла без ошибок; если в первом бите — 0, то вторая в списке транзакция подделана, если — 1, то верификация второй транзакции прошла без ошибок и т.д.

Примеры

Пример правильной транзакции в формате JSON:


```
{"blockHash": "0x00", "blockNumber": 0,
"from": "0x58f6ab42b6345907fa20aeca0274548dee6e2e32",
 "gas": 90000, "gasPrice": "28000000000", "hash": "0x00", "input": "0x", "nonce": 1,
 "r": "0xcc8350cb3a4390832f6c1af92a8354438d6c0d7029666c414bc88f44bfd2caba",
 "s": "0x7a00af5fc114415494e16a9a657801e1813e5f4193b226b409a05d87dd613de7",
 "to": "0x9a1cdf391b4241da2aca61af6b746e273b24fad9", "transactionIndex": 0,
 "v": "0x26", "value": "1236096691978933262"}
```


Пример некорректной транзакции в формате JSON:


```
{"blockHash": "0x00", "blockNumber": 0,
"from": "0x2ed42f423ee2f457ab6968203ba08a7e564ac0e5",
 "gas": 90000, "gasPrice": "53000000000", "hash": "0x00", "input": "0x", "nonce": 0,
 "r": "0x518eeb19f4b25ddadfab228cc4bdd45b585056544effc78d804b4821da648b87",
 "s": "0x348138d4db546f6f488cb5cd99c3940966ab52627a80fd60c4490f857ab76f99",
 "to": "0x4dc922e93e2e1f1059d72bc7e7e0415c2b22335a", "transactionIndex": 0,
 "v": "0x26", "value": "5944533907010808614"}
```

Комментарии

Для решения задачи используйте модуль `transactions` из Python библиотеки `pyethereum`.

Решение

Для решения задачи воспользуемся указанным в примечании модулем `transactions` из библиотеки `pyethereum`.

Входные данные в том виде, в котором они даются в задаче, будем считывать из файла "transactions.txt".

Создадим новый экземпляр класса Transaction с данной транзакцией. При этом стоит заметить, что при инициализации объекта значения v, r и s должны быть приведены к типу *big_endian_int*, а все префиксы для обозначения системы счисления - удалены.

Далее для каждой транзакции необходимо найти адрес её отправителя, для чего можно воспользоваться функцией `sender`, и сравнить его с адресом, записанным в JSON-записи. При их совпадении можно считать транзакцию правильной. Кроме того, если значения v, r или s будут некорректными функция *sender* может поднимать исключение. Это будет означать, что транзакция была подделана, и обрабатывать этот случай необходимо отдельно.

Пример программы

Ниже представлено решение на языке Python3

```
from ethereum.transactions import Transaction
from ethereum import utils

res = ''
f = open('transactions.txt', 'r')
for i in f.readlines():
    in_tx = eval(i)
    t = Transaction(in_tx['nonce'], int(in_tx['gasPrice']), in_tx['gas'], in_tx['to'],
        int(in_tx['value']), in_tx['input'][2:], int(in_tx['v'], 16),
        int(in_tx['r'], 16), int(in_tx['s'], 16))
    try:
        sender = utils.encode_hex(t.sender)
        if sender == in_tx['from'][2:]:
            res += '1'
        else:
            res += '0'
    except:
        res += '0'
print(int(res[::-1], 2))
```

Задача 6.3.11. Построение Merkle Tree (2 балла)

Одним из базовых методов, обеспечивающих обеспечение целостности данных в блокчейн, является построение дерева Меркле (*Merkle Tree*, `https://en.wikipedia.`

org/wiki/Merkle_tree). Поскольку не всегда именно строится классическое дерево, иногда этот метод называют даже хэш-функцией.

Данный метод позволяет решить две проблемы:

- определить хэш какого-то набора данных и использовать этот хэш для проверки их целостности;
- проверить наличие какого-то конкретного набора данных в дереве, используя *merkle path*.

Самым первым дерево Меркле стало использоваться в блокчейн Bitcoin для получения хэша всех транзакций (*merkle root*), включенных майнером в блок.

Напишите программу, которая бы находила *merkle root* для транзакций в блоке, сформированном в блокчейн Bitcoin.

Формат входных данных

Первая входная строка N содержит число ($2 \leqslant N \leqslant 2000$), определяющее количество транзакций, для которых нужно получить *merkle root*. Последующие N строк содержат хэши транзакций, записанные в шестнадцатиричном формате.

Формат выходных данных

Одна строка – итоговый хэш - вершину дерева *merkle root*.

Примеры

Пример №1

Стандартный ввод
4 8c14f0db3df150123e6f3dbbf30f8b955a8249b62ac1d1ff16284aefa3d06d87 fff2525b8931402dd09222c50775608f75787bd2b87e56995a7bdd30f79702c4 6359f0868171b1d194cbee1af2f16ea598ae8fad666d9b012c8ed2b79a236ec4 e9a66845e05d5abc0ad04ec80f774a7e585c6e8db975962d069a522137b80c1d

Стандартный вывод
f3e94742aca4b5ef85488dc37c06c3282295ffec960994b2c0d5ac2a25a95766

Пример №2

Стандартный ввод
9 ef1d870d24c85b89d92ad50f4631026f585d6a34e972eaf427475e5d60acf3a3 f9fc751cb7dc372406a9f8d738d5e6f8f63bab71986a39cf36ee70ee17036d07 db60fb93d736894ed0b86cb92548920a3fe8310dd19b0da7ad97e48725e1e12e 220ebc64e21abece964927322cba69180ed853bb187fbc6923bac7d010b9d87a 71b3dbaca67e9f9189dad3617138c19725ab541ef0b49c05a94913e9f28e3f4e fe305e1ed08212d76161d853222048eea1f34af42ea0e197896a269fbf8dc2e0 21d2eb195736af2a40d42107e6abd59c97eb6cffd4a5a7a7709e86590ae61987 dd1fd2a6fc16404faf339881a90adbde7f4f728691ac62e8f168809cdfae1053 74d681e0e03bafa802c8aa084379aa98d9fcd632ddc2ed9782b586ec87451f20

Стандартный вывод
2fda58e5959b0ee53c5253da9b9f3c0c739422ae04946966991cf55895287552

Комментарии

В качестве хэш функции в Bitcoin используется стандартный *sha256()* (например, из *Python* библиотеки `hashlib`).

Также еще одной особенностью вычисления хэша двух элементов в дереве Меркле блоков Bitcoin, является двойное хэширование:

$H_{AB} = sha256(sha256(H_A + H_B))$

Обратите внимание, что для хранения числа в Bitcoin используется порядок байт little-endian.

Решение

Для начала рассмотрим, что представляет из себя дерево Меркле. В документации к блокчейну Bitcoin можно найти следующий рисунок (`https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch09.asciidoc`):

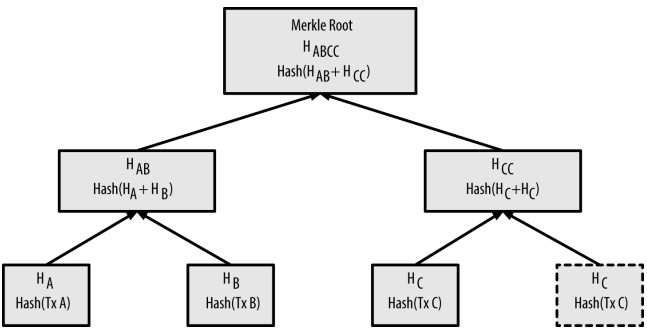


Рис. 6.3: Дерево Меркле

При помощи данного рисунка попробуем разобрать принцип получения *merkle root* в блокчейне Bitcoin. Так, в листьях дерева находятся хэши транзакций, которые хранятся в блоке. Каждая верхняя нода получается при хэшировании результата

конкатенации двух своих детей. Заметим также, что если на каком-либо уровне количество нод нечетное, то мы дублируем последнюю ноду и конкатенируем её с ней же.

Далее, все там же мы можем найти, что представляет из себя функция хэширования в *Bitcoin*. Её результатом будет двойное применение алгоритма *sha256*. То есть, $H_A = sha256(sha256(A))$

И наконец надо вспомнить, что *Bitcoin* использует для кодирования последовательность little-endian. То есть перед тем как применять хэш-функцию к какому-либо значению, предварительно её необходимо представить в виде последовательности байтов, упорядоченных от младшего байта к старшему.

Зная все это, мы можем написать рекурсивный алгоритм по вычислению *merkle root*, который принимает на вход массив из транзакций и возвращает единственное значение - корень дерева Меркле.

Пример программы

Ниже представлено решение на языке Python3

```python
import hashlib, binascii

def hash2(a, b):
    a1 = binascii.a2b_hex(a)[::-1]
    b1 = binascii.a2b_hex(b)[::-1]
    h = hashlib.sha256(hashlib.sha256(a1+b1).digest()).digest()
    return h[::-1].hex()

def merkle(hashList):
    if len(hashList) == 1:
        return hashList[0]
    newHashList = []
    for i in range(0, len(hashList)-1, 2):
        newHashList.append(hash2(hashList[i], hashList[i+1]))
    if len(hashList) % 2 == 1:
        newHashList.append(hash2(hashList[-1], hashList[-1]))
    return merkle(newHashList)

a = []

n = int(input())

for i in range(n):
    a.append(input())

print(merkle(a))
```

Задача 6.3.12. Использование Merkle Patricia Tree (3 балла)

В блокчейн Ethereum вместо вычисления хэша списка транзакций по методу дерева Меркле используется комбинированный подход, который получил название Merkle Patricia Tree.

В основе этого подхода лежит префиксное дерево (*Radix tree*, `https://en.wikipedia.org/wiki/Trie`), которое очень требовательно к потреблению памяти, поэтому для

решения этой проблемы в блокчейн Ethereum было предложена оптимизация, так появился метод построения дерева *Patricia Tree*. Чтобы сохранить возможность подсчитывать итоговую хэш сумму всех конечных элементов дерева в алгоритм были введены механизмы, использующиеся при построении дерево Меркле, и конечное название метода вычисления хэш суммы для какого-то списка данных стал называться *Merkle Patricia Tree*:

- `https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/`;
- `https://github.com/ethereum/wiki/wiki/Patricia-Tree`;
- `https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/`

Для построения дерева используются пары *ключ*: *значение*. *Ключ* определяет путь в дереве, *значение* помещается в конечный элемент ("лист") дерева. Промежуточные узлы дерева также представляются в виде пары *ключ*: *значение*, обозначая промежуточный путь и промежуточный хэш соответственно.

В случае списка транзакций в качестве *ключа* используется порядковый номер транзакции, выданный ей при включении в блок: $rlp(transactionIndex)$. В качестве *значения* используется хэш транзакции.

Напишите программу для определения количества непустых конечных элементов во всех промежуточных узлах (*extensions*), сформированных при построении *Merkle Patricia Tree* для списка транзакций в блоке Ethereum.

Формат входных данных

Три строки, каждая из которых содержит число N ($256 \leqslant N \leqslant 4370$), определяющее количество транзакций, включенных в блок.

Формат выходных данных

Каждой входной строке сопоставляются три выходных строки, в каждой из которых по два числа, разделенных пробелом.

В каждой строке первое число - количество непустых элементов, принадлежащих непосредственно промежуточным (*extensions*) узлам, полученных при построении дерева транзакций. Второе - количество всех дочерних конечных, но не являющихся поддеревьями, элементов в промежуточных узлах. Выведите 0 0, если промежуточные узлы не были сформированы при построении дерева.

Примеры

Пример №1

Стандартный ввод
256 4099 330

Стандартный вывод
0 0 3 3 5 74

Комментарии

Примечание к примеру: для 4099 транзакций при построении дерева сформировался 1 промежуточный узел, который содержит только 3 элемента, а для 330 транзакций при построении дерева сформировался 1 промежуточный узел, который содержит только 5 поддеревьев, в которых содержится 74 элемента.

Решение

Для построения дерева воспользуемся модулем `trie` библиотеки `ethereum`. При помощи данного модуля мы можем построить дерево для списка транзакций. Как было указано в задании в этом случае в качестве ключа используется порядковый номер транзакции. Заметим, что нам не особо важно что использовать в качестве значений, так как это не должно влиять на само дерево.

Таким образом, самым первым шагом необходимо построить дерево из указанного количества транзакций, а затем, используя построенное дерево, надо посчитать количество непустых элементов, принадлежащих непосредственно промежуточным узлам, и количество всех дочерних конечных, но не являющихся поддеревьями, элементов в промежуточных узлах.

Однако для того чтобы полностью обойти дерево и найти количество необходимых узлов и листьев, необходимо прежде всего понять, что же эти узлы значат и каким образом они располагаются в дереве.

На приведенном ниже рисунке схематично изображено Merkle Patricia Tree. Все значения кодируются в шестнадцатеричной системе счисления, поэтому максимальное количество детей у узла равно 16 (по одному на каждый возможный вариант следующего символа ключа). Существует 4 разновидности узлов:

1. Blank (пустой) node - представленный как пустой набор байт.
2. Leaf (конечный) node - узел, представляющий собой обычную пару [ключ, значение].
3. Extension (промежуточный) node - узел, которых также представляет собой пару [ключ, значение], однако в данном узле. значение является хэшом другого узла.
4. Branch (ветвления) node - массив, состоящий из 17 узлов, каждый из которых представляет собой один из 16 возможных следующих символов в ключе и значение для данного ключа, если оно существует.

Как можно заметить, за каждым промежуточным узлом следует узел ветвления, а вот из узел ветвления может вести как в промежуточный узел, так и в конечный.

Наша задача - посчитать сколько узлов в узле ветвления являются не пустыми, и сколько листьев можно достичь пройдя через промежуточный узел.

Для этого напишем рекурсивную функцию обхода всего дерева, которая будет принимать на вход само дерево, текущий узел, количество непустых элементов, принадлежащих промежуточным узлам, переменную, хранящую значение, был ли на данном пути пройден хотя бы один промежуточный узел, и количество конечных элементов в промежуточных узлах.

Пример программы

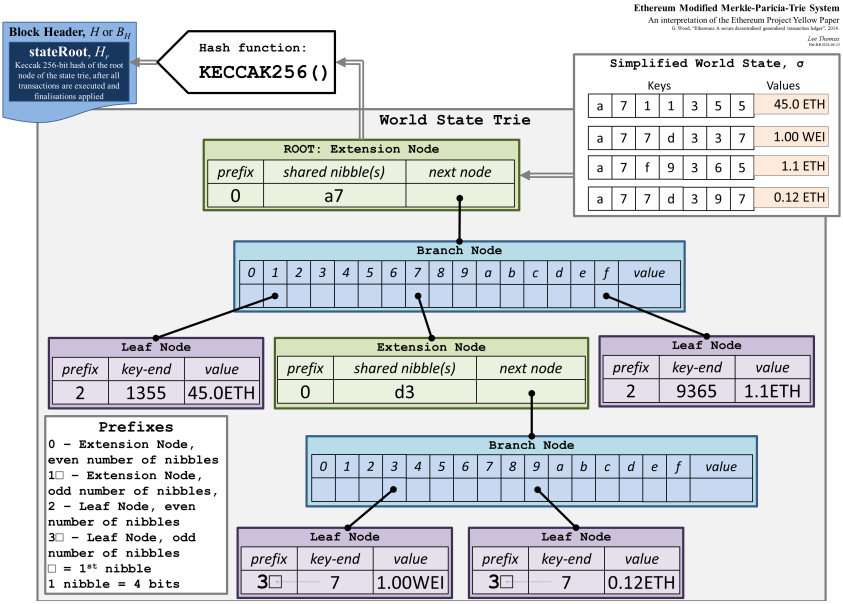


Рис. 6.4: Patricia Trie

Ниже представлено решение на языке Python3

```python
from ethereum import db
from ethereum import trie
from ethereum import utils
import rlp

def countExtensionNodesWithLeaves(state, node, ext_size, flag, leaves_num):
    if state._get_node_type(node) == trie.NODE_TYPE_LEAF and flag:
        leaves_num += 1
    if state._get_node_type(node) == trie.NODE_TYPE_EXTENSION:
        temp = state._decode_to_node(node[1])
        for i in state._decode_to_node(temp):
            if i != b'':
                ext_size += 1
        flag = True
        ext_size, leaves_num = count_extension_nodes(state, state._decode_to_node(temp),
            ext_size, flag, leaves_num)
    elif state._get_node_type(node) == trie.NODE_TYPE_BRANCH:
        for i in node:
            if i != b'':
                temp = state._decode_to_node(i)
                ext_size, leaves_num = count_extension_nodes(state, temp,
                    ext_size, flag, leaves_num)
    return ext_size, leaves_num

res = []

for i in range(3):
    n = int(input())
```

```
    state = trie.Trie(db.EphemDB(), trie.BLANK_ROOT)
    for i in range(0, n):
        state.update(rlp.encode(i), b'')

    res.append(countExtensionNodesWithLeaves(state, state.root_node, 0, False, 0))

for i in res:
    print(*i)
```

Задача 6.3.13. Дерево состояний (7 баллов)

Одним из нововведений, которое было сделано в блокчейн Ethereum, является введение понятия «состояния». Его смысл позаимствован из *Теории автоматов*. Теперь, когда говорится о выпуске нового блок одним из майнеров, то это значит, что блокчейн переходит в новое состояние. Набор транзакций, входящих в блок, определяет каким именно это новое состояние является: какое стало состояние баланса на адресах пользователей, как изменилось внутреннее состояние умных контрактов.

Например, пусть блокчейн был в некотором состоянии S_1, когда баланс адреса $A_{S_1} = 50\ ether$, а баланс адреса $B_{S_1} = 10\ ether$. Тогда, при появлении нового блока с транзакцией перевода 15 $ether$ с адреса A на адрес B и распространении этого блока на все узлы сети, блокчейн перейдет в состояние S_2, которое характеризуется тем, что баланс адреса $A_{S_2} = 35\ ether$, а баланс адреса $B_{S_1} = 25\ ether$.

Для того, чтобы отслеживать соответствие состояния на всех узлах сети Ethereum, в каждый блок майнеры включают хэш новой вершины *Merkle Patricia Tree*, которое хранит информацию о состоянии каждого конкретного адреса и поэтому называется дерево состояния блокчейна (*World State Tree*).

Иными словами, в простом случае, алгоритм изменения состояния блокчейн Ethereum можно описать следующим образом:

1. Выбрать из списка неподтвержденных транзакций те, которые планируется включить в новый блок.
2. Проверить корректность каждой транзакции:
   (a) Адрес, получаемый от проверки цифровой подписи, не нулевой;
   (b) Номер транзакции (nonce), совпадает с ожидаемым номером, хранимым в записи соответствующего адреса в дереве состояний блокчейна, обозначающим сколько транзакций отправитель уже послал;
   (c) Ограничению по использованию газа (газ - условная единица, обозначающая сколько потребуется Ethereum Virtual Machine ресурсов, таких как память и процессорное время, для обработки транзакции), указанное отправителем в транзакции, не меньше, чем количество газа, требуемое для хранения информации, переданной в транзакции;
   (d) В дереве состояния блокчейна на балансе адреса, полученного от проверки цифровой подписи, достаточно количества средств для списания.
3. Для каждой транзакции в дереве состояния блокчейна изменить баланс соответствующих адресов: уменьшить на том адресе, который получился после проверки подписи, и увеличить на адресе-получателе. А также увеличить

номер ожидаемой транзакции у соответствующих адресов.

4. Увеличить баланс адреса, выставленного как coinbase на данном узле.
5. Включить все транзакции в новый блок и отправить его соседним узлам для дальнейшего распространения по сети.

Как только новый блок достигает других участников сети, каждый узел сети Ethereum выполняет следующие операции:

1. Проверка блока на корректность (здесь не будет рассматриваться подробно).
2. Для каждой транзакции включенной в блок проверяется корректность, выполняя шаги 2.a - 2.d, описанные выше.
3. Для каждой транзакции в дереве состояния блокчейна изменить баланс соответствующих адресов: уменьшить на том адресе, который получился после проверки подписи, и увеличить на адресе-получателе. А также увеличить номер ожидаемой транзакции у соответствующих адресов.
4. Увеличить баланс адреса, выставленного как coinbase в блоке.

Если в какой-то момент на шаге 2 будет обнаружена некорректная транзакция, то блок будет отброшен, а значит, ни одна транзакция, включенная в этот блок, не будет использована для изменения баланса адресов. В дальнейшем, если на данный узел не получит другой блок, но уже корректный, с таким же номером, то узел перестанет быть сихнронизированным с остальной сетью.

При этом, поскольку одни и те же действия по изменению баланса адресов выполняются на всех узлах, то дерево состояния блокчейна (*World State Tree*) на всех узлах одно и тоже. Все узлы должны обновлять это дерево с при получении новых блоков для того, чтобы иметь возможность создавать новые блоки, которые будут приняты другими участниками сети. Следовательно, участники сети будут признавать право на получение награды за майнинг и позволять распоряжаться полученными в качестве награды средствами.

Напишите программу для проверки на корректность транзакций в блоках. Под корректной транзакцией в данном задании будет подразумеваться такое изменение баланса адреса Ethereum, которое приводит к значению большему, либо равному нулю.

При подсчете баланса не учитывать оплату газа на обработку транзакции.

Формат входных данных

Первая входная строка содержит число N ($5 \leqslant N \leqslant 1000$), определяющее количество блоков. Следующие N наборов строк описывают транзакции, включенные в блоки следующим образом:

- Первая строка в наборе состоит из двух значений - количества транзакций в блоке и адреса, получающего награду за данный блок, записанного в виде 40 шестнадцатеричных символов. Величина награды - $5 \cdot 10^{18}$ условных единиц криптовалюты.
- Оставшиеся строки в наборе - транзакции в формате Ethereum, закодированные с помощью RLP и записанные в шестнадцатеричном виде.

Формат выходных данных

Вывести сумму на балансе пяти адресов с самым большим остатком на момент обнаружения некорректной транзакции. В случае, если некорректных транзакций нет, вывести сумму на балансе пяти адресов с самым большим остатком на момент применения самого последнего блока.

Примеры

Пример №1

Стандартный ввод

```
6
0 5230b4da9c28bfb089b8dd7bd6898a1594dba960
3 f75c80053674ee58a99cef9ee3b0ff19f112c795
f86d0186246139ca800082520894c6c4571388ecc770a25c4aeb30f9bbe16459371a882c68af0bb
14000008026a0da384535815daa0e910354d74f3a7b35dcd427baa31366595b6f6dc6baa96341a0
18a7ab431c67bd795ea16a8431eb6c5cff5d440c9699752b6e27770df9fed86b
f86d028612309ce540008252089427 2ac4bccc024692903951996b7e4c8b3c19ffc28806f05b59d
3b200008025a09dd726bb23005f20910a4f749a5113e2eba40e35ced62d64bdedcbe681bc1a1fa0
6a4ec0d1f81e1aab839a19a35780af2946ab93f455fe71c509338a2ed76d6fe9
f86d01863205af7670008252089466d3144d6ee8be6a340bea305b13ad202fe6c66f880429d0691
89e00008026a094e3551725a9cd9693a8e923a56155e02ea55b2ece0cd2be15fee9c6c908024aa0
4db3e2303cecb5aed9358b14cfc5fec7be271eb145c499f57a6b74f00304d88d
1 bf0b5a4099f0bf6c8bc4252ebec548bae95602ea
f86d018615d3ef79800082520894ffaf33a6f1f7061b75ce510fbc2ee1a443befb1188058d15e17
62800008025a005ff35ad86a69e28609ae696b2509ea6be3225cf8a9c73bddb041bbdff381366a0
304f44088648e8a4e4c5af9fdba1995241e3ffee13f1bd618c273750b5df0a53
f86d0186246139ca80008252089485bf78837156632e92a80ebe160f64b228994a24880c7d713b4
9da00008025a04bf24fa514dd5264091afff040c0f7533991cd2a31faa36fd63ae583e4cb1e7ba0
46bee665e7319af5da61955931584cabb85cfd8ea91b20fdeeca117ae3f11b8b
5 53a7522e00e530f1076c4056a1cbb47229a39dd6
f86d01861b48eb57e0008252089402bd7d6559bb2597f6515390fae4035ed3a4186e880853a0d23
13c00008025a0bc7eff3689ff290e9b7675508229c57a8ac5d9a044eb884ec92f4085b6e8a6a4a0
240116d133f2e5cced21cb0ea6abc98a942e2700135367b283c07928c3c8d60f
f86d01862d79883d2000825208949702014 40c7bc5689554107ba3f67be0395176f4880c7d713b4
9da00008026a082a236df87e79a872e8a8caaff7c1e4952b15a63e995ce54745b23d106df6b71a0
1a223c0404a7bed357c6523920bbe3fa8c5252ff31d540a97972516ec411b751
f86d02861147c84030008252089416ac683d6c7504ad0b31e808b2e991575cc7e99e88016345785
d8a00008025a013b02f8013baf3b53059b5c951b073b04478fe16557b438e45b8553a91e5afb6a0
23ad05f9d8475bdde280808e8bc55015631cd17395a430dad2dfedd3af1fea13
f86d01861402462f6000825208944d7000adbfac757f9e1337f3033bd33eaeb969078802c68af0b
b1400008026a0ecfeabff10c6cc8d2783359d11c01a4ee29eb52763cfd726eb28cca55f05e441a0
08b68423c874fe23a959fc7d157fc1f85174396896906d782f2a59eafbf7301b
f86d038632ee841b800082520894abd46c781f71a836c941e6fc4d245cb31b0b60da880853a0d23
13c00008026a03f79c560dc049eb19ad7e9b169b65fc07d801f889733dbe78e3d53f5a92e17a3a0
24630de8f523261b36634b13d31fd81e48ccba621c47aaafa26d15ce99036bf9
1 7d050326fdf9c1d94495f22dd3ab8807422f1a96
f86d018617a598c3a00082520894bf0b5a4099f0bf6c8bc4252ebec548bae95602ea88016345785
d8a00008026a03742e9ebfb32039c8104c6853ce01ed505f2a7220e5414aaecdc5a68d0b654a3a0
4be302869f0219688c263b66205ec3b0e0e0191a64335c14dcbc026c9404cde8
```

Стандартный вывод

```
18300000000000000000
```

Пример №2

Стандартный ввод
7 0 db6c6e77162e0e023eb9abd972621265202772e3 0 f3e63cff4f7f3202abe9126658e9b3359a24243a 0 4943b96530bab4678d68fa093a8d7bc003cadd52 1 f75c80053674ee58a99cef9ee3b0ff19f112c795 f86d01861a6016b2d00082520894bca0bc749270a42ab2d747ff62a949ddeba93dae88136dcc951 d8c00008025a0290cd60459f616232a71eae9ca05885e9aca557eb6dfcf953b0b886a09c8fb8ca0 5afb526545a5b9434835994950272d191d389cad02dfa04fe2728175a823d157 0 c948603e1b4b553c3e1fc3d99e95d5260d667fca 0 53a7522e00e530f1076c4056a1cbb47229a39dd6 3 02178117a34fe7e25b8454b0d55cb00c91b1c6a4 f86d0186105ef39b2000825208947abf7cde5d983137d9b09487b5091a3033953461880b1a2bc2e c5000008025a0145ccdf0f330596629dcda12c31d785fd4a9ff0a34f9899a699bceef755317e6a0 6847afb566c391afc3c166b908bc19f0843016fb4d70136ecfbe6a1df8bf1758 f86d01861a6016b2d000825208940839702e3f61f7fed6e058f4a5f08f3bd23be74f88016345785 d8a00008026a0043bce12b091e17858852623889c798ca61106a65dc8dfc420938ce4250bc211a0 255ab21e6be35d7bfdbed86831ff621b41e1cd67ff49392e2b1e6038cf056e5b f86d018628ed6103d0008252089485bf78837156632e92a80ebe160f64b228994a2488016345785 d8a00008025a061d3d728042f6051a42a91c3416b3e18cffc7e7d190b5d8fc1c0978f0d167108a0 703e094e029dcbf8d5022f23e02bd077ea35ba217a59829f04b7046836855baf

Стандартный вывод
25000000000000000000

Решение

Для решения воспользуемся тем функционалом, который предоставляют нам библиотеки `ethereum`, а в частности модулями `transactions`, `utils` и `rlp`.

Так как транзакции закодированы в шестнадцатеричном формате, то перед тем, как декодировать их при помощи rlp, нам необходимо конвертировать их в массив байт. Для этого воспользуемся функцией *decode_hex* модуля *utils*.

Полученный массив байт необходимо десериализовать, используя rlp. Заметим, что модуль *rlp* позволяет сериализовывать свои структуры данных, в частности, мы можем сериализовать объект класса Transaction, конструктор которого находится в модуле *transactions* библиотеки *ethereum*. Соответственно, с таким же успехом мы можем десериализовать объект данного класса. Таким образом, мы получаем возможность работать с данными транзакциями, используя все преимущества этой структуры данных.

Далее, необходимо заметить, что майнер получит награду только в том случае, если выпущенный им блок окажется правильным. То есть, в начале необходимо проверять правильность блока и лишь затем присваивать майнеру награду. Более того, если какая-либо транзакция в блоке оказалась некорректной, то все транзакции в блоке являются недействительными и никак не влияют на состояние сети.

Принимая во внимание описанные выше ограничения, напишем функцию, которая будет возвращать состояние аккаунтов в сети к моменту обнаружения некорректной транзакции. Входные данные будем считывать из файла, а изменения на балансах будем производить только после проверки правильности конкретного блока.

Пример программы

Ниже представлено решение на языке Python3

```python
from ethereum import transactions
from ethereum import utils
import rlp

# Возвращает состояние балансов на момент обнаружения некорректной транзакции
def get_state(f):
    balances = dict()
    n = int(f.readline())
    for i in range(n):
        t_balances = balances.copy()
        tx_num, miner = f.readline().strip().split()
        for j in range(int(tx_num)):
            tx_hash = f.readline().strip()
            tx = (rlp.decode(utils.decode_hex(tx_hash), transactions.Transaction)).to_dict()
            sender, to, value = tx['sender'][2:], tx['to'][2:], int(tx['value'])
            if t_balances.get(sender, 0) - value < 0:
                return balances
            t_balances[sender] -= value
            t_balances[to] = t_balances.get(to, 0) + value
        balances = t_balances.copy()
        balances[miner] = balances.get(miner, 0) + 5*10**18
    return balances

f = open('input.txt', 'r')
balances = get_state(f)

sorted_balances = sorted(list(balances.values()), reverse=True)

res = 0
n = len(sorted_balances)
for i in range(min(n, 5)):
    res += sorted_balances[i]

print(res)
```

Задача 6.3.14. Доказательство работы (5 баллов)

Принято считать, что алгоритмы доказательства работы (Proof-of-Work, PoW) представляют из себя некую математическую "головоломку", которую должен решить узел перед тем, как выпустить новый блок. "Головоломка" (англ. *puzzle*) заключается в том, что узел должен найти число или последовательность байт, удовлетворяющих определенным условиям. Эти условия, чаще всего, зависят от конкретного момента времени, а точнее данных, которые доступны только в данный момент времени, поэтому "головоломку" нельзя решить заранее. Также ответ к "головоломке" не поддается непосредственному вычислению: единственный способ - угадать его или подобрать, перебрав из нескольких возможных вариантов, в то время как проверить правильный ли это ответ для конкретных данных другие участники сети могут относительно просто и без использования дополнительного оборудования.

Узел нашедший ответ, перебрав его из нескольких вариантов, как бы заявляет: «Я отправляю информацию, приложив к ней доказательство того, что я в полном праве эту информацию отправлять, поскольку это право я приобрел, выполнив значительный объем работы для поиска данного доказательства.»

Использование алгоритма доказательства работы в блокчейн преследует несколько целей:

1. Выполнение долгого по времени перебора для поиска подходящего ответа позволяет защитить информацию, записанную в блокчейн от изменения. Каждый, кто захочет изменить данные "задним числом" должен подобрать новое число для измененной записи. А поскольку между записями существует связь, когда последующая запись ссылается на предыдущую, то нужно новое подобрать новое число для каждой последующей записи. Чем их больше, тем дольше придется подбирать новые числа, а за это время могут появиться новые записи, что снова добавит работы злоумышленнику.

2. Поскольку заранее невозможно предугадать сколько займет времени подбор нового числа, то эта неопределенность позволяет осуществить выбор узла, который выпустит новый блок транзакций. Разная вычислительная производительность систем, разное количество транзакций, накопившихся у каждого элемента сети, делают процесс выбора узла вероятностным. С точки зрения стороннего наблюдателя кажется, что выбор узла, который выпустил новый блок, происходит случайным образом.

3. За счет перебора узел не может посылать блоки очень часто, что защищает сеть блокчейн от "наводнения" (*flooding*) фальшивыми блоками. Как было сказано выше, проверка выполненной работы осуществляется очень быстро, поэтому если блок приходящий в сеть не отвечает условиям проверки, то узлы начинают его игнорировать и перестают пересылать дальше соседним узлам.

В блокчейн Ethereum для доказательства используется алгоритм *Ethash* (`https://github.com/ethereum/wiki/wiki/Ethash`), взявший свое начало от алгоритма *Dagger-Hashimoto*. Основным свойством данного алгоритма является то, что он не столько требователен к вычислительной мощности, сколько к объему памяти: узел, желающий выпускать новые блоки с высокой частотой, так чтобы конкурировать с другими узлами также занимающимися майнингом, должен иметь не меньше 2 гигабайт оперативной памяти.

Данное свойство делает неоправданно дорогим выпуск и продажу ASIC (*Application-specific integrated circuit*, `https://en.wikipedia.org/wiki/Application-specific_integrated_circuit`), что позволяет владельцам относительно недорогого оборудования (например, графических карт) участвовать в выпуске новых блоков, а значит не дает сконцентрировать вычислительные мощности, направленные на майнинг, в одной или нескольких "богатых" организациях, а значит не дает возможность им единолично принимать решения о том какие блоки должны быть выпущены и включены в цепочку блоков.

Алгоритм Ethash полагается на огромный ($\geqslant 2Gb$) массив данных, которые сгенерированы псевдослучайным образом на базе текущего момента времени. Если быть более точным то один и тот же массив данных действителен в течение 30000 блоков, после чего он перестраивается заново для следующих 30000 блоков. Причем изменяется не только содержимое массива, но и его длина: каждую итерацию она растет на $\approx 8Mb$. Время, в течение которого существует один и то же набор данных, называется эпохой.

При формировании массива используется направленный ациклический граф (Directed acyclic graph, DAG, `https://en.wikipedia.org/wiki/Directed_acyclic_graph`):

- Сначала строится небольшой ($\approx 32Mb$) кэш, его наполнение зависит от номера эпохи - он используется для генерации псевдослучайных последовательностей. Кэш состоит из элементов по 64 байта каждый.
- Затем, строится большой массив данных из элементов по 64 байта каждый.

При вычислении каждого элемента используется 257 элементов кэша, которые выбираются псевдослучайным образом, управляемым номером элемента в массиве данных.

В дальнейшем построенный массив данных используется для алгоритма доказательства работы, а именно ищется такая последовательность из 8 байт, называемая *nonce*, которая при объединении с хэшированными данными из заголовка блока и 128 элементами массива данных даст значение определенного свойства. Причем псевдослучайная выборка элементов массива обуславливается как *nonce*, так и данными из заголовка блока (`https://www.vijaypradeep.com/blog/2017-04-28-ethereums-memory-hardness-explained/`). Свойство, которому должно удовлетворять полученное значение, определяется величиной *difficulty*. Значение *difficulty* разное от одного блока к другому и зависит от частоты выпуска блоков таким образом, что позволяет удерживать периодичность появления новых блоков примерно на одном уровне, даже если вычислительная мощность сети возрастает. Изменение *difficulty* позволяет выдерживать примерно одинаковое время задержки между появлением в сети двух соседних блоков, что важно для задания "тактов" синхронизации узлов в сети блокчейн и обеспечивает целостность данных.

В итоге, поскольку майнеру, узлу подготавливающий новый блок, во время перебора *nonce* приходится выполнять операции обращения к элементам массива данных очень много и часто, а скорость обращения к массиву критична для победы в гонке за награду за новый блок, то ему необходимо держать этот массив в памяти постоянно.

Узлы, которые только получают новые блоки и не выпускают новые, могут хранить только кэш. Для того, чтобы проверить, что новый пришедший блок содержит правильное доказательство работы, можно пробежаться по $128 * 257$ элементам кэша и убедиться, что полученное значение соответствует заданному *difficulty*.

Напишите программу, которая бы находила последовательность байт (*mixHash*), получающуюся в ходе процесса выпуска нового блока для заданного *nonce* и *difficulty*.

Формат входных данных

Описание полей заголовка блока в JSON-формате.

Формат выходных данных

Необходимо вывести *mixHash* - последовательность из 32 байт в шестнадцатеричном формате.

Примеры

Пример №1

Стандартный ввод

```
{
    "difficulty": 175807,
    "extraData": "0xd583010702846765746885676f312e39856c696e7578",
    "gasLimit": 4712388,
    "gasUsed": 21772,
    "logsBloom": 0,
    "miner": "0xd6c76b3ca5c36ced890f0fa70d884b42de96d0cb",
    "nonce": "0x5a0ce998ca547aa5",
    "number": 1912,
    "parentHash": "0xfe6ffd6e80e878367f63b4454a8d834b5561242241df76b4dfbfe83e55
177806",
    "receiptsRoot": "0x590e8d789b4dff06cbf8e561d13df39dd23542874cc1b5b296a368b4
f71a7105",
    "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40
d49347",
    "stateRoot": "0x7b43df1701d2a7709285b75fa9024f3e4db3b1a598fec63eb44cdeee1a3
bc561",
    "timestamp": 1508783678,
    "transactionsRoot": "0xaa639781610199b925119e24d868d32b689eb2676d0e1e4e1dd3
13e95db6e1a6"
}
```

Стандартный вывод

```
5fa4a4dffaa1111990fb02ee453626d24f6f7db9bc467044f127c071bc894119
```

Решение

Для решения этой задачи будем использовать модули `block` и `utils` библиотеки `ethereum`, а также библиотеку `pyethash`. Входные данные в том виде, в котором они даются в задаче, будем считывать из файла 'ethash.txt'.

При помощи конструктора из модуля *block* создадим новый заголовок блока, включив туда значения из входных данных. Это делается для удобного получения заголовка блока, необходимого для майнинга.

Кэш можно получить при помощи функции *mkcache_ bytes*, которая на вход принимает номер блока, для которого нужно посчитать кэш, а возвращает набор байт, необходимый для генерации данных.

Так как мы знаем значение nonce, то для получения mixHash мы можем воспользоваться функцией *hashimoto_ light*, которая использует кэш для создания только тех значений из датасета, которые необходимы, и таким образом позволяет не генерировать полный набор данных. На вход функция получает номер блока, сгенерированный кэш заголовок блока и nonce, а возвращает значения mixHash и result. В процессе майнинга result используется для определения валидности значения nonce при данном difficulty. Так, result должно быть меньше чем $\frac{2^{256}}{difficulty}$. А значение mixHash используется для проверки на то, что во время процесса майнинга был использован правильный DAG.

Пример программы

Ниже представлено решение на языке Python3

```
1 from ethereum.utils import encode_hex, decode_hex, big_endian_to_int
2 from ethereum.block import BlockHeader
```

```
import pyethash

with open('ethash.txt', 'r') as f:
    desc = eval(f.read())

blockHeader = BlockHeader(decode_hex(desc['parentHash'][2:]),
    decode_hex(desc['sha3Uncles'][2:]),
    desc['miner'][2:], decode_hex(desc['stateRoot'][2:]),
    decode_hex(desc['transactionsRoot'][2:]),
    decode_hex(desc['receiptsRoot'][2:]), desc['logsBloom'],
    desc['difficulty'], desc['number'],
    desc['gasLimit'], desc['gasUsed'], desc['timestamp'],
    decode_hex(desc['extraData'][2:]),
    nonce=desc['nonce'])

b_number = blockHeader.number
b_hash = blockHeader.mining_hash
b_nonce = decode_hex(blockHeader.nonce[2:])
cache = pyethash.mkcache_bytes(b_number)

ans = pyethash.hashimoto_light(b_number, cache, b_hash,
                               big_endian_to_int(b_nonce))[b'mix digest']

print(encode_hex(ans))
```

Задача 6.3.15. Отправка транзакций (3 балла)

Базовой операцией, которая может изменять состояние блокчейн Ethereum, является транзакция. Самое простое действие для изменения состояния - отправка средств с адреса на адрес. Для этого в транзакции указывается адрес получателя и сумма, которая пересылается. Если транзакция отправляется интерактивно через JavaScript консоль или в приложении непосредственно помощью библиотек `web3` для Python или JavaScript, то также указывается адрес отправителя.

Узел, на котором в итоге формируется транзакция для отправки в сеть, пытается получить доступ к приватному ключу отправителя и подписать им транзакцию. Для того, чтобы узел успешно смог выполнить операцию, чаще всего, необходимо предварительно разблокировать соответствующий адрес на нем с использованием пароля доступа к приватному ключу.

Помимо суммы, указываемой в поле "value", можно также передавать данные через поле "data". Эти данные обычно не используются программным обеспечением кошельков, и поэтому к ним сложно получить доступ - для этого нужно вызывать метод `getTransaction` модуля `eth` в консоли или в приложении, использующем `web3` библиотеку.

Обработка каждой транзакции на узле, включающем их в новый блок (при условии, что потом блок будет принят остальной частью сети Ethereum), потребляет газ. Газ - понятие, введенное в блокчейн Ethereum, чтобы контролировать ресурсы потребляемые виртуальной машиной Ethereum (Ethereum Virtual Machine, EVM). Например, обработка обычной транзакции перевода средств с адреса на адрес потребляет 21000 газа.

Предполагаемое количество газа (поле "gas"), которое может потратиться в ходе обработки транзакции умножается на цену за единицу газа ("gasPrice") и вычитается с баланса адреса, отправившего транзакцию. Если выясняется, что у адреса на ба-

лансе не хватает средств для обработки транзакции, то она не включается в блок и не обрабатывается.

Газ тратится не только на обработку информации, но также и на ее хранение, поэтому он потребляется за каждый байт, переданный как "data" в теле транзакции.

Итоговое количество газа, потраченное в ходе включения транзакции в блок, можно узнать в поле "gasUsed" при вызове метода `getTransactionReceipt` модуля `eth`.

Отправьте в сеть Ethereum две транзакции с разными данными в поле "data" и определите, на сколько в одной транзакции потреблялось больше газа, чем в другой.

Формат входных данных

Во входном файле - две строки, определяющие данные, которые должны быть отправлены. Первая строка для первой транзакции, вторая строка - для второй. Данные указаны в шестнадцатеричной системе счисления.

Формат выходных данных

Одно число – разница потребленного газа между двумя транзакциями. При указании ответа нужно использовать модуль числа.

Примеры

Пример №1

Стандартный ввод
`0000000022957a00000000000000006e0000b3495300` `f10019070000b9bb66b717004e000056d9`

Стандартный вывод
`236`

Решение

Настроим и подключимся к узлу Ethereum по RPC, как это указано в инструкции.

Адреса аккаунтов хранятся в отдельном файле. При этом каждый адрес хранится в шестнадцатеричном представлении с префиксом '0x' (Например, 0xf6e1fde85ac7a238cf905e5e8f69a28a34e72904). Web3 может выдавать исключение с описанием, что адрес имеет неверную *EIP* проверочную сумму. Это может происходить из-за того, что буквенные значения адреса, которые должны быть в верхнем регистре, написаны в нижнем. В этом случае имеет смысл явно привести адрес к EIP-55.

Данные, которые необходимо отправить, также должны быть в шестнадцатеричном формате с префиксом '0x'.

Для отправки транзакции воспользуемся методом *sendTransaction* из модуля `eth` библиотеки `web3`, которая возвращает хэш транзакции.

Осталось найти разницу между затраченным количеством газа, для чего можно воспользоваться описанной в задании функцией *getTransaction*, и вывести результат.

Пример программы

Ниже представлено решение на языке Python3

```python
from web3 import Web3, HTTPProvider

web3 = Web3(HTTPProvider('http://localhost:48002'))
eth = web3.eth
personal = web3.personal

accounts = []
f = open('accounts.txt', 'r')
for i in f:
    accounts.append(i.strip())
f.close()

with open('input.txt') as f:
    data1 = '0x' + f.readline().strip()
    data2 = '0x' + f.readline().strip()

personal.unlockAccount(accounts[0])

transaction1 = eth.sendTransaction({'from': accounts[0], 'to': accounts[1],
                                    'value': 1, 'data': data1})
transaction2 = eth.sendTransaction({'from': accounts[0], 'to': accounts[1],
                                    'value': 1, 'data': data2})

print(abs(eth.getTransaction(transaction2)['gas'] - eth.getTransaction(transaction1)['gas']))
```

Задача 6.3.16. Получение баланса (5 баллов)

Транзакции формируют блоки, а затем между двумя соседними блоками образуется связь по принципу ”предыдущий” – ”следующий”. Каждый блок фиксирует набор изменений, выполняемых транзакциями, поэтому блок можно контрольной точкой, определяющей "моментальный снимок"данных в блокчейн.

Таким образом, в памяти блокчейн хранятся все "моментальные снимки"и можно проследить историю изменения любых данных, например, баланса адресов.

Баланс адреса в блокчейн Ethereum измеряется в *wei* - самая маленькая единица измерения, которая составляет 10^{-18} от одного *ether*. Вот список всех единиц измерения:

Единица измерения	Значение
wei	1
kwei	1.000
ada	1.000
mwei	1.000.000
babbage	1.000.000
gwei	1.000.000.000
shannon	1.000.000.000
szabo	1.000.000.000.000
finney	1.000.000.000.000.000
ether	1.000.000.000.000.000.000

Когда отправляется транзакция для перечисления средств с одного счета на другой, то сумма для перевода указывается в *wei*. Это достаточно не удобно для человека. Но для виртуальной машины Ethereum так проще - не нужно иметь дело с дробными значениями, и, следовательно, с проблемами округления.

Напишите программу, которая бы позволяла бы получать баланс адресов на момент выпуска определенного блока.

Формат входных данных

В первой строке входного файла - одно число - номер блока, после обработки которого сформировался новый баланс на адресе.

Вторая строка - единицы измерения (ether, finney, shannon, ..., wei), в которых нужно предоставить ответ.

Дальше идет 20 строк, в каждой из которых - Ethereum адрес в тестовой сети Ropsten, записанный в шестнадцатеричном виде. Длина адреса - 40 символов.

Формат выходных данных

Одно целое число – максимальный баланс среди адресов, указанных во входных данных, сформировавшийся в World State тестовой сети Ethereum (Ropsten) после выпуска соответствующего блока. При переводе из одних единиц измерения в другие округление нужно производить отбрасыванием дробной части. Баланс сравнивать до перевода единиц измерения.

Примеры

Пример №1

Стандартный ввод
2173756 gwei 0xcabcb63c7dccc50c8bd715ebd6ddd9c3cc6055ab 0x20b094933ae12b443093bae4c948ab4976064fbb 0x90d0e1a5ca91a6c4c2390ce996e74c3d711c7f0e 0xf5a997a020b80cbbe4abacbe303628a19cc7df54 0xc9bbc295ec70274ebf96e886cea78998672fb04f 0x22d519accb88f4a30589b89baeb19d58419d682b 0x00f605655ab76cd3e195811234dd1b4b2b9b393b 0x54697fb47b8328de48a78e7b0136fb2b7a8fffcf 0x48f15f9bba01df85a2ab73891e603ae7c0f778ea 0x05b515ef047376ad74a6c00e8cf14b9c977b4f9d 0x159881171ae3cba81fecc23f724c0ba018cdfff7 0x11b52232b64ac629a451e8167c9a48299d299014 0x3ee3e6e5f01206eea239d74ba165719d940b84b4 0x24c29cfe4d95661e0178b340f1f0e60d6bccce89 0x4f59a86775e89f4d504d5393ab03ece01d583ee1 0x270aa72779584fd0b4f645c348b39af4a1492f2c 0x39847a89c6da9a625b3e4da9a1bae1544c3c93ee 0x1ad895f8fa2578590257b45216b0cffca912e2dc 0x4454bc5e7609bb4b2860711ae34530bb0a1f41a3 0x496f829701f68869d806e6fa6b458af132847c82

Стандартный вывод
6177000000

Решение

Так как для решения задачи не нужен аккаунт в тестовой сети, мы можем подключиться к сети Ropsten по удаленной ноде расположенной по адресу `https://ropsten.infura.io/`. Это позволит не скачивать всю историю сети, без чего нельзя было бы обойтись, так как для решения задачи необходим доступ к истории изменений балансов, которую не сможет обеспечить "ускоренная" синхронизация узла с блокчейн сетью.

Для решения воспользуемся функцией *getBalance*, находящуюся в модуле *eth*. С её помощью мы можем получить баланс для каждого аккаунта на момент выпуска данного блока. Найдя максимальный из них, можно получить ответ, который необходимо преобразовать в нужную единицу измерения.

В примере, данном ниже, входные данные считываются из файла и представлены в том виде, в котором даются во входных данных. Функция *get_max* находит максимальный баланс, а функция *convert* преобразовывает его в ответ.

Пример программы

Ниже представлено решение на языке Python3

```
1  def get_max(addr: list, block_n: int):
2      maxeth = -1
3      for i in addr:
```

```
        if eth.getBalance(i, block_n) > maxeth:
            maxeth = eth.getBalance(i, block_n)
    return maxeth

def convert(unit: str, bal):
    if unit == 'wei':
        return bal
    if unit == 'kwei' or unit == 'ada':
        return int(bal / 10**3)
    if unit == 'mwei' or unit == 'babbage':
        return int(bal / 10**6)
    if unit == 'gwei' or unit == 'shannon':
        return int(bal / 10**9)
    if unit == 'szabo':
        return int(bal / 10**12)
    if unit == 'finney':
        return int(bal / 10**15)
    return int(bal / 10**18)

from web3 import Web3

web3 = Web3(Web3.HTTPProvider('https://ropsten.infura.io/'))
eth = web3.eth

n = 20
addr = []
f = open('input.txt', 'r')
block_n = int(f.readline().strip())
unit = f.readline().strip()
for i in f.readlines():
    addr.append(web3.toChecksumAddress(i.strip()))

ans = convert(unit, get_max(addr, block_n))
print(ans)
```

Задача 6.3.17. Определение адреса контракта (2 балла)

Поскольку блокчейн с технологической точки зрения - это децентрализованная база данных, то как и большинство современных баз данных она позволяет хранить не только данные, но и исполняемый код.

Изначально подразумевалось, что исполняемый код в блокчейн должен регулировать денежные потоки, так как основная цель блокчейн, как базы данных, подразумевалась для хранения записей о перемещении условных единиц, эквивалентных деньгам. Например, код мог определять появление какой-то суммы на балансе счета, предназначенного для накопления полученной предприятием прибыли, после чего он автоматически мог бы делить данную прибыль между акционерами данного предприятия. Это было бы похоже на то, как исполняются обязательства по договорам (англ. *contracts*). Именно поэтому код, управляющий денежными потоками в блокчейн, согласились называть смарт-контрактами.

И блокчейн Bitcoin, и блокчейн Ethereum позволяют пользователями заносить в блокчейн свой собственный исполняемый код. Отличия только в том, где этот код в итоге храниться, и насколько универсальные алгоритмы могут быть реализованы с использованием тех команд, которые доступны для программирования.

В блокчейн Ethereum набор доступных команд является полным по Тьюрин-

гу (`https://ethclassic.ru/2016/10/21/turing-completeness-reality/`), что позволяет составлять довольно сложные смарт-контракты, в которых будут и циклы, и условные переходы, и процедуры (части программы, выделенные в отдельный блок из-за частого использования). Пользователь Ethereum может подготовить смарт-контракт и, с помощью специальной транзакции, поместить его в блокчейн. Процесс загрузки смарт-контракта называется деплойментом (*deployment*). По-русски его иногда называют регистрацией.

Транзакция для регистрации смарт-контракта отправляется без адреса получателя. В поле данных (*data*) у такой транзакции находится набор байт - скомпилированный в виде байт-кода смарт-контракт. При этом байт-код состоит из двух частей: *заголовка* и *байт-кода для хранения*:

- В заголовке хранится набор инструкций, которые выполнятся только один раз при регистрации контракта. Здесь может быть изначальная инициализация каких-то переменных, выполнение "конструктора" контракта.
- В байт-коде для хранения располагается тело самого смарт-контракта, которое содержит инструкции, исполняющиеся во время дальнейших обращений к этому смарт-контракту.

Дальнейшая судьба транзакции, регистрирующей смарт-контракт такова:

1. Транзакция включается в новый блок одним из майнеров сети Ethereum. При этом майнер
   (a) Инициализирует новое хранилище (*storage*), в котором будут храниться данные смарт-контракта (глобальные переменные, массивы, словари и т.п.). Данное хранилище выстраивается с использованием структуры данных Patricia Merkle Tree, поэтому оно называется *Storage tree*.
   (b) Выполняет все действия из заголовка байт-кода, что приводит к начальному заполнению *Storage tree*.
   (c) Формирует у в своей копии дерева состояния блокчейна (*World State Tree*) новую вершину, которая используется для хранения данных о смарт-контракте: *(balance, nonce, bytecode, storage tree root)*, где *balance* - сумма на счету контракта, по умолчанию - ноль, но может быть другая, если в траназкции, регистрируеющей контракт, выставлено значение *value*; *nonce* - количество дочерних смарт-контрактов, который данный контракт создал, если в ходе инициализации смарт-контракт создает другие контракты, то после регистрации здесь возможно значение отличное от нуля; *bytecode* - байт-код для хранения; *storage tree root* - хэш - корень *Storage Tree*, позволяющий контролировать синхронизацию изменений данных смарт-контракта на всех узлах в сети блокчейн.

   С вершиной *World State Tree* ассоциируется адрес, который называется адресом смарт-контракта. Поскольку, у смарт-контракта нет собственного приватного ключа, для назначения адреса смарт-контракта используется адрес аккунта, который отправляет транзакцию на регистрацию в сеть, а также номер этой транзакции относительно других транзакций, отправленных с этого адреса: $right(keccak256(rlp([account, nonce])), 20)$.

   Из описанного выше видно, что регистрация контракта изменяет дерево состояния блокчейн (*World State Tree*), следовательно измениться его *State*

Root, который будет отражен в новом блоке.

2. Блок распространяется по всем узлам сети.
3. При получении нового блока узел сети обрабатывает транзакцию на регистрацию смарт-контракта ровно таким же образом, как это делал майнер, включивший ее в блок.

   При этом в узле измениться сначала *Storage Root*, а соответственно и *State Root*. Вычисленный *State Root* должен совпадать с соответствующим полем в принятом блоке.

Таким образом, можно отметить, что несмотря на то, что код смарт-контракта хранится изначально в транзакции, как часть процесса майнинга блока он перемещается в *World State Tree* сначала только на одном узле, а потом на всех устройствах в сети блокчейн. Также выполнение инициализации (заголовка байт-кода) происходит на всех узлах сети блокчейн, что обеспечивает децентрализованное хранение не только кода, но и данных смарт-контракта.

Напишите программу, которая бы определяла бы адрес смарт-контракта по приватному ключу отправителя транзакции на регистрацию данного смарт-контракта.

Формат входных данных

Первая строка входного файла определяет приватный ключ, записанный в шестнадцатеричном формате. Длина строки - 64 символа.

Остальные строки в файле - текст смарт-контракта на языке Solidity, который необходимо сохранить в **приватной сети** блокчейн Ethereum.

Формат выходных данных

Одна строка - адрес смарт-контракта в блокчейн Ethereum.

Примеры

Пример №1

Стандартный ввод

```
227afa8360dacd29e170f0da77ea730e432aec597231d0e96d77360a126205b2
//Code of the smart contract starts below
contract CheckHashing {
    function check(uint8 n) view public returns(uint8) {
        bytes32 s = keccak256(msg.sender, block.number, n);

        return uint8(s[1]);
    }
}
```

Стандартный вывод

```
228343b58a53e4d0ec674e7aeb719f2920c8e291
```

Комментарии

Для решения задачи нужно использовать только что созданную приватную сеть блокчейн Ethereum. Т.е. не должно быть транзакций, отправленных с адреса, формируемого приватным ключом, передающимся в качестве входных данных.

Решение

Заметим, что для получения адреса контракта нам достаточно знать только адрес отправителя и количество транзакций, проведенных с этого адреса.

Так как по условию задачи с адреса, с которого отправляется смарт-контракт, на момент отправки не должно быть ни одной транзакции, можно заключить, что он находится в начальном состоянии(initial state). В момент создания аккаунта его начальное состояние устанавливается равным 1, а nonce вычисляется как разница между состоянием аккаунта и 1.

Таким образом, для того, чтобы узнать адрес смарт-контракта, нам необходимо использовать метод, описанный в условии задачи (*right(keccak256(rlp([account, nonce])), 20)*), где nonce устанавливается равным 0 (state - 1), а адрес равен адресу, полученному из приватного ключа.

Пример программы

Ниже представлено решение на языке Python3

```python
from sha3 import keccak_256
from py_ecc.secp256k1 import secp256k1
import rlp

def privToPub(priv):
    priv = int(priv, 16).to_bytes(32, 'big')
    res = secp256k1.privtopub(priv)
    x = res[0].to_bytes(32, 'big')
    y = res[1].to_bytes(32, 'big')
    return x+y

def pubToAddr(pub):
    return keccak_256(pub).hexdigest()[24:]

def privToAddr(priv):
    return pubToAddr(privToPub(priv))

with open('input.txt') as f:
    priv = f.readline().strip()

sender_addr = privToAddr(priv)
contract_addr = pubToAddr(rlp.encode([int(sender_addr, 16), 0]))
print(contract_addr)
```

Задача 6.3.18. Вызов метода контракта (3 балла)

Для исполнения кода смарт-контракта используется виртуальная машина Ethereum (*Ethereum Virtual Machine*, EVM). EVM является частью практически любого клиента сети Ethereum. Виртуальной ее называют, поскольку она реализована на уровне программного обеспечения, что позволяет абстрагироваться от аппаратного уровня

компьютера, а значит гарантирует, что на каком бы реальном железе ни запускался бы код смарт-контракта, результат выполнения будет один и тот же без необходимости перекомпилировать код (`https://en.wikipedia.org/wiki/Virtual_machine`).

Когда происходит запуск смарт-контракта на выполнение, в память EVM передается байт-код данного контракта, а также обеспечивается доступ к его хранилищу (*storage*). Определенные наборы байт в байт-кода интерпретируются в команды виртуальной машины и операнды к этим командам.

Пример части смарт-контракта, представленного в виде команд и операндов:

```
SHA3
SWAP4
POP
PUSH1 0x35
PUSH1 0x01
DUP7
PUSH2 0xffff
AND
MULMOD
```

Если посмотреть спецификацию Ethereum, то можно увидеть, что каждая команда EVM потребляет какое-то количество ресурсов, которые было принято измерять с использованием понятия "газ" (*gas*). Газ - это обобщенный подход к потреблению ресурсов, выражающий одновременно потребление процессорного времени и памяти. Концепция газа была введена, чтобы решить несколько проблем:

- Поскольку подразумевается, что смарт-контракты будут выполнять майнеры во время выпуска новых блоков, то несмотря на различные возможные аппаратные платформы у майнеров, необходимо ввести какой-то универсальный механизм измерения ресурсов, потраченных на выполнение смарт-контракта. Таким образом, заказчик выполнения смарт-контракта (отправитель транзакции) сможет построить свои ожидания относительно того, в каком объеме работа оборудования майнера должна быть компенсирована.
- Ограниченные вычислительные ресурсы майнера требуют возможности оценки сколько выполнение смарт-контракта может занять еще до того, как смарт-контракт будет запущен на выполнение. Благодаря тому, что отправитель транзакции указывает максимальное количество газа (*gas, сокр. от gasLimit*), которое он готов оплатить, и которое, в свою очередь, характеризует количество газа потребляемое смарт-контрактом, майнер может решать - возьмется он за выполнение данного смарт-контракта

Разные команды потребляют разное количество газа. Изначально, потребление газа в командах было ассоциировано с предположением того, каких ресурсов будет требовать от майнера выполнение этой команды - чем больше ресурсов для вычисления или хранения нужно, тем больше газа будет потребляться командой. Так, например, команды сохранения данных в хранилище смарт-контракта и команда создания нового смарт-контракта являются самыми тяжеловесными с этой точки зрения, поскольку требуют от майнера выделения дополнительной памяти, а, следовательно, дополнительных ресурсов на долговременное хранение этой памяти.

Запуск смарт-контракта, который уже зарегистрирован в блокчейн может выполняться несколькими способами:

- Если в ходе выполнения код смарт-контракта будет менять данные в *storage*, то тогда требуется отслеживание данных изменений в *World State* всеми узлами сети блокчейн. Это возможно только через посылку транзакции и последующим ее включением в новый блок.
- Если код контракта будет только что-то вычислять, базируясь на переданных ему параметрах или данных из *storage*, или, иными словами, не меняет *World State* то в этом случае можно использовать локальный вызов смарт-контракта. Данный способ не требует оплаты потребленного газа, но требует доступ к web3 модулю на узле Ethereum.

И в том и в другом случае, в итоге, произойдет загрузка целиком всего байт-кода контракта в виртуальную машину, после чего EVM начнет его исполнение. При этом важной деталью является то, что поле *data* транзакции, вызывающей код контракта, передается на вход коду, загруженному в EVM.

Большинство современных смарт-контрактов в блокчейн Ethereum написано на языке Solidity - высокоуровневом языке программирования. Байт-код для EVM получается в результате компиляции исходного кода на Solidity. Процесс компиляции автоматически подготавливает набор команд для выполнения EVM способный обработать входные данные, пришедшие через поле *data* транзакции. В частности, за счет этого набора команд реализуется возможность иметь в смарт-контракте несколько точек вызова - отдельных методов (функций) одного и того же контракта. Пошаговый анализ исполнения контракта виртуальной машиной показывает, что в первые команды кода по-факту определяют к какой из точек вызова должно перейти управление.

EVM ожидает, что описание точки вызова находится в первых четырех байтах поля *data* транзакции, отправленной для запуска смарт-контракта. Код смарт-контракта начинает сравнение этих четырех байт с теми, что у него прописаны. Если найдено совпадение, то происходит переход к соответствующей части кода:

Псевдокод, описывающий определение точки вызова в смарт-контракте

```
селектор = извлечь первые четыре байта поля data транзакции;
если (селектор == селектор_1) тогда
    переход к метка_1;
иначе если (селектор == селектор_2) тогда
    переход к метка_2;
. . .
иначе если (селектор == селектор_N) тогда
    переход к метка_N;
иначе
    вызвать исключение;
конец если;
метка_1:
    <<команды первого метода контракта>>;
метка_2:
    <<команды второго метода контракта>>;
    . . .
метка_N:
    <<команды N-го метода контракта>>;
```

Чаще всего, первые четыре байта поля *data* транзакции, вызывающей код смарт-

контракта, генерируются автоматически, на основании установленного стандарта. Для этого берется имя метода контракта (функции), который нужно вызвать, а также типы и последовательность аргументов, которые передаются в нанный метод. Эта информация представляется в виде строки, и от нее берется хэш $keccak256$. Например, для функции "get_balance", которая принимает Ethereum адрес и номер блока, преобразование будет выглядеть следующим образом:

$$desc = keccak256('get_balance(address, uint32)')[:4]$$

Имя функции и аргументов, берутся из исходного кода контракта. Но есть ситуации, когда не всегда целесообразно использовать код целиком или когда вообще нет возможности его использовать. Вместо этого используется описание интерфейсов смарт-контракта (*Application Binary Interface*, ABI), представляющий из себя JSON-структуру, где перечислены те сущности контракта на языке Solidity, к которым можно обращаться в конкретном смарт-контракте.

Пример ABI - описания интерфейса смарт-контракта

```
[
    {
        "constant": false,
        "inputs": [
            {
                "name": "n",
                "type": "uint16"
            }
        ],
        "name": "checkin",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    }
]
```

Большинство современных приложений, предоставляющих графический интерфейс пользователя для операций с блокчейн Ethereum умеют генерировать ABI из исходного кода смарт-контракта. А практически любая программная библиотека, работающая с Ethereum, позволяет с использованием известного ABI отправлять транзакции для вызова методов смарт-контракта.

Напишите программу, которая бы позволяла вызвать описанный выше метод *checkin* у смарт-контракта, расположенного по заданному адресу.

Формат входных данных

Первая строка входного файла определяет Ethereum адрес в **тестовой сети** Ropsten, по которому располагается смарт-контракт. Адрес контракта записан в шестнадцатеричном формате. Длина строки - 40 символов.

Вторая - целое число N ($1 \leqslant N \leqslant 198$), которое нужно будет передать в метод *checkin* смарт-контракта.

Формат выходных данных

Одна строка - количество газа, которое потратилось на обработку транзакции.

Примеры

Пример №1

Стандартный ввод
a623ea48a1a6f7257f24dfd59698773d5ca354c9 162

Стандартный вывод
79441

Комментарии

Для решения задачи нужно использовать тестовую сеть Ropsten блокчейн Ethereum.

Решение

Для решения задачи воспользуемся модулем `eth.contract` из библиотеки `web3`.

Функция *estimateGas* высчитывает и возвращает количество газа, которое потребовалось бы для вызова метода данного контракта. Таким образом, мы можем посчитать необходимое количество газа без необходимости создавать аккаунт в сети. Это дает возможность подключаться к тестовой сети Ropsten, не создавая узла на локальной машине.

ABI-описание интерфейса метода *checkin*, которое можно найти в условии к задаче, для удобства хранится в отдельном файле.

Все, что необходимо - это создать экземпляр контракта с данным ABI и вывести количество газа, которое потребуется на обработку транзакции.

Пример программы

Ниже представлено решение на языке Python3

```python
from web3 import Web3, HTTPProvider

web3 = Web3(Web3.HTTPProvider('https://ropsten.infura.io/'))

eth = web3.eth

# Считывание ABI метода 'checkin'
with open('scABI.txt', 'r') as f:
    scABI = f.read()

addr = input()
v = int(input())

scObj = web3.eth.contract(web3.toChecksumAddress(addr), abi=scABI)
print(scObj.functions.checkin(v).estimateGas())
```

Задача 6.3.19. Исполнение смарт-контрактов (4 балла)

Дан смарт-контракт:

```
pragma solidity ^0.4.17;

contract A {
    mapping (address => bool) checked;
    uint16 P = <some prime number>;

    function randomize(uint16 n) internal pure returns(uint8) {
        return uint8(mulmod(n, 1, 2));
    }

    function is_required(uint16 n) internal view returns(uint16) {
        return uint16(P - (n * addmod(n, randomize(n), P)));
    }

    function hash(uint256 message) internal pure returns (bytes32) {
        bytes memory prefix = "\x19Ethereum Signed Message:\n";
        return keccak256(prefix, '32', message);
    }

    function verify(uint16 n, uint8 v, bytes32 r, bytes32 s) public {
        require(is_required(n) == 0);

        bytes32 h = hash(uint256(n));

        address retAddr = ecrecover(h, v, r, s);

        assert(retAddr == msg.sender);

        checked[msg.sender] = true;
    }
}
```

Необходимо найти такие параметры для функции `verify`, при которых функция выполнится полностью. Т.е. словарь `checked` для отправителя транзакции запишется "true".

Содержимое смарт-контракта (поле контракта `P`) может изменяться для разных наборов входных данных.

Формат входных данных

Первая строка входного файла определяет приватный ключ, записанный в шестнадцатеричном формате. Длина строки - 64 символа.

В следующих строках идет код смарт-контракта, основанного на том, что представлен выше.

Формат выходных данных

В выходном файле должно быть три строки. В каждой из них содержатся r, s, v компоненты цифровой подписи, записанные в шестнадцатеричном виде, без префикса '0x'.

Первая строка длиной 64 символа - r. Вторая строка длиной 64 символа - s. Третья строка длиной 2 символа - v.

Примеры

Пример №1

Стандартный ввод

```
b10dbc23cce6ea8b29e791346c161c3c17fc24db7f4295eb8a96494d0f8fbfec
//Code of smart contract starts below
pragma solidity ^0.4.17;

contract A {
    mapping (address => bool) checked;
    uint16 P = 52837;

    function randomize(uint16 n) internal pure returns(uint8) {
        return uint8(mulmod(n, 1, 2));
    }

    function is_required(uint16 n) internal view returns(uint16) {
        return uint16(P - (n * addmod(n, randomize(n), P)));
    }

    function hash(uint256 message) internal pure returns (bytes32) {
        bytes memory prefix = "\x19Ethereum Signed Message:\n";
        return keccak256(prefix, '32', message);
    }

    function verify(uint16 n, uint8 v, bytes32 r, bytes32 s) public {
        require(is_required(n) == 0);

        bytes32 h = hash(uint256(n));

        address retAddr = ecrecover(h, v, r, s);

        assert(retAddr == msg.sender);

        checked[msg.sender] = true;
    }
}
```

Стандартный вывод

```
b215d1f3539a85437a194908c423c6a87d3f832bcf811dca9d41cf7b375af63e
71707da3071ac1b1af495d09bf1472545ce378565881a9d0f22cd3852864119c
27
```

Комментарии

Для решения задачи нужно использовать тестовую сеть Ropsten блокчейн Ethereum.

Решение

Для решения задачи воспользуемся модулями `keccak_256` из библиотеки `sha3` и `ecdsa_raw_sign` из библиотеки `py_ecc`.

Для того чтобы найти, какие значения должно выдавать наше решения, необходимо для начала понять, что же делает данный смарт-контракт и найти условия, при которых он полностью выполнится. Для этого попробуем разобраться с каждой функцией по отдельности.

1. Сначала разберемся с переменной *checked*. Согласно документации Solidity, mapping - это динамическая структура данных похожая на хэш-таблицу. То есть она ставит в соответствие некоторому ключу ячейку в памяти, где записано значение этого ключа. В нашем случае это соответствие булевой переменной для каждого адреса.

2. Переменная P имеет формат unsigned int размером в 16 бит. Она может может принимать значения в промежутке от 0 до 65535. В дальнейшем, когда в решении к этой задаче будет написано n-битное число, будет подразумеваться число, не имеющее знака, то есть всегда положительное, так как в данном контракте присутствуют только такие числа. Важно понимать, что будет происходить со значением переменной P при её переполнении, то есть если ей будет присвоено значение, выходящее за рамки данного диапазона. В этом случае лишние биты будут обрезаться. Реализовать это можно при помощи операции сравнения по модулю, в случае с 16-битным числом - по модулю 65536.

3. Функция *randomize* принимает на вход 16-битное число n, а возвращает 8-битное, полученное как остаток по модулю 2 от умножения n на 1. Легко заметить, что результат всегда будет равен либо 0, либо 1, в зависимости от того, чётное число подавалось на вход или нечётное. Поэтому при обрезании до 8 бит информация теряться не будет

4. Как мы увидим позднее, функция *is_required* является одной из самых главных для того, чтобы контракт полностью выполнился. Она принимает и возвращает 16-битное число n. Функция *addmod*, вызываемая внутри, возвращает остаток от деления суммы числа n и результата функции *randomize(n)* на число P. В большинстве случаев результат выражения в скобке будет больше числа P, поэтому велика вероятность, что при вычитании произойдет так называемый *underflow*, то есть, когда результат будет меньше минимально возможного. Однако и в этом случае нахождение остатка от деления по модулю будет давать правильный результат.

5. Для имплементации функции *hash* необходимо посмотреть как работает функция keccak256 на языке solidity. При её вызове для нескольких аргументов данные аргументы кодируются минимально необходимым количеством байт и конкатенируются. И уже к полученному набору байт применяется keccak256.

6. Главное назначение функции *verify* - проверка соответствия адреса отправителя значениям v, r и s, полученным при подписи документа.

   Функция *ecrecover* возвращает адрес пользователя, подписавшего документ. Данный адрес должен совпадать с адресом пользователя, обратившегося к контракту, иначе будет вызван метод *assert*, который откатит все изменения к предыдущему состоянию и израсходует весь предоставленный газ.

   Метод *recovery* похож на метод *assert* с той лишь разницей, что он не потребляет весь оставшийся газ, а просто откатывает состояние контракта и завершает выполнение функции.

   Таким образом, для выполнения вызванного метода значение n необходимо подобрать так, чтобы функция *is_required* от этой переменной равнялось нулю. А значения v, r и s должны быть подобраны так, чтобы адрес, полученный при помощи n, совпадал с адресом полученным из приватного ключа отправителя. Если все условия выполняются, то в соответствие данному ад-

ресу в переменной *checked* устанавливается значение true, и функцию *verify* можно считать полностью выполненной.

Для начала необходимо подобрать число n, чтобы выполнялся первый require. Для этого нужно перебрать все числа на промежутке от 0 до 65535 и выбрать то из них, которое даёт 0 при вызове функции *is_required*. На самом деле, возможны несколько таких значений, но для примера, данного в задании, подходит первое из них, соответственно будем брать также первое подходящее число n.

Значения v, r, s будем получать при помощи функции `ecdsa_raw_sign`, которая принимает на вход два набора байт: закодированные данные и приватный ключ пользователя, - а возвращает искомые значения в целых числах. Полученный ответ перед выводом необходимо перевести в шестнадцатеричный вид и привести к нужной длине.

Пример программы

Ниже представлено решение на языке Python3

```python
from sha3 import keccak_256
from py_ecc.secp256k1 import ecdsa_raw_sign

def randomize(n):
    return (n*1)%2

def is_required(n, P):
    Mod = 65536
    return (P - (n * (((n + randomize(n)) % P)))) % Mod

def zpad(s, l):
    return '0' * (l-len(s)) + s

P = int(input())
priv = input()

i = 0
while is_required(i, P) != 0:
    i+=1
n = i

prefix = "\x19Ethereum Signed Message:\n".encode()
number = '32'.encode()
message = n.to_bytes(32, byteorder='big')
hash = prefix+number+message
h = keccak_256(hash).digest()

priv = int(priv, 16).to_bytes(32, 'big')
ans = ecdsa_raw_sign(h, priv)

r = zpad(hex(ans[1])[2:], 64)
s = zpad(hex(ans[2])[2:], 64)
v = zpad(str(ans[0]), 2)

print(r, s, v, sep = '\n')
```

Задача 6.3.20. Получение баланса ERC-20 token (5 баллов)

Несмотря на относительно небольшую историю существования технологии смарт-контрактов Ethereum, на текущий момент уже наработано большое число шаблонных контрактов.

Одним из самых известных шаблонов является контракт ERC-20 (`https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md`), реализующий функционал, так называемого, токена. Использование измененного шаблона контракта «ERC-20 token» настолько распространено, что поэтому сложно дать точное определение слову токен.

Вот несколько примеров использования токена:

- токены, как внутренняя валюта. т.е. накапливая токены, я могу обменять их на товар, даже если товаром является валюта (подобным образом работают системы накопленных бонусов в магазинах);
- токены, как билеты. т.е. я могу обменять токен на какой-то сервис;
- токен, как акция. т.е. я могу получить часть прибыли при предъявлении токена;
- токен, как право участия. т.е. я могу принимать участие в решении каких-то вопросов, если я владею токеном.

В общем случае, получается, что токен - некая условная единица, которая может быть накоплена на счетах владельцев токенов. При этом должен предоставляться функционал получения информации о наличии и количестве токенов на определенном счету, а владельцы токенов имеют возможность передавать токены друг другу.

Стандарт «ERC-20 token» описывает несколько основных методов, которые должен предоставлять контракт, реализующий токен.

- *totalSupply* - возвращает общее количество токенов находящихся в обращении.
- *balanceOf* - возвращает количество токенов, находящихся во владении у определенного аккаунта.
- *transfer* - если транзакция с вызовом этого метода вызвана со определенного аккаунта, то данный аккаунт может сделать владельцем части своих токенов другой аккаунт. При этом сгенерируется событие *Transfer*, которое позволит отслеживать внешними сервисами списания и начисления токенов.

Один из алгоритмов обращения с токенами, используя методы, перечисленные выше, может выглядеть следующим образом:

1. Компания-владелец социальной сети в сети Интернет решает использовать токены для предоставления дополнительных услуг своим пользователям.
2. При инициализации токена на определенном выставляется, сколько токенов выпущено сразу после регистрации контракта.
3. Потенциальные пользователи токена перечисляют на счет смарт-контракта токена средства, *fallback* функция (см. документацию по Solidity) определяет количество перечисленных пользователем средств и ассоциирует с адресом пользователя определенное число токенов. Сумма полученных средств идет на разработку и поддержку функционала дополнительных услуг на сайте.

4. Позднее, пользователи могут передавать часть своих токенов сайту для получения доступа к дополнительным услугам.
5. Также пользователи могут передавать/дарить/обменивать токены друг другу. Таким образом, могут возникать экономические отношения с использованием токена, независящие от основной валюты блокчейна: ценность токена может изменяться в зависимости от качества и востребованности услуг сайта. Что при росте ценности позволит получать экономическую выгоду пользователям, купившим токен на ранних стадиях развития услуги.

Поскольку основными данными любого токена является информация о распределении токенов между его пользователями, практически любое современное программное обеспечение, выполняющее функции кошелька умеет получать эту информацию.

Подразумевая, что в сети Ropsten есть смарт-контракт, основанного на шаблоне *StandardToken* (`https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/StandardToken.sol`) таким образом, что никаких новых полей не было добавлено в контракт, напишите программу, которая бы получала количество токенов для заданных аккаунтов, сформированных после подтверждения блокчейн сетью определенного блока.

Формат входных данных

Первая строка входного файла определяет Ethereum адрес в **тестовой сети** Ropsten, по которому располагается смарт-контракт ERC-20 token. Адрес контракта записан в шестнадцатеричном формате. Длина строки - 40 символов.

Во второй строке целое число - номер блока, после обработки которого сформировались новое распределение токенов в смарт-контракте ERC-20 token.

Дальше идет 9 строк, в каждой из которых - Ethereum адрес владельца ERC-20 token, записанный в шестнадцатиричном виде. Длина адреса - 40 символов.

Формат выходных данных

Одно целое число – среднее медианное значение количества токенов среди адресов, указанных во входных данных. Необходимо работать с балансами сформировавшимися в Storage ERC-20 token после выпуска соответствующего блока.

Примеры

Пример №1

Стандартный ввод
63945c28ea8e15eabdf58d668e91c8518c710347 2218311 e09ba0ef60e3ec006071cc92caa1d43947fb25ff 4f59a86775e89f4d504d5393ab03ece01d583ee1 496f829701f68869d806e6fa6b458af132847c82 05b515ef047376ad74a6c00e8cf14b9c977b4f9d 59009daaefded4fd404ca7f0a484aac99cb2cfcd 84374ae93b30670fd657fc742923e10e35781efc 796a0c211aa2588f7bef4ecf94f05ccf3065ee91 90d0e1a5ca91a6c4c2390ce996e74c3d711c7f0e 5634ec3fb89c2667fdccf9a110800b62eb2ce249

Стандартный вывод
38272

Комментарии

Для решения задачи нужно использовать тестовую сеть Ropsten блокчейн Ethereum. Обратите внимание, что для решения задачи будет необходим доступ к истории изменений полей смарт-контракта, которую не сможет обеспечить "ускоренная" синхронизация узла с блокчейн сетью.

Для получения доступа к данным контракта, сформированным после применения определенного блока нужно использовать один из JSON-RPC методов, описанных на странице `https://github.com/ethereum/wiki/wiki/JSON-RPC`.

Решение

Для решения задачи воспользуемся методом `getStorageAt` модуля `web3.eth`. Входные данные будем считывать из файла "smart_contracts_4.txt".

В первом задании на смарт-контракты рассказано, как происходит регистрация смарт-контракта в сети блокчейн. Так, во время регистрации майнер инициализирует новое хранилище, в котором будут храниться данные контракта, или его состояние. Это состояние мы и можем считать при помощи метода *getStorageAt*, если знаем, в каком порядке расположены элементы в этом хранилище.

Все параметры в хранилище имеют размер 256 бит и индексируются с самого начала. Таким образом, мы можем узнать состояние первой объявленной в смарт-контракте переменной, обратившись к ней по индексу 0, второй переменной - по индексу 1 и т.д. Хочется заметить, что индекс должен быть представлен набором из 32 байт.

Однако, когда мы пытаемся прочитать состояние переменной, которая может хранить в себе несколько значений, например, mapping, возникает проблема: состояние всегда равно нулевому набору байт. Происходит это из-за того, что mapping имеет другую систему индексации, и для того, чтобы узнать его значение, необходимо знать значение ключа, состояние которого мы хотим прочитать. Это значение можно найти как результат применения *keccak_256* кодирования к сумме значения ключа и индекса (конкатенации двух наборов из 32 байт).

В смарт-контракте, основанном на шаблоне StandardToken первая переменная хранит в себе общее количество токенов, находящихся в обращении, а вторая - ко-

личество токенов для каждого адреса по отдельности. То есть необходимый индекс для поиска в хранилище равен '1'.

Таким образом, для решения задачи необходимо найти количество токенов для всех адресов, указанных во входных данных, и выбрать из них среднее медианное значение, которое в нашем случае равно количеству токенов для адреса, находящегося по этому количеству на 4 месте.

Пример программы

Ниже представлено решение на языке Python3

```python
from web3 import Web3
from ethereum.utils import sha3, encode_int32, zpad, encode_hex, decode_hex

def get_balance_at_contract(c, a, bn):
    object_num = 1
    object_bytes = encode_int32(object_num)
    addr_bytes = zpad(decode_hex(a[2:]), 32)
    maddr = sha3(addr_bytes + object_bytes).hex()
    return web3.toInt(web3.eth.getStorageAt(c, int(maddr, 16), bn))

web3 = Web3(Web3.HTTPProvider('https://ropsten.infura.io/'))
eth = web3.eth

f = open('smart_contracts_4.txt', 'r')

c_address = web3.toChecksumAddress(f.readline().strip())
bn = int(f.readline())
balance = []
for i in f.readlines():
    address = web3.toChecksumAddress(i.strip())
    balance.append(get_balance_at_contract(c_address, address, bn))

print(sorted(balance)[4])
```

Критерии определения призеров и победителей второго этапа

Количество баллов, набранных при решении всех задач суммируется. Призерам второго отборочного этапа было необходимо набрать 20 баллов (для участников из 9 класса) и 30 (для участников из 10-11 класса). Победители второго отборочного этапа должны были набрать 80 баллов и выше.

3. ФИНАЛЬНЫЙ ЭТАП

Заключительный этап олимпиады состоит из двух частей: индивидуальное решение задач по предметам (математика, информатика) и командное решение задачи на создание прототипа программного обеспечения.

Задачи индивидуального тура

На индивидуальное решение задач дается по 2 часа на один предмет. Для каждой из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, задачи по информатике - общие для всех участников.

Решение каждой задачи по математике дает определенное количество баллов (см. критерии оценки). При этом каждая задача делилась на несколько подзадач таким образом, что решение каждой подзадачи подводило к решению следующей. За каждую подзадачу можно получить от 0 до указанного количества баллов.

Решение задач по информатике предполагало написание программ. Ограничения по используемым языкам программирования не было. Проверочные тесты для каждой задачи по информатике делились на несколько групп. Прохождение всех тестов в первых двух группах для первой и второй задач дает количество баллов пропорциональное максимальному баллу за данную группу. То же самое применимо к третьей задаче. Последние группы тестов в первой и второй задачах дают определенное количество баллов только при условии успешного прохождения всех тестов.

Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 200 баллов:

- **Математика 9 класс** количество набранных баллов (от 0 до 45) умноженное на коэффициент 1,22;
- **Математика 10-11 класс** количество набранных баллов (от 0 до 50) умноженное на коэффициент 2;
- **Информатика** количество баллов от 0 до 100.

8.1. Задачи по математике (9 класс)

Задача 8.1.1. (20 баллов)

Числом Кармайкла будем называть всякое составное число n, которое удовлетворяет сравнению $b^n \equiv b \pmod{n}$ для всех целых b.

а) **(4 балла)** Докажите, что число n не является числом Кармайкла, если делится на квадрат простого числа, т.е. существует $p \in \mathbb{P}$, что $n \vdots p^2$.

б) **(8 баллов)** Докажите, что n является числом Кармайкла, если для каждого его простого делителя p выполнены следующие два условия:

(1) n не делится на p^2;

(2) $n-1$ делится на $p-1$.

в) **(8 баллов)** Пусть n — натуральное число. Положим $p = 6n+1$, $q = 12n+1$ и $r = 18n+1$. Докажите, что если p, q, r — простые числа, то произведение $p \cdot q \cdot r$ является числом Кармайкла.

Решение

а) Так как n — число Кармайкла, то противоречие получится, если мы найдем такое целое b, для которого $b^n \not\equiv b \pmod{n}$. Пусть $b = p$, тогда $p^n \equiv p \pmod{n} \Longrightarrow p^n \equiv p \pmod{p^2}$. Противоречие, так как $p^n \equiv 0 \pmod{p^2}$, но $p \not\equiv 0 \pmod{p^2}$.

б) Предположим, что p — простой делитель числа n, покажем тогда $b^n \equiv b \pmod{p}$. Если $b \vdots p$, то очевидно. Иначе, по малой теореме Ферма $b^{p-1} \equiv 1 \pmod{p}$. Пусть $n = (p-1) \cdot q + 1$, тогда

$$b^n \equiv (b^{p-1})^q \cdot b \equiv b \pmod{p}.$$

Так как $n = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, следовательно $b^n - b \vdots p_i$. Так как все p_i различны, то

$$b^n - b \vdots p_1 \cdot p_2 \cdot \ldots \cdot p_k = n.$$

Значит n является числом Кармайкла.

в) Докажем, что выполняются свойства пункта (б).

$$pqr - 1 = (18n+1)(12n+1)(6n+1) - 1 = 18n(12n+1)(6n+1) + 18n(4n+1) \vdots 18n.$$

$$pqr - 1 = (18n+1)(12n+1)(6n+1) - 1 = 12n(18n+1)(6n+1) + 12n(9n+2) \vdots 12n.$$

$$pqr - 1 = (18n+1)(12n+1)(6n+1) - 1 = 6n(18n+1)(12n+1) + 6n(36n+5) \vdots 6n.$$

Дополнительные критерии оценки

Отсутствуют.

Задача 8.1.2. (10 баллов)

На бесконечную шахматную доску со стороной клетки $2a$ наудачу брошена монета радиуса $r < a$ (центр монеты лежит на доске). Найти вероятность того, что:

а) **(5 баллов)** монета попадет целиком внутрь одной клетки;

б) **(5 баллов)** монета пересечет не более одной стороны одной клетки.

Решение

Посмотрим в какой области может лежать центр монеты внутри каждой клетки:

а) Для того, чтобы монета попадала целиком внутрь одной клетки, ее центр должен лежать внутри центрированного квадрата со стороной $2(a-r)$. Следовательно, вероятность для всей доски будет $P = \frac{64 \cdot 4(a-r)^2}{64 \cdot 4a^2} = \frac{(a-r)^2}{a^2}$.

б) Для того, чтобы монета пересекла не более одной стороны одной клетки, ее центр должен лежать в одном из четырех прямоугольников (по прямоугольнику $2(a-r) \times r$ возле каждой из сторон). Следовательно, вероятность для всей доски будет $P = \frac{64 \cdot 4 \cdot 2r(a-r)}{64 \cdot 4a^2} = \frac{2r(a-r)}{a^2}$.

Дополнительные критерии оценки

Отсутствуют.

Задача 8.1.3. (15 баллов)

На доске $n \times n$ одна из клеток закрашена невидимыми чернилами. Разрешается выделить любой прямоугольник и спросить, есть ли в нем выделенная клетка. За какое минимальное число вопросов можно найти выделенную клетку, если

а) **(7 баллов)** $n = 4$;

б) **(8 баллов)** $n = 5$.

Решение

Закодируем каждую клетку двоичным кодом. Задавая вопрос про любой прямоугольник мы получаем не больше одного бита (получаем ответ да или нет).

а) Чтобы закодировать 16 клеток, нам нужно хотя бы 4 бита. Следовательно, быстрее чем за 4 вопроса мы не можем найти закрашенную клетку. (пример легко строится: за два вопроса узнаем в каком из четырех квадратов 2×2 лежит закрашенная клетка; за остальные 2 вопроса узнаем уже точное расположение клетки в квадрате 2×2).

б) Чтобы закодировать 25 клеток, нам нужно хотя бы 5 битов. Следовательно, быстрее чем за 5 вопросов мы не можем найти закрашенную клетку. (пример легко строится: выделяем 4×4; если клетка в нем, то по предыдущему пункту находим за 4 вопроса; иначе в оставшейся области выделяем 1×4 и 1×5 за один вопрос определяем в какой части лежит клетка и за оставшиеся 3 вопроса находим точное расположение клетки).

Дополнительные критерии оценки

Отсутствуют.

8.2. Задачи по математике (10-11 класс)

Задача 8.2.1. (18 баллов)

Числом Кармайкла будем называть всякое составное число n, которое удовлетворяет сравнению $b^n \equiv b \pmod{n}$ для всех целых b.

а) **(5 баллов)** Докажите, что n является числом Кармайкла, если для каждого его простого делителя p выполнены следующие два условия:

(1) n не делится на p^2;

(2) $n-1$ делится на $p-1$.

б) **(6 баллов)** Докажите, что не существует чисел Кармайкла вида $n = p \cdot q$.

в) **(7 баллов)** Найдите все числа Кармайкла вида $n = 3 \cdot p \cdot q$, где p и q — простые числа.

Решение

а) Предположим, что p — простой делитель числа n, покажем тогда $b^n \equiv b \pmod{p}$. Если $b \vdots p$, то очевидно. Иначе, по малой теореме Ферма $b^{p-1} \equiv 1 \pmod{p}$. Пусть $n = (p-1) \cdot q + 1$, тогда

$$b^n \equiv (b^{p-1})^q \cdot b \equiv b \pmod{p}.$$

Так как $n = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, следовательно $b^n - b \vdots p_i$. Так как все p_i различны, то

$$b^n - b \vdots p_1 \cdot p_2 \cdot \ldots \cdot p_k = n.$$

Значит n является числом Кармайкла.

б) Для того чтобы $n = p \cdot q$ было числом Кармайкла необходимо и достаточно, чтобы $p \neq q$, $(p \cdot q - 1) \vdots (p-1)$ и $(p \cdot q - 1) \vdots (q-1)$. Получаем

$$(p-1) \cdot q + (q-1) \vdots (p-1) \Longrightarrow (q-1) \vdots (p-1).$$

Аналогично

$$p \cdot (q-1) + (p-1) \vdots (q-1) \Longrightarrow (p-1) \vdots (q-1).$$

Следовательно, $(p-1) = (q-1)$. Противоречие.

в) Для того чтобы $n = 3 \cdot p \cdot q$ было числом Кармайкла необходимо и достаточно, чтобы $p \neq q \neq 3$, $(3 \cdot p \cdot q - 1) \vdots (p-1)$, $(3 \cdot p \cdot q - 1) \vdots (q-1)$ и $(3 \cdot p \cdot q - 1) \vdots (3-1)$. Из последней делимости следует, что p, q – нечетные простые числа. Будем считать, что $p > q > 3$. Получаем

$$3(p-1)q + (3q-1) \vdots (p-1) \Longrightarrow (3q-1) \vdots (p-1).$$

Так как $p > q$, то возможны только два варианта:

1) $3q - 1 = p - 1 \Longrightarrow 3q = p$. Противоречие.

2) $3q - 1 = 2(p - 1) \Longrightarrow 3q + 1 = 2p$. Аналогично можно получить

$$(3p - 1) \vdots (q - 1) \Longrightarrow 2(3p - 1) \vdots (q - 1) \Longrightarrow (9q + 1) \vdots (q - 1) \Longrightarrow 10q \vdots (q - 1)$$

А так как $(q, q - 1) = 1$, то $10 \vdots q - 1$. Получаем единственный возможный вариант $q = 11$, $p = 17$ и $n = 561$.

Дополнительные критерии оценки

Отсутствуют.

Задача 8.2.2. (14 баллов)

Через специальный кабель Алиса передает случайную последовательность битов $x_1x_2 \cdots x_n$ своему однокласснику Бобу, который на выходе получает последовательность $y_1y_2 \cdots y_n$. Боб получает искаженное сообщение в следствии того, что кабель неидеальный (имеется вероятностный шум):

(1) бит 0 с вероятностью ε изменяется на 1;

(2) бит 1 с вероятностью ε изменяется на 0.

Обозначим через α — вероятностное распределение, передаваемых Алисой символов (будем считать, что $P(\text{бит } a_i \text{ равен } 0) = p$ и $P(\text{бит } a_i \text{ равен } 1) = 1 - p$).

а) **(4 балла)** Выразите через p и ε вероятностное распределение β, получаемых Бобом битов;

б) **(4 балла)** Вычислите величину $Q = \max\limits_{\alpha} H(\beta)$, где H — энтропия Шеннона;

в) **(6 баллов)** Вычислите пропускную способность кабеля $R = \max\limits_{\alpha} I(\alpha : \beta)$ и выясните при каких ε она максимальна и минимальна (I – взаимная информация).

Решение

а)

$$P(b_i = 0) = P(a_i = 0) \cdot (1 - \varepsilon) + P(a_i = 1) \cdot \varepsilon = p \cdot (1 - \varepsilon) + (1 - p) \cdot \varepsilon.$$

$$P(b_i = 1) = P(a_i = 1) \cdot (1 - \varepsilon) + P(a_i = 0) \cdot \varepsilon = (1 - p) \cdot (1 - \varepsilon) + p \cdot \varepsilon.$$

б)

$$Q = \max_{\alpha} H(\beta) = \max_{p} H(\beta)$$

При $p = 1$ получаем $P(b_i = 0) = P(b_i = 1) = 0.5$, следовательно при таком p энтропия будет максимальна $H(\beta) = \log_2 2 = 1$. Значит $Q = 1$.

в)

$$R = \max_{\alpha} I(\alpha : \beta) = \max_{\alpha}(H(\beta) - H(\beta|\alpha)) = \max_{\alpha}(H(\beta) - H(\beta|a_i = 0)P(a_i = 0) -$$

$$-H(\beta|a_i = 1)P(a_i = 1)) = \max_{\alpha} H(\beta) - \varepsilon \log_2 \frac{1}{\varepsilon} - (\varepsilon - 1)\log_2 \frac{1}{(\varepsilon - 1)} = 1 - h(\varepsilon).$$

Предпоследнее равенство верно так как

$$H(\beta|a_i = 0) = H(\beta|a_i = 1) = \varepsilon \log_2 \frac{1}{\varepsilon} - (\varepsilon - 1)\log_2 \frac{1}{(\varepsilon - 1)} = h(\varepsilon).$$

Функция $h(\varepsilon)$ принимает минимальное значение при $\varepsilon = 0$ или 1 и максимальное значение при $\varepsilon = 0.5$.

Дополнительные критерии оценки

Отсутствуют.

Задача 8.2.3. (18 баллов)

Пусть p и q — различные простые числа и $n = p \cdot q$. Блок b будем называть неподвижным относительно RSA-системы с открытым ключом (n, e), если он не изменяется после кодирования, т.е. $E(b) = b$. Определите число неподвижных блоков относительно RSA-системы в случае:

а) **(8 баллов)** $e = 3$, $p = 3$ и $q > 3$;

б) **(10 баллов)** $e = 5$, $p > 5$ и $q > 5$.

Решение

Так как b неподвижный блок относительно RSA-системы с открытым ключом (n, e), то $b^e \equiv b \pmod{n}$. Следовательно, $b^e \equiv b \pmod{p}$ и $b^e \equiv b \pmod{q}$. Решим каждое из этих сравнений и применим китайскую теорему об остатках. Получим, что общее число неподвижных блоков это произведение числа решений $b^e \equiv b \pmod{p}$ и $b^e \equiv b \pmod{q}$.

а) $b^3 \equiv b \pmod{3}$ имеет ровно 3 решения (любой остаток по модулю 3 подходит); $b^3 \equiv b \pmod{q}$ имеет также ровно 3 решения ($b(b-1)(b+1) \vdots q \Longrightarrow b \equiv 0, \pm 1 \pmod{q}$) В итоге получаем 9 неподвижных блоков.

б) $b^5 \equiv b \pmod{p} \Longrightarrow b(b-1)(b+1)(b^2+1) \vdots p$. Получаем 3 решения точно имеются $b \equiv 0, \pm 1 \pmod{p}$. Остальные два зависят от того является ли -1 квадратичным вычетом по модулю p. Если $p = 4k + 1$, то -1 — квадратичный вычет $\Longrightarrow$ решений будет 5, а если $p = 4k + 3$, то -1 — квадратичный невычет $\Longrightarrow$ решений будет 3. Аналогичное утверждение можно сделать для делителя q.

В итоге, в зависимости от того какие остатки дают p и q при делении на 4, получаем 9, 15 или 25 неподвижных блоков.

Дополнительные критерии оценки

Отсутствуют.

8.3. Задачи по информатике

Задача 8.3.1. Платежные каналы (25 баллов)

В технологии блокчейн Probocoin для передачи средств используются односторонние платежные каналы, каждый из которых связывает два узла сети ProboNetwork.

Средства из узла a в узел b в этой сети передаются либо через прямой платежный канал, либо через цепочку платежных каналов.

Используя данные о существующих платежных каналах, ответьте на следующий вопрос:

Есть ли возможность передать средства от человека X человеку Y, не создавая новые каналы?

Человек X представляет собой узел s, человек Y представляет собой узел t.

Формат входных данных

В первой строке вводятся два целых числа n, m ($1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 2 \cdot 10^5$) — количество узлов и каналов в системе.

В следующих m строках вводятся по два целых числа a_i и b_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$) — начало и конец i-го канала.

В последней строке вводятся два целых числа s и t ($1 \leqslant s, t \leqslant n, s \neq t$).

Формат выходных данных

Выведите *Yes*, если можно передать средства от человека X человеку Y, не создавая новые каналы. В противном случае выведите *No*.

Примеры

Пример №1

Стандартный ввод
3 1 1 3 1 3

Стандартный вывод
Yes

Пример №2

Стандартный ввод
3 1 3 1 1 3

Стандартный вывод
No

Способ оценки работы

За решение задачи начислялось:

- **5 баллов**, если пройдена только первая группа тестов;
- **10 баллов**, если пройдена первая и вторая группы тестов;
- **10 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```python
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()
```

Решение

Первая задача сводится к тому, что у нас есть ориентированный граф состоящий из n вершин и m ребер и нам нужно проверить есть ли путь из вершины s в вершину t.

Это стандартная задача и решается она с помощью алгоритма DFS или алгоритма BFS.

Время работы: $O(n + m)$

Память: $O(n + m)$

Пример программы

Ниже представлено решение на языке C++

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;
#define pb push_back
#define sz(x) ((int)x.size())
#define fi first
#define se second


const int MAXN = 100000+10;


vector<int> g[MAXN];
char used[MAXN];
void dfs(int v){
    used[v]= 1;
    for(int to:g[v]){
        if(!used[to]){
            dfs(to);
        }
    }
}

```

```
void solve(){
    int n,m;
    cin >> n >> m;
    for(int i=1;i<=m;++i){
        int a,b;
        cin >> a >> b;
        g[a].pb(b);
    }
    int s,t;
    cin >> s >> t;
    dfs(s);
    if(used[t]){
        cout <<"Yes"<<endl;
    } else{
        cout <<"No"<<endl;
    }

}


int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    solve();

}
```

Задача 8.3.2. Комиссия (40 баллов)

В сети ProboNetwork каждый платежный канал имеет свою комиссию, которая взимается при передаче средств по каналу. Эта комиссия не зависит от переводимой по платежному каналу суммы и одинакова для всех переводов средств по платежному каналу.

Также в сети ProboNetwork можно добавлять новые платежные каналы. За добавление нового канала потребуется заплатить q_0. Но добавлять можно не все каналы, а только те, которые разрешены системой. Всего существует q таких каналов.

Используя данные о существующих платежных каналах и списке каналов, которые разрешены к добавлению системой, ответьте на следующий вопрос:

Какова минимальная сумма, которую придется заплатить, чтобы произвести передачу денег из узла s в узел t.

Формат входных данных

В первой строке вводятся два целых числа n, m ($1 \leqslant n \leqslant 10^5$, $0 \leqslant m \leqslant 10^5$) — количество узлов и каналов в системе.

В следующих m строках вводятся по три целых числа a_i, b_i, c_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$, $1 \leqslant c_i \leqslant 10^4$) — начало, конец и комиссия i-го канала.

В следующей строке вводятся два целых числа s и t ($1 \leqslant s, t \leqslant n$, $s \neq t$).

В следующей строке вводятся два целых числа q и q_0 ($1 \leqslant q \leqslant 2 \cdot 10^5$, $0 \leqslant q_0 \leqslant 1000$).

В следующих q строках вводятся по три целых числа a_i, b_i, c_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$, $1 \leqslant c_i \leqslant 10^4$) — начало, конец и комиссия i-го канала в списке.

Формат выходных данных

Выведите ответ на вопрос или -1, если передать средства из узла s в узел t невозможно.

Примеры

Пример №1

Стандартный ввод
3 1 1 3 5 1 3 2 6 1 2 1 2 3 1

Стандартный вывод
5

Пример №2

Стандартный ввод
3 1 1 2 5 1 3 1 1 1 3 2

Стандартный вывод
3

Пример №3

Стандартный ввод
3 1 1 2 1 3 1 1 1 2 3 1

Стандартный вывод
-1

Способ оценки работы

За решение задачи начислялось:

- **5 баллов**, если пройдена только первая группа тестов;

- **15 баллов**, если пройдена первая и вторая группы тестов;
- **20 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```python
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()
```

Решение

Можно заметить, что ребра, которые можно добавить в граф за стоимость q_0, можно представить в виде ориентированных ребер в графе со стоимостью $q_0 + c_i$.

Таким образом, у нас есть ориентированный взвешенный граф состоящий из n вершин и $m + q$ ребер. И нам нужно найти длину кратчайшего пути из вершины s в вершину t или сказать, что пути из s в t не существует.

Это стандартная задача и решается она с помощью алгоритма Дейкстры для разреженных графов.

Для решения первых двух подзадач можно было воспользоваться обычным алгоритмом Дейкстры, алгоритмом Форда-Беллмана или алгоритмом Левита.

Время работы: $O((m + q) \cdot \log_2(n))$

Память: $O(n + m + q)$

Пример программы

Ниже представлено решение на языке C++

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
#define pb push_back
#define sz(x) ((int)x.size())
#define fi first
#define se second
const ll inf_ll = 2e18;

const int MAXN = 100000+10;


vector<pii> g[MAXN];
ll dis[MAXN];
void dey(int st){
    fill(dis,dis+MAXN,inf_ll);
    dis[st] = 0;
    set<pll> s;

    s.insert({0,st});
    while(!s.empty()){
```

```
24          int v = s.begin()->se;
25          s.erase(s.begin());
26          for(pii x:g[v]){
27              int to = x.fi;
28              int len = x.se;
29              if(dis[to]>dis[v]+len){
30                  s.erase({dis[to],to});
31                  dis[to]= dis[v]+len;
32                  s.insert({dis[to],to});
33              }
34          }
35      }
36  }
37
38
39  void solve(){
40  int n,m;
41      cin >> n >> m;
42      for(int i=1;i<=m;++i){
43          int a,b,c;
44          cin >> a >> b >> c;
45          g[a].pb({b,c});
46      }
47      int s,t;
48      cin >> s >> t;
49      int q,q0;
50      cin >> q >> q0;
51      for(int i=1;i<=q;++i){
52          int a,b,c;
53          cin >> a >> b >> c;
54          g[a].pb({b,c+q0});
55      }
56      dey(s);
57      if(dis[t]==inf_ll){
58          cout << -1<<endl;
59      } else{
60          cout <<dis[t]<<endl;
61      }
62
63  }
64
65  int main(){
66      ios_base::sync_with_stdio(0);
67      cin.tie(0);
68      cout.tie(0);
69      solve();
70
71  }
```

Задача 8.3.3. Дерево блоков (35 баллов)

Программист Иван создал свою версию *blockchain* технологии. Ему показалось странным, что для выстраивания цепочки блоков большинство используют линейный список, хотя по его мнению использование дерева позволило бы добиться большей производительности *blockchain* сети.

В его версии *blockchain* в каждый определенный момент времени цепочка блоков состоит из n блоков, которые образуют дерево. Каждому ребру в дереве присваивают

случайное число val_i. Для верификации блоков используют XOR всех значений на пути между текущей вершиной и другой случайной вершиной.

Система Ивана начала работать, но Иван стал замечать что, некоторых значения стали слишком часто совпадать. И теперь он боится, что совершил ошибку в коде.

Для этого он решил проверить задачу на специальных запросах: он выбирает вершину v_i и значение q_i и хочет узнать какое количество вершин u_i, таких что XOR всех значений на пути от v_i к u_i равен q_i .

Формат входных данных

В первой строке вводятся два целых числа n, q ($2 \leqslant n \leqslant 10^5$, $1 \leqslant q \leqslant 10^5$) — количество блоков и запросов.

В следующих $n-1$ строках вводятся по три целых числа a_i, b_i, val_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$, $0 \leqslant val_i \leqslant 10^9$), которые означают, что между вершинами a_i и b_i в дереве есть ребро и ему присвоено число val_i.

Гарантируется, что ребра образуют дерево.

В следующих q строках вводятся по два целых числа v_i, q_i ($1 \leqslant v_i \leqslant n$, $0 \leqslant q_i \leqslant 10^9$).

Формат выходных данных

Выведите q чисел, i-е из которых - это количество вершин u_i, таких что XOR всех значений на пути от v_i к u_i равен q_i.

Примеры

Пример №1

Стандартный ввод

```
4 3
1 2 1
2 3 2
2 4 4
1 1
1 3
1 7
```

Стандартный вывод

```
1
1
0
```

Способ оценки работы

За решение задачи начислялось:

- **35 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()
```

Решение

Наивное решение:

Будем запускать DFS каждый раз во время запроса, и считать количество вершин во время обхода поддерживая текущее число xor.

Время работы: $O(q \cdot n)$

Память: $O(n + q)$

Полное решение:

Давайте выберем одну вершину корнем нашего дерева (например, 1). Теперь запустим DFS из корня и предподсчитаем значение $pref[v]$.

$pref[v]$ - xor всех ребер на пути, от корня до вершины v.

Для решения задачи нужно воспользоваться следующим свойством операции XOR:

$a\ xor\ a = 0$

Заметим, что если мы возьмем две вершины v_1 и v_2, то xor ребер на пути от v_1 до v_2 будет равен $pref[v_1]\ xor\ pref[v_2]$. Потому что ребра, которые не принадлежат пути, попадут два раза и не будут учитываться.

Заметим, что $pref[v_1]\ xor\ pref[v_2] = q_i \Leftrightarrow pref[v_2] = q_i\ xor\ pref[v_1]$

Следовательно, для каждого запроса нам нужно найти количество вершин, у которые значение $pref$ равно $q_i\ xor\ pref[v_1]$.

Это можно решить с помощью обычного словаря.

Время работы: $O((n + q) \cdot \log_2(n))$

Память: $O(n + q)$

Пример программы

Ниже представлено решение на языке C++

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  typedef pair<int,int> pii;
5  typedef pair<ll,ll> pll;
6  #define pb push_back
7  #define sz(x) ((int)x.size())
8  #define fi first
9  #define se second
10 const ll inf_ll = 2e18;
11 const int MAXN = 1e5+1;
12
```

```

vector<pii> g[MAXN];
int val[MAXN];

void dfs(int v,int p){
    for(pii x:g[v]){
        int to = x.fi;
        if(to==p)
            continue;
        int len = x.se;
        val[to] = val[v]^len;
        dfs(to,v);
    }
}

void solve(){
    int n,q;
    cin >> n >> q;
    for(int i=1;i<=n-1;++i){
        int a,b, c;
        cin >> a >> b >> c;
        g[a].pb({b,c});
        g[b].pb({a,c});
    }
    map<int,int> mp;
    dfs(1,-1);

    for(int i=1;i<=n;++i){
        mp[val[i]]++;
    }

    for(int i=1;i<=q;++i){
        int v1,q1;
        cin >> v1>> q1;
        int res = val[v1]^q1;

        cout << mp[res]<<"\n";
    }

}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    solve();

}
```

Задача командного тура

В командной части заключительного этапа участники должны разработать децентрализованное приложение на базе платформы Ethereum, для взаимодействия автономных электромобилей и автоматических станций сервисного обслуживания, предоставляющих услуги по замене разряженных тяговых аккумуляторных батарей. Приложение включает в себя логику проверки батарей на контрафакт, а также экономическую систему мотивации участия и защиты от мошенничества.

Командная часть заключительного этапа имеет продолжительность 3,5 дня (всего 24 астрономических часа), которые включают работу по настройку среды разработки на базе узла сети Ethereum, разработку частей приложения на языке Python и Solidity, создание интеграционных тестов, а также приемка решения в системе автоматической проверки.

9.1. Легенда

Наступило ближайшее будущее автобаны и дороги в городах наводнены электромобилями! Свободный рынок позволяет каждому производителю автомобилей, существующему еще с 20 века, выпускать широкий модельный ряд устройств, сильно отличающихся от своих предшественников на органическом топливе. Огромное количество крупных и мелких компаний снабжают автозаводы всеми необходимыми комплектующими начиная от дворников и брызговиков, заканчивая электромоторами и тяговыми аккумуляторными батареями.

На тяговом аккумуляторе автомобиль может проехать относительно небольшое расстояние — в среднем, 400-500 километров. Затем, ему нужно перемещаться к зарядной станции или центру сервисного обслуживания.

На зарядной станции можно не спеша зарядить батарею за 45-50 минут. На автоматизированном центре сервисного обслуживания аккумуляторное устройство могут быстро заменить за 3-4 минуты. Естественно, центры обслуживания пользуются заслуженной популярностью.

Однако, при большом количестве производителей аккумуляторных батарей, огромном количестве электромобилей и сотнях различных компаний, держащий центры сервисного обслуживания возникает проблема доверия: действительно ли в обмене между транспортным средством и центром обслуживания участвуют только качественные батареи, ведь получив батарею с израсходованным ресурсом зарядов автомобиль рискует застрять где-то в пути, а станция сервисного обслуживания будет терпеть убытки, связанные с необходимостью обновления своих складских запасов.

Поэтому необходимо написать программное обеспечение для использования участ-

никами обмена, которое бы позволяло гарантированно проверять качество аккумуляторных батарей, автоматизировало бы процесс ведения сделки по оказанию услуг сервисным центром, а также обеспечивало вы финансовое сопровождение всех необходимых операций без привлечения человека: ведь не только станция может быть автоматической, но и электромобиль может оказаться беспилотным.

Чтобы исключить возможность злонамеренных действий со стороны каждой из сторон участвующей в системе экономических взаимоотношений: компании производителя ПО, производителя батарей, электромобиля и центра сервисного обслуживания, программное обеспечение должно быть разработано на базе платформы распределенного реестра. Это позволит решению обладать следующими преимуществами:

- Нет единой точки компрометации - код децентрализованного приложения будет выполняться на узлах сети блокчейн - валидаторах сети, а изменения в данных, принятые в ходе исполнения кода, будут многократно перепроверяться всеми другими узлами.
- Нет единой точки хранения данных - код приложения и данные хранятся одновременно на всех узлах сети блокчейн, контролируемые криптографической связностью данных. Это, вместе с предыдущим пунктом, позволяет гарантировать, что никакой из участников системы не сможет поменять свои договоренности "задним числом" или, подменив код приложения, заставить взаимодействовать участников по новым, неизвестным для них правилам.
- Нет единой точки входа, что позволяет участникам избегать атак, построенных по принципу man-in-middle.
- опирающаяся на безопасность и простоту сети блокчейн экономическая модель будет поддерживать мотивацию участников системы к честному поведению и продолжению взаимодействия.

Участники взаимодействия

При разработке программного обеспечения должно учитываться взаимодействие следующих сущностей:

- **производитель программного обеспечения** - выполняет развертывание системы: распространение скриптов между участниками системы, регистрацию смарт-контрактов, а также настройку систему. Он является выгодоприобретателем при подключении новых участников системы. Производитель ПО отвечает за введение в оборот расчетных токенов, обеспечение механизмов распределения расчетных токенов между участниками (не рассматривается в данной задаче).

  Все действия, происходящие от имени производителя ПО, выполняются через скрипт `setup.py`.
- **производитель тяговых аккумуляторных батарей** - оплачивает свою регистрацию в системе. Выпускает новые батареи и регистрирует их в системе. Первые 1000 батарей регистрируются за счет первоначальной оплаты, за регистрацию каждой следующей батареи производитель должен вносить дополнительную сумму. Сумма оплаченная за регистрацию производителя батарей и за регистрацию самих батарей перечисляется производителю программного обеспечения. После регистрации аккумуляторные батареи могут

быть распределены производителем батарей между сервисными центрами и электромобилями.

Все действия, происходящие от имени производителя батарей, выполняются через скрипт `vendor.py`.

- **тяговые аккумуляторные батарей** - объект реального мира, состоящий из элементов, аккумулирующих электрический заряд, электронной управляющей системы с постоянной памятью. В рамках данной задачи считается, что нельзя изменить физико-химические характеристики батареи без выведения из строя электронной управляющей системы. В самом простом случае управляющая система включает в себя подсистему подсчитывающую количество циклов заряда аккумуляторной батареи, аппаратного ключа защиты, которые хранит приватный ключ батареи, он генерируется при создании батареи и уникальный для каждого устройства, и содержит код для создания цифровой подписи по запросу от управляющей системы. Также батарея имеет специальный цифровой порт, через который можно получить так называемый статус батареи: количество циклов заряда, временную метку и соответствующие им цифровую подпись. За счет того, что временная метка изменяется от одного считывания к другому, цифровая подпись будет все время отличаться даже при одинаковом значении циклов заряда.

  В данном решении электронная управляющая система батареи будет эмулироваться скриптом на языке `Python`.

- **сервисный центр по замене тяговых аккумуляторных батарей** - регистрируется в системе (регистрация бесплатная), после чего получает право отвечать на запросы электромобилей для замены батарей. Подключается к батарее, установленной в электромобиле для получения информации о статусе батареи, находит подходящую заряженную тяговую батарею, оформляет контракт на сделку по замене батарей. После замены батарей, сервисный центр получает от электромобиля компенсацию за свою работу в виде определенного количества расчетных токенов. Изначально получает батареи от производителя батарей.

  Все действия, происходящие от имени сервисного центра, выполняются через скрипт `scenter.py`.

- **электромобиль** - регистрируется в системе (регистрация бесплатная), после чего получает право быть участником в сделках по замене батарей. Получает информацию о контракте по замене батарей от сервисного центра. После ознакомления с условиями контракта соглашается с ними, переводя расчетные токены на счет контракта. После замены батарей, проверяет статус вновь установленной батареи и завершает контракт, что инициирует перевод расчетных токенов на счет сервисного центра.

  Все действия, происходящие от имени электромобиля, выполняются через скрипт `car.py`.

9.2. Набор заданий

Решение командной задачи было разбито на подзадачи, сгруппированные в 3 набора.

Каждая подзадача (*user story*) формулировала необходимый функционал, ко-

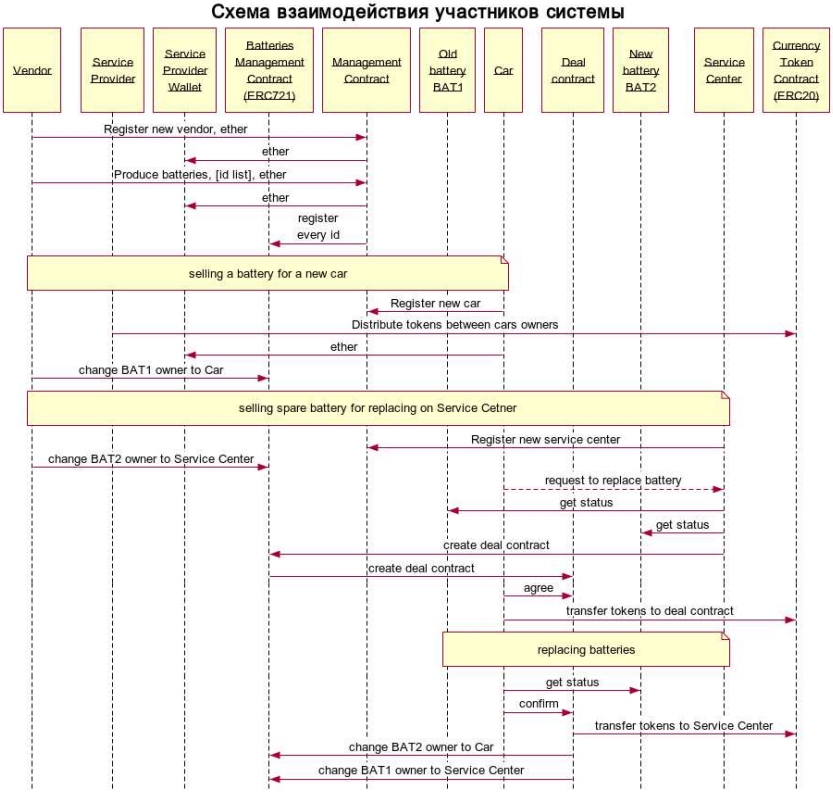


Рис. 9.1: Диаграмма взаимодействия участников системы

торый должен был быть реализован командой, а также набор приемочных тестов (*acceptance criteria*), позволявших проверить в полном ли объеме решена данная подзадача.

Каждый набор подзадач предлагалось решать в отдельном этапе (*итерации*). Первые два этапа итеративно подводили участников к решению полной финальной задачи, осуществляемому во время последнего третьего этапа. На каждом этапе решение подзадач проверялось с помощью системы автоматической системы оценивания, которая запускала решение участников с теми или иными параметрами в соответствии с приемочными тестами.

Решение командной задачи было разбито на подзадачи, сгруппированные в 3 набора.

Каждая подзадача (*user story*) формулировала необходимый функционал, который должен был быть реализован командой, а также набор приемочных тестов (*acceptance criteria*), позволявших проверить в полном ли объеме решена данная подзадача.

Каждый набор подзадач предлагалось решать в отдельном этапе (*итерации*).

Первые два этапа итеративно подводили участников к решению полной финальной задачи, осуществляемому во время последнего третьего этапа. На каждом этапе решение подзадач проверялось с помощью системы автоматической системы оценивания, которая запускала решение участников с теми или иными параметрами в соответствии с приемочными тестами.

Первая итерация

Участникам необходимо разработать базовый функционал для скриптов `setup.py`, `vendor.py`, `scenter.py`, `car.py`, реализовать Ethereum-контракты `Management Contract` и `Battery Management Contract`, тем самым решая следующие подзадачи:

user story (*)	**acceptance criteria**
US-001 Регистрация аккаунта компании производителя ПО	все
US-002 Регистрация контрактов	все
US-005 Создание аккаунта производителя батарей	все
US-007 Регистрация нового производителя	AC-007-01
US-010 Поштучная регистрация производимых батарей	все
US-020 Создание аккаунта сервисного центра	все
US-021 Регистрация аккаунта сервисного центра	все
US-022 Создание аккаунта электромобиля	все
US-023 Получение аккаунта электромобиля	все
US-024 Регистрация аккаунта электромобиля	все

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- настройка приватной сети Ethereum для тестирования решения в ходе процесса разработки;
- настройка окружения разработки программного обеспечения для использования системы ведения версий `Git`, Python-библиотек для взаимодействия с узлами блокчейн платформы Ethereum и компиляции и регистрации Ethereum контрактов;
- написание и отладка программ на языке Python, работающих с параметрами командной строки;
- обеспечение отправки JSON-RPC запросов (включая платежные транзакции и транзакции на вызов методов Ethereum контрактов) из программ на языке Python к провайдерам `Web3`;
- реализация алгоритмов работы с приватными ключами для подписи Ethereum транзакций;
- написание и отладка Ethereum контрактов на языке Solidity.

Вторая итерация

Участникам необходимо завершить работу над минимально достаточным функционалом децентрализованного приложения, который бы позволял производить проверку тяговых аккумуляторных батарей на контрафакт, регистрировать новую сделку, подтверждать и завершать ее, обеспечивал бы экономическое взаимодействие

участников для стандартного сценария поведения. Данный функционал покрывался следующими подзадачами:

user story (*), все acceptance criteria
US-003 Установка сборов за регистрацию одной батареи
US-006 Получение информации о необходимом депозите за регистрацию производителя батарей
US-007 Регистрация нового производителя
US-008 Защита от повторной регистрации нового производителя
US-009 Получение информации о сборе за регистрацию одной батареи
US-011 Регистрация батарей с закрепленным сбором за регистрацию батарей
US-012 Получение размера остатка по депозиту
US-013 Пакетная регистрация производимых батарей
US-015 Поштучная продажа новых батарей
US-017 Генерация прошивок аппаратных ключей батарей
US-018 Изменение количества заряда батареи
US-019 Получение информации о батарее из прошивки
US-025 Проверка подлинности батареи сервисным центром
US-026 Проверка подлинности батареи электромобилем
US-027 Получение адреса контракта для замены батареи
US-029 Получение информации, что контракт замены в ожидании подтверждения
US-030 Получение условий контракта
US-031 Подтверждение согласия с условиями контракта
US-032 Подтверждение готовности к физической замене батарей
US-033 Подтверждение контракта
US-037 метод setBatteryManagementContract()
US-038 метод registerVendor()
US-039 метод registerBatteries()
US-040 метод setFee()
US-041 метод registerServiceCenter()
US-042 метод registerCar()
US-043 метод createBattery()
US-044 метод transfer() (с указанием одного идентификатора батареи)
US-045 метод delegatedApprove()
US-046 метод initiateDeal()
US-048 метод agreeToDeal()

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- разработка архитектуры простого децентрализованного приложения;
- реализация хранилища уникальных цифровых объектов и алгоритмов изменения их свойств на базе стандарта ERC721 (non-fungible tokens);
- реализация хранилища расчетных токенов и алгоритмов передачи прав владения на базе стандарта ERC20;
- реализация алгоритмов:
  - упаковки цифровой информации;
  - проверки истинности информации с использованием цифровой подписи;
- выстраивание процесса интеграционного тестирования компонент децентрализованного приложения.

Третья итерация

Работа участников должна быть направлена на наращивание функционала децентрализованного приложения и оптимизацию его работы, а также разработку инструмента сбора статистики. После завершения итерации у участников должно быть 5 Python-скриптов и 4 Ethereum контракта, закрывающие последнии набор подзадач:

user story (*), все acceptance criteria
US-004 Регистрация контрактов с постоянными адресами
US-014 Безопасная пакетная регистрация производимых батарей
US-016 Пакетная продажа новых батарей
US-028 Отправка запроса на отмену сделки
US-034 Установка времени запрета разблокировки токенов
US-035 Запрос на разблокирование токенов
US-036 Отсутствие подтверждения замены батарей для завершения сделки
US-047 метод setTimeoutThreshold()
US-050 метод releaseTokensByCar()
US-051 метод releaseTokensByServiceCenter()
US-052 метод cancelDeal()
US-053 Передача прав на батарею новому владельцу
US-054 Отложенная передача прав владения
US-055 Регистрация прав владения
US-056 Оптимизация по использованию вычислительных ресурсов
US-057 Оптимизация комиссии за валидацию транзакций
US-058 Сбор статистики

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- оптимизация работы Ethereum контрактов;
- организация взаимодействия с ненадежными источниками данных;
- реализация обхода цепочки блоков с последующим извлечением информации из транзакций для сбора статистики о работе децентрализованного приложения.

9.3. Условия проведения

- В первые день соревнований все члены команд получают ноутбук со следующим набором установленного программного обеспечения:
  - клиент сети Ethereum `geth`;
  - компилятор языка Solidity `solc`;
  - набор Python инструментария Anaconda с установленными библиотеками `web3.py` и `py-solc`;
  - среды разработки PyCharm и WingIDE;
  - клиент системы ведения версий `git`.
  - Интернет-браузер Chrome.

  Каждый ноутбук имеет возможность выхода в сеть Интернет. Команды могут использовать собственные ноутбуки.

- Команды могут устанавливать дополнительное программное обеспечение, но после согласование с членами жюри.
- Участники во время командного этапа финального тура могут использовать интернет и заранее подготовленные библиотеки для решения задачи.
- Участники должны использовать язык программирования Python для написания программ, использующих командную строку. Для написания Ethereum контрактов участники могут использовать любой язык программирования.
- Участники не могут использовать помощь тренера, сопровождающего лица или привлекать третьих лиц для решения задачи.
- Финальная задача формулируется участникам в первый день финального тура, но участники выполняют решение задачи поэтапно. Критерии прохождения каждого этапа формулируются для каждого дня финального тура. За подзадачи, решенные в конкретном этапе начисляются, баллы. Баллы за подзадачи можно получить только в день, закрепленный за конкретным этапом.
- В начале первого дня состязаний участники каждой команды получают доступ к репозиторию на серверах GitLab.com. Каждая команда имеет свой собственный репозиторий. Члены других команд не имеют доступ к чужим репозиториям.
- В начале первого дня состязний участникам выдется описание интерфейсов контрактов `Management Contract`, `Battery Management Contract`, `Deal contract`, `Currecncy token contract`, также участникам выдается исходный код `Service provider wallet` контракта, поскольку не является предметом оценки.
- В течение дня не ведется учет количества изменений, которые команды регистрируют в Git-репозитории.
- В конце каждого дня финального этапа жюри проверяет решение участников на соответствие приемочным тестам для каждой подзадачи, входящей в набор для соответствующей итерации.
- Баллы за все подзадачи, для которых прошло приемочное тестирование, определяют баллы, набранные командой в данный день соревнований.
- После выставления баллов, командам предоставляется доступ к системе автоматического тестирования, ответственной за проведение приемочных тестов в конкретный день состязаний, так что члены команды могут ознакомиться с логикой проверки и подать апелляцию, если не согласны с корректностью проведения тестов.
- После рассмотрения сути апелляции, жюри вправе провести тестирование вручную и назначить команде баллы за соответствующие подзадачи.
- В начале следующего дня состязаний жюри выдает всем командам свое решение подзадач предыдущего дня, которое команды могут использовать для того, чтобы решать следующий набор подзадач.

9.4. Процедура проведения приемочного тестирования и критерии оценки

Для каждого дня соревнований (для каждой итерации) справедлива следующая процедура приемочного тестирования:

- В конце каждого дня финального этапа команды должны сформировать запрос на слияние (*Merge Request*) из своей ветки исходного кода в основную ветку (*master*) в Git-репозитории.
- Команда ответственна за то, чтобы в запросе на слияние не должно быть конфликтов. Запрос на слияние с конфликтами может не рассматриваться жюри для выполнения приемочного тестирования.
- После того, как все команды отправили запросы на слияние, жюри одобряет все запросы и приступает к приемочному тестированию, для тех подзадач, которые входят в соответствующую итерацию. Для этого исходный код приложения команды копируется на сервер жюри, и прогоняются автоматические тесты на соответствие решения участников требованиям к приемочным тестам (*acceptance criteria*).
- Если все приемочные тесты для данной подзадачи пройдены успешно, команда получает баллы за данную подзадачу. Если хотя бы один тест не проходит, то баллы за данное подзадачу не начисляются.

Приемочные тесты для каждой подзадачи описаны в разделе "Подробное описание подзадач".

Дальше перечислены баллы, которые получала команда за решение подзадач в каждой итерации.

Максимальное количество баллов, которое могла набрать команда за решение всех подзадач — 400.

Первая итерация

user story	**баллы**
US-001 Регистрация аккаунта компании производителя ПО	2
US-002 Регистрация контрактов	12
US-005 Создание аккаунта производителя батарей	2
US-007 Регистрация нового производителя	8
US-010 Поштучная регистрация производимых батарей	6
US-020 Создание аккаунта сервисного центра	2
US-021 Регистрация аккаунта сервисного центра	6
US-022 Создание аккаунта электромобиля	2
US-023 Получение аккаунта электромобиля	2
US-024 Регистрация аккаунта электромобиля	6

Максимальное количество баллов за итерацию — 48.

Вторая итерация

user story	баллы
US-003 Установка сборов за регистрацию одной батареи	6
US-006 Получение информации о необходимом депозите за регистрацию производителя батарей	4
US-007 Регистрация нового производителя	4
US-008 Защита от повторной регистрации нового производителя	4
US-009 Получение информации о сборе за регистрацию одной батареи	4
US-011 Регистрация батарей с закрепленным сбором за регистрацию батарей	4
US-012 Получение размера остатка по депозиту	4
US-013 Пакетная регистрация производимых батарей	6
US-015 Поштучная продажа новых батарей	6
US-017 Генерация прошивок аппаратных ключей батарей	4
US-018 Изменение количества заряда батареи	3
US-019 Получение информации о батарее из прошивки	8
US-025 Проверка подлинности батареи сервисным центром	9
US-026 Проверка подлинности батареи электромобилем	9
US-027 Получение адреса контракта для замены батареи	16
US-029 Получение информации, что контракт замены в ожидании подтверждения	4
US-030 Получение условий контракта	8
US-031 Подтверждение согласия с условиями контракта	16
US-032 Подтверждение готовности к физической замене батарей	4
US-033 Подтверждение контракта	16
US-037 метод setBatteryManagementContract()	2
US-038 метод registerVendor()	4
US-039 метод registerBatteries()	3
US-040 метод setFee()	2
US-041 метод registerServiceCenter()	2
US-042 метод registerCar()	2
US-043 метод createBattery()	3
US-044 метод transfer() с указанием одного идентификатора батареи	2
US-045 метод delegatedApprove()	4
US-046 метод initiateDeal()	6
US-048 метод agreeToDeal()	6

Максимальное количество баллов за итерацию — 175.

Третья итерация

user story	баллы
US-004 Регистрация контрактов с постоянными адресами	20
US-014 Безопасная пакетная регистрация производимых батарей	6
US-016 Пакетная продажа новых батарей	6
US-028 Отправка запроса на отмену сделки	8
US-034 Установка времени запрета разблокировки токенов	6
US-035 Запрос на разблокирование токенов	10
US-036 Отсутствие подтверждения замены батарей для завершения сделки	10
US-047 метод setTimeoutThreshold()	2
US-050 метод releaseTokensByCar()	4
US-051 метод releaseTokensByServiceCenter()	4
US-052 метод cancelDeal()	4
US-053 Передача прав на батарею новому владельцу	8
US-054 Отложенная передача прав владения	12
US-055 Регистрация прав владения	8
US-056 Оптимизация по использованию вычислительных ресурсов	33
US-057 Оптимизация комиссии за валидацию транзакций	6
US-058 Сбор статистики	30

Максимальное количество баллов за итерацию — 177.

9.5. Подробное описание подзадач

Единой точкой взаимодействия с децентрализованным приложением для всех компонент является `Management Contract`. Практически для любого сценария, за исключением сценариев создания аккаунтов и регистрации (deployment) контрактов в сети блокчейн, компоненты должны для получения адреса `Management Contract` обращаться к файлу в формате `JSON` `database.json`, находящемуся в текущей рабочей директории:

```
{"mgmtContract": "0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd"}
```

Если это явно не прописано в описании функционала соответствующей компоненты, она не должна оставлять после себя никаких файлов-данных. Аналогично и с получением данных - чтение данных из файла должно происходить только в нескольких случаях

- для получения адреса `Management Contract`,
- для получения `Application Binary Interface` контракта или его байткода,
- или если это явно указано в описании компоненты,

в остальных случаях данные должны быть получены из блокчейн.

Все компоненты, отправляющие транзакции в блокчейн, должны уметь обрабатывать ситуации, когда:

- недоступно подключение к узлу Ethereum (сообщение об ошибке ''`No connection to node`'');
- на счету аккаунта, от чьего имени выполняется транзакция, недостаточно средств, чтобы покрыть все затраты по использованному газу (сообщение об ошибке ''`No enough funds to send transaction`'');
- транзакция может не включаться в блок продолжительное время в зависимости от нагрузки на сеть (сообщение об ошибке ''`Transaction is not validated too long`'').

При возникновении данных событий в терминал должно выдаваться сообщение понятное пользователю, а не stack trace.

Подключение к узлу Ethereum, там где это необходимо, должно происходить с помощью автоматического определения Web3 провайдера. Для этого гарантируется, что перед запуском той или иной компоненты децентрализованного приложения будет установлена переменная окружения `WEB3_PROVIDER_URI` (`http://web3py.readthedocs.io/en/latest/quickstart.html#provider-uri`).

Настройка децентрализованного приложения

Развертывание и настройка децентрализованного приложения выполняется компанией производителем программного обеспечения (тем, кто разрабатывает данный проект). Предполагается, что в потенциальные пользователи проекта (производители батарей, владельцы станции сервисного обслуживания и производители электромобилей) могут ознакомиться с опубликованными в открытом доступе исходными кодами контрактов децентрализованного приложения, принять решение на основе этого, доверять ли этому разработчику и начать пользоваться сервисом.

Фиксация контрактов в блокчейн гарантирует, что приозводитель не сможет изменить логику работы приложения позднее, т.е. условия участия в системе, с которыми ознакомились пользователи, не поменяется.

Функционал децентрализованного приложения должен быть доступен сразу, минуя фазы настройки, когда децентрализованное приложение развернутое в блокчейн, но еще находится в нерабочем состоянии. Поэтому первичная настройка базовых параметров по максимуму должна выполняться вовремя инициализации контрактов во время их регистрации (deploy).

Имя скрипта
`setup.py <command> [options]`

В тех случаях, когда скрипту необходимо отправлять транзакции для изменения состояния блокчейн, аккаунт и пароль, которые будут использоваться для доступа к приватному ключу для подписи транзакций, должны браться из файла в формате JSON `account.json`:

```
{
    "account": "0x5338d77b5905cdeea7c55a1f3a88d03559c36d73",
    "password": "s0me_p455w0rd"
}
```

US-[3]USCounter Регистрация аккаунта компании производителя ПО

Использование скрипта
`setup.py --new <password>`
Подключается к локальному узлу Ethereum и создает новый аккаунт с паролем, заданным в строке ввода. Также создает файл базы данных `account.json`, в котором сохраняется аккаунт и пароль для дальнейшего использования данным производителем. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --new 's0me_p455w0rd'` `0x009e2Ec03B40360d64A00E5bCd6D819ba817D24b`
Комментарий: На локальном узле создан пользователь. В текущей рабочей директории создается файл `account.json`, в котором сохраняется аккаунт и пароль в формате `JSON` для дальнейшего использования данным производителем.

EPIC-[2]EpicCounter Регистрация и конфигурация контрактов

US-[3]USCounter Регистрация контрактов

Использование скрипта
`setup.py --setup <service fee>`
Подключается к узлу Ethereum и регистрирует контракты в сети блокчейн: • `Service Provider Wallet Contract` • `Management Contract` • `Battery Management Contract` • `Currency Token Contract` Параметр `<service fee>` используется для того, чтобы при создании контракта `Management Contract` указать величину сборов за регистрацию одной батареи. В терминал выводится адрес `Management Contract`, `Service Provider Wallet Contract` и `Currency Token Contract`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# setup.py --setup 0.01`
`Management contract: 0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd`
`Wallet contract: 0xcb3ad4765bc9056a6bb09d24bb5ef3306532db9a`
`Currency contract: 0x2380943c84daa97b5d3b4be6d77692b2eb486491`

Комментарий: В блокчейн сеть отправляются транзакции для регистрации и первичной настройке контрактов. Транзакции успешно включены в один или несколько блоков. `Management Contract` можно проинспектировать на предмет установки: • суммы сборов за регистрацию одной батареи; • адреса `Service Provider Wallet Contract`; • адреса `Battery Management Contract`. `Battery Management Contract` можно проинспектировать на предмет установки: • адреса `Management Contract`; • адреса `Currency Token Contract`. `Currency Token Contract` можно проинспектировать на наличия на балансе компании производителя ПО 10^{20} токенов, необходимых для экономического взаимодействия участников системы. В текущей рабочей директории создается файл в формате `JSON` `database.json`, в котором указан адрес `Management Contract`.

US-[3]USCounter Установка сборов за регистрацию одной батареи

Несмотря на то, что величина сборов за регистрацию одной батареи устанавливается во время регистрации контрактов, должна быть возможность сменить данную величину, например, из-за требований рынка: сбор оказался в самом начале слишком высокий или, наоборот, инфляция требует поднять величину сбора, чтобы сделать проект экономически целесообразным.

Использование скрипта
`setup.py --setfee <service fee>`
Подключается к узлу Ethereum и отправляет запрос к `Management Contract` на изменение сумму сборов за регистрацию одной батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --setfee 0.01` `Updated successfully`
Комментарий: В блокчейн сеть отправляется транзакция. Транзакция успешно включена в блок. `Management Contract` можно проинспектировать на предмет изменений.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --setfee 0.01` `No permissions to change the service fee`
Комментарий: Если попытка изменения суммы сборов происходит не от аккаунта компании производителя ПО, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

US-[3]USCounter Регистрация контрактов с постоянными адресами

Одним из конкуретных преимуществ приложения может быть то, что адреса контрактов, по развернуты контракты - заранее известные. Это позволит повысить доверие к системе (адреса заранее указаны в контрактах, а не передаются на этапе инициализации) и упростить процедуру подключения новых пользователей (адрес контракта не нужно настраивать через конфигурационные файлы).

В реальной жизни данный функционал особенно востребован в случае присутствия какого-то контракта (например, контракта токена) в разных, так называемых, side chains, когда функционал контракта распределяется между отдельными блокчейн сетями: например при наличии контракта non-fungible token его сложная бизнес-логика изменения его свойств или генерации других контрактов, реализована в одном блокчейн, где стоимость вычислений мала, а более легковесная логика перехода прав владения, связанная со сделками купли-продажи с использованием ликвидной криптовалюты, - в другом. Тогда выглядит логичным вариант, когда в разных блокчейн сетях данный контракт имеет один и тот же адрес.

Поскольку, подразумевается, что исходный код скрипта `setup.py` доступен каждому, то в нем не должно храниться приватного ключа для подписи транзакций регистрации контрактов. Также приватный ключ не должен извлекаться из других файлов, поскольку их также пришлось бы выкладывать в открытый доступ для максимального доверия. В рамках данной US предлагается найти способ, который требует наличия только одного заранее аккаунта, созданного через `setup.py -new`, информация о котором передается через `account.json`.

Использование скрипта
`setup.py --setup <service fee>`
Подключается к узлу Ethereum и регистрирует контракты в сети блокчейн: • `Service Provider Wallet Contract` • `Management Contract` • `Battery Management Contract` • `Currency Token Contract` Вне зависимости от того какой аккаунт принадлежит компании производителю ПО и вне зависимости от того, в блокчейн с каким идентификатором сети происходит регистрация контрактов, каждый из контрактов будет иметь один и тот же адрес.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# echo $WEB3_PROVIDER_URI`
`http://127.0.0.1:8545`
`# setup.py --setup 0.01`
`Management contract: 0x519D75381DfD9e8e1128698C6B6f9f76FC242404`
`Wallet contract: 0xC1Af0c06cAe672B3514Be9683663c88d8Db920bf`
`Currency contract: 0xbFD04af48C978cC0d9Bc5E06d9593Cb4fb7f6f98`
`# setup.py --new 'secr3tw0rd'`
`# env WEB3_PROVIDER_URI=http://127.0.0.1:7545 setup.py --setup 0.01`
`Management contract: 0x519D75381DfD9e8e1128698C6B6f9f76FC242404`
`Wallet contract: 0xC1Af0c06cAe672B3514Be9683663c88d8Db920bf`
`Currency contract: 0xbFD04af48C978cC0d9Bc5E06d9593Cb4fb7f6f98`

Комментарий: Транзакции после вызова первой команды `setup.py` отправляются на узел, принадлежащий одной блокчейн сети. А транзакции после вызова второй команды отправляются на узел, принадлежащий другой блокчейн сети. Адреса всех четырех контрактов, получившихся после регистрации в одной и в другой сети - одинаковые: адрес `Management Contract` в одной сети, равен адресу `Management Contract` в другой сети, адрес `Service Provider Wallet Contract` в одной сети, равен адресу `Service Provider Wallet Contract` в другой сети и т.д.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --setup 0.01` `At least one address already occupied.`
Комментарий: Если при регистрации контрактов в сети обнаруживается, что один из адресов под которым контракты должны зарегистрироваться уже занят под адрес пользователя или адрес контракта, то выдается ошибка. Транзакции на регистрацию контрактов не отправляется в блокчейн. *Примечание:* данный критерий будет проверяться только при успешной приемке предыдущего критерия.

Производитель батарей

Имя скрипта
`vendor.py <command> [options]`

В тех случаях, когда скрипту необходимо отправлять транзакции для изменения состояния блокчейн, аккаунт и пароль, которые будут использоваться для доступа к приватному ключу для подписи транзакций, должны браться из файла в формате JSON `account.json`:

```
{
    "account": "0x5338d77b5905cdeea7c55a1f3a88d03559c36d73",
    "password": "s0me_p455w0rd"
}
```

EPIC-[2]EpicCounter Регистрация нового производителя батарей

US-[3]USCounter Создание аккаунта производителя батарей

Использование скрипта
`vendor.py --new <password>`
Подключается к локальному узлу Ethereum и создает новый аккаунт с паролем, заданным в строке ввода. Также создает файл базы данных `account.json`, в котором сохраняется аккаунт и пароль для дальнейшего использования данным производителем. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --new 's0me_p455w0rd'` `0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd`
Комментарий: На локальном узле создан пользователь. В текущей рабочей директории создается файл `account.json`, в котором сохраняется аккаунт и пароль в формате `JSON` для дальнейшего использования данным производителем.

US-[3]USCounter Получение информации о необходимом депозите за регистрацию производителя батарей

Использование скрипта
`vendor.py --regfee`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения размера регистрационного взноса у `Management Contract`. В терминал выводится полученная информация (в *ether*) в виде десятичной дроби с точностью до 6 знаков после запятой, незначащие завершающие нули не выводятся.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --regfee` `Vendor registration fee: 0.5`
Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

US-[3]USCounter Регистрация нового производителя

Необходим депозит для предотвращения нежелательных регистраций.

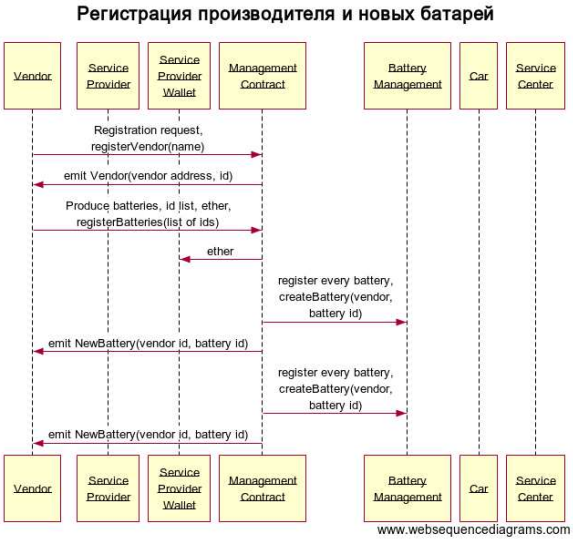


Рис. 9.2: Диаграмма взаимодействия компонент

Использование скрипта
`vendor.py --reg <vendor name> <fee>`
Подключается к узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию нового производителя батарей, название которого передается в качестве первого параметра к команде, а отправитель запроса ассоциируется как представитель данного производителя. Со счета отправителя запроса списывается сумма указанная в качестве второго параметра в качестве депозита за предоставляемый сервис. Сумма указывается в *ether*. В терминал выводится статус выполнения команды и идентификатор производителя батареи, который является шестнадцатеричной записью первых четырех байт от хэш-суммы (*keccak*256) адреса, наименования производителя и номера блока, в который включена транзакция на регистрацию производителя, объединенных вместе. *Примечание:* Поскольку частью процесса регистрации производителя является внесение депозита, то на счету аккаунта, от имени которого происходит регистрация должна быть достаточная для депозита сумма.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --reg 'Tesla Gigafactory' 0.5`
`Success.`
`Vendor ID: ______`

Комментарий: В блокчейн сеть отправляется транзакция на регистрацию производителя, транзакция успешно включена в блок, `Management Contract` можно проинспектировать на предмет добавленных данных (имя производителя ассоциировано с адресом отправителя, за данным производителем закреплен депозит). Баланс `Service Provider Wallet Contract` увеличивается на сумму депозита.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Nissan Energy' 0.1` `Failed. Payment is low.`
Комментарий: Если при попытке регистрации производителя батарей указывается сумма недостаточная для депозита, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Nissan Energy' 0.5` `Failed. No enough funds to deposit.`
Комментарий: Если при попытке регистрации производителя батарей на счету аккаунта, который регистрирует производителя, недостаточно средств, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

US-[3]USCounter Защита от повторной регистрации нового производителя

Должна обеспечиваться только одна ассоциативная связь между аккаунтом производителя и его названием. В перспективе могут быть использованы алгоритмы машинного обучения для обеспечения защиты уникальности названия.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Tesla Gigafactory' 0.5` `Failed. The vendor name is not unique.`
Комментарий: При попытке регистрации производителя батарей с названием, которое уже зарегистрировано, выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Nissan Energy' 0.5` `Failed. The vendor address already used.`
Комментарий: При попытке регистрации производителя батарей из под аккаунта, с которым уже регистрировался производитель, выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

EPIC-[2]EpicCounter Регистрация новых батарей

При регистрации новых батарей владельцем каждой батареи в `Battery Management Contract` назначается производитель данной батареи.

US-[3]USCounter Получение информации о сборе за регистрацию одной батареи

Использование скрипта
`vendor.py --batfee`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения размера взноса за одну батарею у `Management Contract`. Размер взноса должен быть возвращен тот, который был установлен на момент регистрации производителя в системе. В терминал выводится полученная информация (в *ether*) в виде десятичной дроби с точностью до 6 знаков после запятой, незначащие завершающие нули не выводятся.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --batfee` `Production fee per one battery: 0.01`
Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# setyp.py --setfee 0.005`
`Updated successfully`
`# vendor.py --batfee`
`Production fee per one battery: 0.01`

Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

US-[3]USCounter Поштучная регистрация производимых батарей

При положительном балансе депозита производителя батарей в `Management Contract` можно регистрировать батарею без перечисления средств. Иначе, при регистрации батареи необходимо пополнять депозит до суммы, достаточной для регистрации.

Использование скрипта
`# vendor.py --bat <number> [<deposit>]`
Подключается к узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию новой батареи. Для определения производителя данной батареи в `Management Contract` используется аккаунт отправителя транзакции. С депозита производителя списывается сумма равная сбору за регистрацию одной, зафиксированному на момент регистрации производителя батарей и умноженному на количество батарей, которые предлагается зарегистрировать. В командной строке может быть передана сумма для пополнения депозита (сумма указывается в *ether*). В терминал выводится идентификатор батареи, который формируется по тому же принципу, что и адрес Ethereum. Приватный ключ, которому соответствует данный идентификатор, генерируется (псевдо)случайным образом в диапазоне от 0 до $2^{256} - 1$. *Примечание:* Поскольку частью процесса регистрации батареи является пополнение депозита, то на счету аккаунта, от имени которого происходит регистрация должна быть достаточная для пополнения сумма.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 1` `Created battery with ID: d65bc90bb030472db9a6e9d653cb3a24bb556a3f`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на сумму, необходимую для регистрации одной батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 1` `Failed. No enough funds to register object.`
Комментарий: Если при попытке регистрации батареи депозит на `Management Contract` не содержит достаточно средств, то выдается ошибка. Транзакция на регистрацию батареи не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 1 0.0005` `Created battery with ID: 915d3b4be6d776923824daa97bb2eb4860943c84`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Баланс `Service Provider Wallet Contract` увеличивается на сумму пополнения депозита. Депозит производителя батарей в `Management Contract` изменяется на сумму, необходимую для регистрации одной батареи с учетом средств, перечисленных для пополнения депозита.

US-[3]USCounter Регистрация батарей с закрепленным сбором за регистрацию батарей

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --batfee`
`Production fee per one battery: 0.01`
`# setyp.py --setfee 0.005`
`Updated successfully`
`# vendor.py --bat 1`
`Created battery with ID: 44bf53f1bda83c64f29a61ed671cfc8ff7fa3b58`

Комментарий: В блокчейн сеть отправляется транзакция на регистрацию батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на 0.1 *ether*.

US-[3]USCounter Получение размера остатка по депозиту

Использование скрипта
`vendor.py --deposit`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции `Management Contract` получения остатка по депозиту для аккаунта, от которого выполняется вызов.. В терминал выводится полученная информация (в *ether*) в виде десятичной дроби с точностью до 6 знаков после запятой, незначащие завершающие нули не выводятся.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --deposit` `Deposit: 0.095`
Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --deposit` `Vendor account is not registered.`
Комментарий: При попытке получения остатка депозита от аккаунта, который не зарегистрирован как производитель батарей, выдается ошибка. Нет транзакций в блокчейн для `Management Contract`.

US-[3]USCounter Пакетная регистрация производимых батарей

При положительном балансе депозита производителя батарей в `Management Contract` регистрировать батареи можно в количестве, соответствующем остатку депозита. Иначе, при регистрации батареи необходимо пополнять депозит до суммы, достаточной для регистрации.

Примечание: Данные сценарии проверяются только в случае успешного прохождения тестирования *"Поштучная регистрация производимых батарей"*.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 3`
`Created battery with ID: 1e6d7ef161b777d3f7422bec456bc251b459e77`
`Created battery with ID: c64a6b0267dc5fc156592b958078644a1946f8e`
`Created battery with ID: a8faf5e89eb2107a74bdb5d29141bf5e4615cd8`

Комментарий: В блокчейн сеть отправляется **одна** транзакция на регистрацию трех батарей, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на сумму, необходимую для регистрации трех батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 3` `Failed. No enough funds to register object.`
Комментарий: Если при попытке регистрации батарей депозит на `Management Contract` не содержит достаточно средств, то выдается ошибка. Транзакция на регистрацию батарей не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 4 0.001`
`Created battery with ID: 8069dfa9abaef192224c962f81a81f6fdf84241f`
`Created battery with ID: cfbc3f5d56753efa436251d382a74b607b3241b0`
`Created battery with ID: 11d87f2214deb3f901fc91acfee053e5b9f1a7c4`
`Created battery with ID: 17a51f0cf04f3544410a079da96327a10e8acd04`

Комментарий: В блокчейн сеть отправляется **одна** транзакция на регистрацию четырех батарей, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Баланс `Service Provider Wallet Contract` увеличивается на сумму пополнения депозита. Депозит производителя батарей в `Management Contract` изменяется на сумму, необходимую для регистрации четырех батарей с учетом средств, перечисленных для пополнения депозита.

US-[3]USCounter Безопасная пакетная регистрация производимых батарей

При пакетной регистрации батарей может сложится такая ситуация, что потребляемый при обработке транзакции газ превысит максимальное количество газа, допустимое для транзакций, включенных в блок. В этом случае, регистрация батарей должна происходить в несколько транзакций. Количество идентификаторов батарей, включенных в транзакцию не должно быть предварительно задано, вместо этого оно должно вычисляться автоматически в зависимости от настроек блокчейн — максимальное количество газа, на сколько могут потребить все транзакции, включенные в один блок).

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 5800`
`Created battery with ID: ae318f3b81c25acf9d8eb9cf004b7dc8d3980f7`
`Created battery with ID: ae8eff9801b40a5e25af5212d50e2c7d8ca683f`
`Created battery with ID: c51eb5db9fefc2497203a8f19a74f449d24f4ba`
`. . .`
`Created battery with ID: 121759890801155f310e26b541c861ed5dfc423`
`Created battery with ID: dbcdd9074c66705051cb8645c892080c3c742ed`
`Created battery with ID: e42474a90e439aef1c5a7d872eee5f62b9d8c19`

Комментарий: В блокчейн сеть отправляется **две** транзакции на регистрацию 5800 батарей, обе транзакции успешно включены в **разные** блоки, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на сумму, необходимую для регистрации 5800 батарей.

EPIC-[2]EpicCounter Дистрибьюция батареи

Дистрибьюция батарей происходит в двух случаях:

- установка новой батареи в автомобиль на заводе производителе;
- продажа батарей на замену сервисным центрам и партнерам.

И в том, и в другом случае должен смениться владелец батареи в `Battery Management Contract`.

US-[3]USCounter Поштучная продажа новых батарей

Поскольку сценариев того, как батарея может отпускаться от производителя новым владельца достаточно много, то экономическая составляющая этого процесса не будет покрываться данным сценарием.

Использование скрипта
`# vendor.py --owner <battery_id> <new_owner>`
Подключается к узлу Ethereum и отправляет запрос к `Battery Management Contract` на смену владельца батареи. Для определения текущего владельца батареи в `Management Contract` используется адрес отправителя транзакции. В терминал выводится статус выполнения команды.

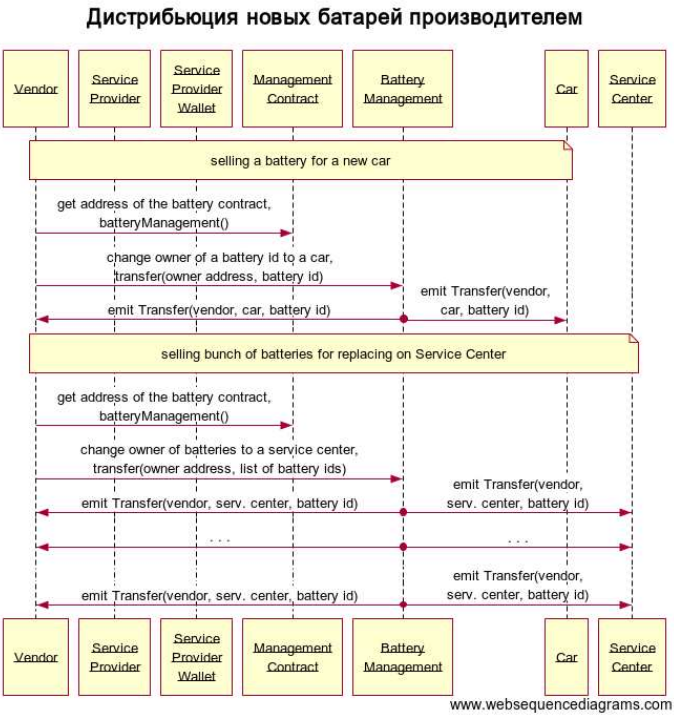


Рис. 9.3: Диаграмма взаимодействия компонент

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --owner ae318f3b81c25acf9d8eb9cf004b7dc8d3980f7 \`
`0x50dfbef376d01e05e134a9b3a322afef50111ec1`
`Success`

Комментарий: В блокчейн сеть отправляется транзакция на изменение владельца батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --owner 121759890801155f310e26b541c861ed5dfc423 \`
`0x0dcd2f752394c41875e259e00bb44fd505297caf`
`Failed. Not allowed to change ownership.`

Комментарий: При попытке смены владельца у батареи, для которой право владения перемещено другому, выдается ошибка. Транзакция на изменения владельца батареи не отправляется в блокчйен. Нет локальных вызов методов контракта для проверки текущего владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --owner ce2e25258e041d07bc3c4906f2b45d68c9aa3420 \`
`0xbe428c3867f05dea2a89fc76a102b544eac7f772`
`Failed. Not allowed to change ownership.`

Комментарий: При попытке смены владельца у батареи на владельца, который не зарегистрирован ни как сервисный центр, ни как электромобиль, выдается ошибка. Транзакция на изменения владельца батареи не отправляется в блокчйен. Нет локальных вызов методов контракта для проверки текущего владельца батареи.

US-[3]USCounter Пакетная продажа новых батарей

Если вместо идентификатора батареи передается путь до файла, то происходит смена владельца для всех батарей, чьи идентификаторы перечислены в файле.

В файле каждый новый идентификатор, записанный в шестнадцатеричном виде и начинающийся с `0x`, находится в новой строке.

Путь до файла может быть, как абсолютным, так и относительным.

Примечание: Данный сценарий проверяются только в случае успешного прохождения тестирования *"Поштучная продажа новых батарей"*.

Подразумевается, что для всех батарей из списка текущий владелец один и тот же. Сценарии, когда у какой-то из батарей нельзя поменять владельца из-за ошибки владения, проверяться не будет.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --owner /path/to/file 0x50dfbef376d01e05e134a9b3a322afef50111ec1` `Success`
Комментарий: В блокчейн сеть отправляется **одна** транзакция на изменение владельца батарей, перечисленных в файле, доступном по пути `/path/to/file`. Транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батарей.

Аппаратный ключ защиты (HASP)

Поскольку, в случае олимпиадной задачи, не предоставляется работать с реальным аппаратным ключом защиты, то функционал ключа будет эмулироваться скриптом на языке Python, он же будет файлом прошивки аппаратного ключа, содержащим статические (read-only memory, хранящая неизменяемые данные специфичные для конкретного аппаратного ключа).

Изменяемые данные (data storage) аппаратного ключа будут эмулироваться через файл, в котором информация хранится в форматае JSON.

EPIC-[2]EpicCounter Генерация прошивок

US-[3]USCounter Генерация прошивок аппаратных ключей батарей

Результатом регистрации батареи будет не только внесение изменений в блок-

чейн, но и генерация файла, содержащего прошивку аппаратного ключа конкретной батареи.

Примечание: Данный сценарий проверяется только в случае успешного прохождения тестирования *"Поштучная регистрация производимых батарей"* или *"Множественная регистрация производимых батарей"*.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 3`
`Created battery with ID: 842d20af461195aaa13f248df340b5e0b3d4042`
`Created battery with ID: ee4d92e6d4183caac6e45b2c31f74e37440ef11`
`Created battery with ID: 6244345b1459bde1cc14be06d11d00421dab3b5`

Комментарий: В директории `firmware`, находящейся в той же директории, что и скрипт `vendor.py`, добавляются три файла `842d20af.py`, `ee4d92e6.py`, `6244345b.py`. Если директории `firmware` не существует, то она автоматически создается. В каждом файле есть глобальная переменная `_privkey`, содержащая набор байт - приватный ключ соответствующей батареи. Приватный ключ может быть проверен на совпадение с батарей, если из него получить Ethereum адрес. Первые четыре байта адреса, записанные в шестнадцатеричном виде, должны совпадать с именем файла. *Примечание:* Если сценарий выполняется с запросом на регистрацию только одной батареи, то соответственно в директории должен появиться только один новый файл.

EPIC-[2]EpicCounter Операции с аппаратным ключом

Имя скрипта
`<battery_id>.py <command>`

Аппаратный ключ работает полностью автономно и не требует никаких данных из блокчейн.

US-[3]USCounter Изменение количества заряда батареи

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# 842d20af.py --charge`
Комментарий: Увеличивает значение в файле, эмулирующем data storage, значение на единицу. Если такого файла не было, то он создается и в него записывается 1. Имя файла с хранилищем данных должно совпадать с именем файла-прошивкой аппаратного ключа. Расширение файла `.json` - файл содержит информацию в формате JSON.

US-[3]USCounter Получение информации о батарее из прошивки

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# 842d20af.py --get`
`4`
`1515748372`
`27`
`8f8f14002e48267c85323a89f0d1c34deda1af13ef39e0aae851cdff0bdde18b`
`3197422d66ae3f1b11950dfbef376d01e05e134a9b3a322afef50111ec9bd656`

Комментарий: Выводит в терминал пять строк (статус батареи): 1. n - текущее количество проведенных зарядов батареи. Берется из файла, эмулирующего data storage, или 0, если такого файла нет. Целое число, максимум $2^{32} - 1$, записано в десятичном формате. 2. t Текущее время в формате unix time - количество секунд с 1 января 1970 года. Целое число, максимум $2^{32} - 1$, записано в десятичном формате. 3. v, r, s - компоненты цифровой подписи, где v - целое число, максимум $2^{32} - 1$, записано в десятичном формате; r, s - наборы байт длиной 32, записанный в шестнадцатеричном формате. Цифровая подпись формируется с использованием приватного ключа, соответствующего данной батареи. Подпись берется для хэш-суммы ($keccak256$) от объединенных вместе количества зарядов батарей и текущего времени ($m = n \cdot 2^{32} + t$). Действительность цифровой подписи может быть проверена через генерацию Ethereum адреса для данных количества зарядов батарей и текущего времени. Первые восемь шестнадцатеричных символа адреса должны совпадать с именем файла прошивки.

Регистрация участников системы

Для замены батарей взаимодействуют два участника системы: сервисный центр и электромобиль. Подразумевается, что электромобиль может быть полностью автономный, именно поэтому он выступает самостоятельным участником.

В общем виде вызов скрипта, отвечающего за операции выполняемые от имени сервисного центра, выглядит следующим образом:

Имя скрипта
`scenter.py <command> [options]`

В тех случаях, когда скрипту необходимо отправлять транзакции для изменения состояния блокчейн, аккаунт и пароль, которые будут использоваться для доступа к приватному ключу для подписи транзакций, должны браться из файла в формате JSON `scenter.json`:

```
{
    "account": "0x5338d77b5905cdeea7c55a1f3a88d03559c36d73",
    "password": "s0me_p455w0rd"
}
```

Скрипт, отвечающего за операции выполняемые от имени электромобиля:

Имя скрипта
`car.py <command> [options]`

Поскольку хранение информации о всей блокчейн в памяти электромобиля не оправдано, то должна предоставляться возможность для подготовки транзакций или подписи данных, передаваемых вне блокчейн. Для этого приватный ключ должен браться из файла в формате JSON `car.json`:

```
{
    "key": "338d77b5905cda7c55a1f3a885338d77b5905cdeea7c55a1f3a88d03559c36d73"
}
```

Все транзакции должны отправляться в сеть блокчейн с использованием данного ключа.

EPIC-[2]EpicCounter Регистрация сервисного центра

US-[3]USCounter Создание аккаунта сервисного центра

Использование скрипта
`scenter.py --new <password>`
Подключается к локальному узлу Ethereum и создает новый аккаунт с паролем, заданным в строке ввода. Также создает файл базы данных `scenter.json`, в котором сохраняется аккаунт и пароль для дальнейшего использования данным производителем. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --new 's0me_p455w0rd'` `0x5338d77b5905cdeea7c55a1f3a88d03559c36d73`
Комментарий: На локальном узле создан пользователь. В текущей рабочей директории создается файл `scenter.json`, в котором сохраняется аккаунт и пароль в формате `JSON` для дальнейшего использования данным сервисным центра.

US-[3]USCounter Регистрация аккаунта сервисного центра

Использование скрипта
`scenter.py --reg`
Подключается к локальному узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию аккаунта отправителя транзакции в качестве сервисного центра.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`scenter.py --reg` `Registered successfully`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию аккаунта сервисного центра. Транзакция успешно включена в блок. `Management Contract` можно проинспектировать на предмет изменений.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`scenter.py --reg` `Already registered`
Комментарий: Если при попытке аккаунта сервисного центра в `Management Contract` уже есть такой аккаунт в качестве сервисного центра или электромобиля, то выдается ошибка. Транзакция на регистрацию батареи не отправляется в блокчейн.

EPIC-[2]EpicCounter Регистрация электромобиля

US-[3]USCounter Создание аккаунта электромобиля

Использование скрипта
`car.py --new`
Случайным образом выбирается приватный ключ и записывается в файл `car.json`. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --new` `0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3`
Комментарий: Файл `car.json` создан в текущей рабочей директории.

US-[3]USCounter Получение аккаунта электромобиля

Использование скрипта
`car.py --account`
Выводит в терминал адрес аккаунта автомобиля на основе приватного ключа из файла `car.json`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --account` `0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3`
Комментарий: Адрес аккунта соответствует приватному ключу, из файл `car.json`, расположенному в текущей рабочей директории.

US-[3]USCounter Регистрация аккаунта электромобиля

Использование скрипта
`car.py --reg`
Подключается к локальному узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию аккаунта отправителя транзакции в качестве электромобиля.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`car.py --reg` `Registered successfully`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию аккаунта электромобиля. Транзакция успешно включена в блок. `Management Contract` можно проинспектировать на предмет изменений.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`car.py --reg` `Already registered`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию аккаунта электромобиля. Транзакция успешно включена в блок, выясняется, есть такой аккаунт в качестве сервисного центра или электромобиля. Локальных запросов для проверки аккаунта перед отправкой транзакции не создается.

Замена батарей

В данном разделе подробно рассматриваются действия, которые должны выполнить участники во время замены батарей. При этом действия не учитывают нежелательное поведение участников, когда участник сознательно отказывается от сделки или пытается мошенничать. Такие сценарии будут рассмотрены в следующих разделах.

EPIC-[2]EpicCounter Проверка подлинности батареи

Каждый участник системы может проверить батарею на подлинность, если имеет физический доступ к батарее.

US-[3]USCounter Проверка подлинности батареи сервисным центром

Использование скрипта
`scenter.py --verify <path>`
Обращается к файлу с прошивкой батареи и получает информацию о количестве циклов заряда, текущее время и цифровую подпись (статус батареи). После этого подключается к узлу Ethereum и выполняет **локальный** вызов функции проверки данных от батареи. В терминал выводится статус проверки, количество выполненных циклов заряда, идентификатор и название производителя батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --verify firmware/ee4d92e6.py`
`Verified successfully.`
`Total charges: 3`
`Vendor ID: ______`
`Vendor Name: 'Tesla Gigafactory'`

Комментарий: Проверка выполнена успешно. Нет транзакций в блокчейн для проверки батареи с идентификатором ee4d92e6d4183caac6e45b2c31f74e37440ef11.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --verify firmware/00360d2b.py` `Verification failed. Probably the battery forged.`
Комментарий: Проверка показывает, что данные от батареи не проходят валидацию в блокчейн - такая батарея не зарегистрирована. Нет транзакций в блокчейн для проверки батареи с идентификатором 00360d2b7d240ec0643b6d819ba81a09e40e5bcd.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --verify firmware/6244345b.py` `Battery with the same status already replaced. Probably the battery forged.`
Комментарий: Проверка показывает, что батарея с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5 и данным статусом уже участвовала в замене. Это указывает на возможную подмену прошивки, а, значит, мошенничество. Нет транзакций в блокчейн для проверки батареи с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5.

US-[3]USCounter Проверка подлинности батареи электромобилем

Использование скрипта
`car.py --verify <path>`
Обращается к файлу с прошивкой батареи и получает информацию о количестве циклов заряда, текущее время и цифровую подпись (статус батареи). После этого подключается к узлу Ethereum и выполняет **локальный** вызов функции проверки данных от батареи. В терминал выводится статус проверки, количество выполненных циклов заряда, идентификатор и название производителя батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# car.py --verify firmware/ee4d92e6.py`
`Verified successfully.`
`Total charges: 3`
`Vendor ID: ______`
`Vendor Name: 'Tesla Gigafactory'`

Комментарий: Проверка выполнена успешно. Нет транзакций в блокчейн для проверки батареи с идентификатором ee4d92e6d4183caac6e45b2c31f74e37440ef11.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --verify firmware/00360d2b.py` `Verification failed. Probably the battery forged.`
Комментарий: Проверка показывает, что данные от батареи не проходят валидацию в блокчейн - такая батарея не зарегистрирована. Нет транзакций в блокчейн для проверки батареи с идентификатором 00360d2b7d240ec0643b6d819ba81a09e40e5bcd.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --verify firmware/6244345b.py` `Battery with the same status already replaced. Probably the battery forged.`
Комментарий: Проверка показывает, что батарея с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5 и данным статусом уже участвовала в замене. Это указывает на возможную подмену прошивки, а, значит, мошенничество. Нет транзакций в блокчейн для проверки батареи с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5.

EPIC-[2]EpicCounter Подготовка к замене батарей

Когда электромобиль прибывает в сервисный центр для замены аккумуляторной батареи, то сервисный центр должен получить информацию о батарее, которая уже

установлена в автомобиле. На основании информации о батареях, участвующих в замене, сервисный центр составляет контракт, в рамках которого будет происходить выполнение работ по замене и дальнейшее экономическое взаимодействие.

US-[3]USCounter Получение адреса контракта для замены батареи

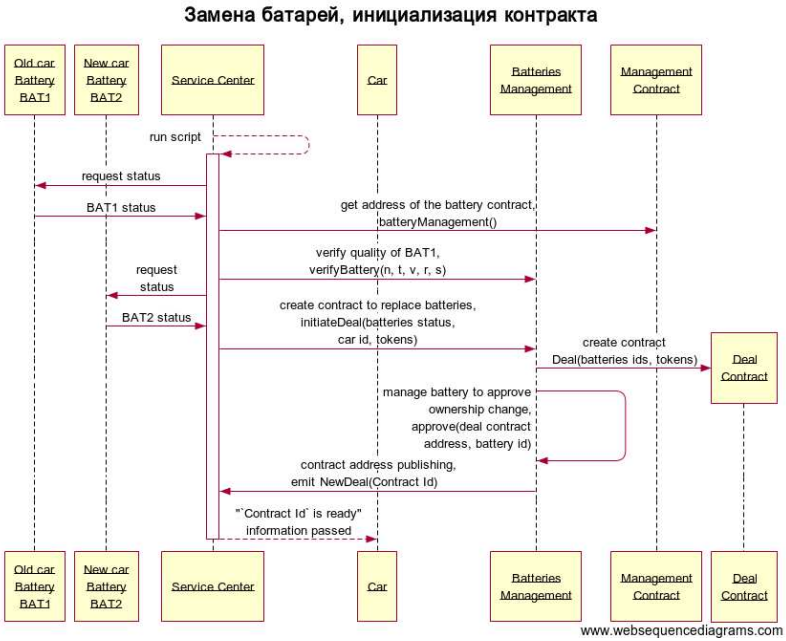


Рис. 9.4: Диаграмма создания контракта на замену батарей

Использование скрипта
`scenter.py --contract <new battery path> <old battery path> <car> <value>`
Обращается к файлам с прошивками батарей и получает информацию о статусе батарей. После этого подключается к узлу Ethereum и отправляет запрос на создание контракта на замену указанных батарей с указанием стоимости работ по замене батарей, заданный через `value` в виде целого числа - количества расчетных токенов. Параметр `car` позволяет ограничить ограничить доступ к контракту только для данного электромобиля. В терминал выводится адрес заключаемого контракта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/00360d2b.py firmware/ee4d92e6.py \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal: 0x3597bfD533a99c9aa083587B074434E61Eb0A258
```

Комментарий: В блокчейн сеть отправляется транзакция на создание контракта, в транзакцию передаются статусы обеих батарей, стоимость работ. Транзакция успешно включена в блок, в ходе ее обработки создается контракт на замену батарей, контракт можно проинспектировать на предмет идентификаторов батарей, участвующих в замене. Нельзя создать еще один контракт на замену, который бы содержал одну из использованных в данном контракте батарей. Результатом проверки того же самого статуса каждой батареи, что использовался в при создании контракта, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/00360d2b.py firmware/ee4d92e6.py  \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при отправке транзакции на создание контракта сделки определяется, что такой батареи не зарегистрировано в системе, то выдается ошибка. Контракт на замену батарей не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/00360d2b.py firmware/ee4d92e6.py  \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при попытке создания контракта сделки определяется, что батарея с данным идентификатором в текущий момент уже участвует в каком-то контракте замены, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при попытке создания контракта сделки определяется, что сервисный центр не является владельцем батареи, которую собирается использовать для замены, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для определения прав владения батареей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при попытке создания контракта сделки определяется, что аккаунт электромобиля, использующийся в контракте, не является текущим владельцем батареи, которую собирается использовать для замены, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0x2a0c0dbecc7e4d658f48e01e3fa353f44050c208 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что аккаунт отправитель транзакции не принадлежит сервисному центру, зарегистрированному в системе, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0x16332b24b07b2ee0ad9203587f9e5ffd0b30b851 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что несмотря на то, что аккаунт отправителя транзакции зарегистрирован как сервисный центр, ни один производитель батарей никогда не передавал ему права на владение батареями, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0x38c3c36677ed76d39efa17ab575e387818ec75d7 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что адрес, участвующий в сделке в качестве получателя новой батареи, не принадлежит электромобилю, зарегистрированному в системе, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0xdcfd11e4546551a09bc5c8529490347a9840162c 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что несмотря на то, что адрес, участвующий в сделке в качестве получателя новой батареи, зарегистрирован в системе как электромобиль, ни один производитель батарей никогда не передавал ему права на владение батареями, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

US-[3]USCounter Отправка запроса на отмену сделки

И сервисный центр может отменить сделку сделку после создания соответствующего контракта, но до того, как электромобиль послал запрос на согласие.

Использование скрипта
`scenter.py --canceldeal <contract>`
Подключается к узлу Ethereum и посылает транзакцию для отмены контракта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0xa650c24c4a9aadbb05e6047ca3598c93f5889be1` `Deal canceled successfully`
Комментарий: В блокчейн сеть отправляется транзакция на отмену контракта. Транзакция успешно включена в блок. `Battery Management Contract` можно проинспектировать, что владелец старой и новой батареи остались прежними. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей. Контракт сделки недоступен для проведения дальнейших операций

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0xa030017e8f8d12620ae4ad13066b0df905f91d30` `Cannot cancel the deal`
Комментарий: Если при попытке отмены сделки определяется, что электромобиль уже подтвердил согласие со сделкой, то выдается ошибка. Транзакция на отмену сделки не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0xa030017e8f8d12620ae4ad13066b0df905f91d30` `Cannot cancel the deal`
Комментарий: Если при попытке отмены сделки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена, то выдается ошибка. Транзакция на отмену сделки не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0x71819a042e604ba76ffd54cc1776f4fd3d18704f` `Cannot cancel the deal`
Комментарий: Если при попытке отмены сделки определяется, что аккаунт, который использовался для подписи отправленной транзакции, не соответствует адресу сервисного центра, зарегистрированного контракте сделки, то выдается ошибка. Транзакция на отмену сделки не отправляется в блокчейн.

US-[3]USCounter Получение информации, что контракт замены в ожидании подтверждения

Использование скрипта
`scenter.py --status <contract>`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения статуса контракта. В терминал выводится строка, обозначающая в какой фазе находится контракт.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --status 0x3597bfD533a99c9aa083587B074434E61Eb0A258` `Waiting for agreement`
Комментарий: Статус определился успешно. В блокчейн нет транзакций для контракта 0x3597bfD533a99c9aa083587B074434E61Eb0A258.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --status 0x87B074434E61Eb0A2583597bfD533a99c9aa0835` `Not found`
Комментарий: Статус не может определиться по причине того, что указанный адрес либо не является адресом контракта на замену батарей, либо был отменен, либо был завершен, либо вообще не существует в локальной копии блокчейн. В блокчейн нет транзакций для контракта 0x87B074434E61Eb0A2583597bfD533a99c9aa0835.

EPIC-[2]EpicCounter Принятие условий контракта на замену батарей

После создания контракта на замену батарей, электромобиль может ознакомиться с условиями контракта, где указывается:

- Какие батареи участвуют в замене: идентификатор и название производителя, количество циклов заряда каждой батареи.
- Количество расчетных токенов, определяющее компенсацию за разность амортизации батарей.
- Количество расчетных токенов, определяющих оплату выполнения работ по замене батареи.

Если условия контракта принимаются, электромобиль подтверждает согласие на замену перечислением необходимого количества токенов на счет контракта на замену батареи.

Компенсация за разность амортизации батарей определяется контрактом по следующим правилам:

Таблица 9.1: Определение компенсации при замене батарей

	$0 \leqslant N_{new} < 50$	$50 \leqslant N_{new} < 150$	$150 \leqslant N_{new} < 300$	$N_{new} > 300$
$0 \leqslant N_{old} < 50$	Δ	N/A	N/A	N/A
$50 \leqslant N_{old} < 150$	$\Delta \cdot 1,003^{\Delta}$	Δ	N/A	N/A
$150 \leqslant N_{old} < 300$	$\Delta \cdot 1,0035^{\Delta}$	$\Delta \cdot 1,003^{\Delta}$	Δ	N/A
$N_{old} > 300$	1000	1000	1000	0

где

- N_{old} - количество циклов заряда, полученных из статуса батареи, установленной в автомобиле;
- N_{new} - количество циклов заряда, полученных из статуса батареи, подготовленной в сервисном центре для замены;
- $\Delta = N_{old} - N_{new}$;
- N/A - поскольку подразумевается, что сервисный центр всегда предлагает такие батареи, чей количество циклов заряда выше или равно циклам заряда батареи из электромобиля, то другие сценарии не рассматриваются.

Итоговое количество токенов должно быть представлено числом, округленным посредством отбрасывания дробной части.

US-[3]USCounter Получение условий контракта

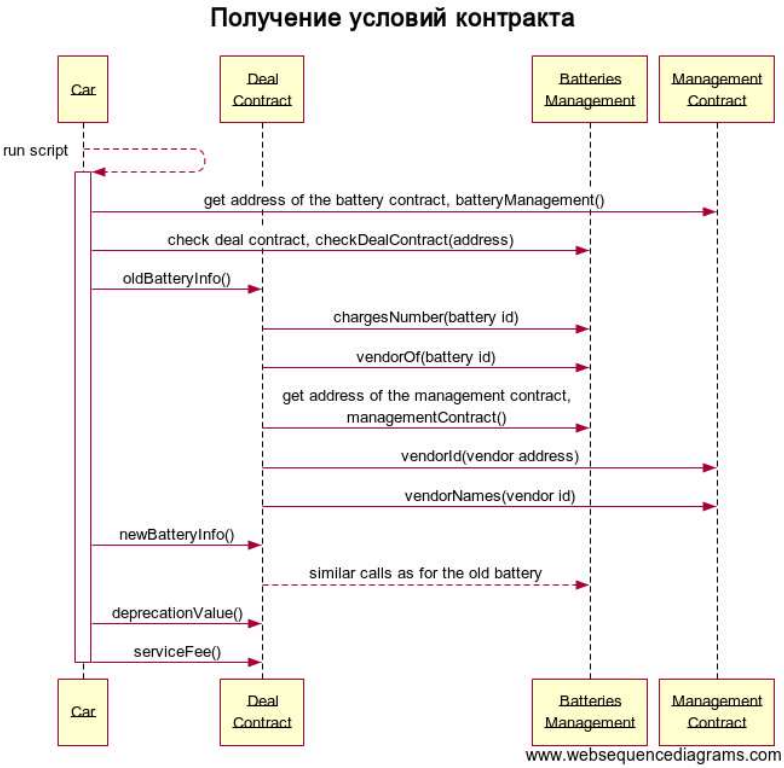


Рис. 9.5: Диаграмма получения информации о контракте

Использование скрипта
`car.py --info <contract>`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения данных о сделке из контракта. Данные о сделке выводятся в терминал.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# car.py --info 0x3597bfD533a99c9aa083587B074434E61Eb0A258`
`OLD BATTERY:`
`Total charges: 27`
`Vendor ID: ______`
`Vendor Name: 'Nissan Energy'`
`NEW BATTERY:`
`Total charges: 3`
`Vendor ID: ______`
`Vendor Name: 'Tesla Gigafactory'`
`WEAR COMPENSATION: 24`
`REPLACEMENT WORK: 100`

Комментарий: Информация о сделке выведена в терминал. Нет транзакций в блокчейн для получения информации о сделке.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --info 0x86fa049857e0209aa7d9e616f7eb3b3b78ecfdb0` `No deal registered.`
Комментарий: Если данная сделка не зарегистрирована в `Battery Management Contract`, то выдается ошибка.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --info 0x4092678e4e78230f46a1534c0fbc8fa39780892b` `Cannot check the deal.`
Комментарий: Если данная сделка была отменена или завершена, то выдается ошибка.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --info 0xaef53aa490e632bd4870296599efeec241fac96a` `Cannot check the deal.`
Комментарий: Если аккаунт автомобиля, выполняющий запрос, не является владельцем батаери, участвующей данной сделке, то выдается ошибка.

US-[3]USCounter Подтверждение согласия с условиями контракта

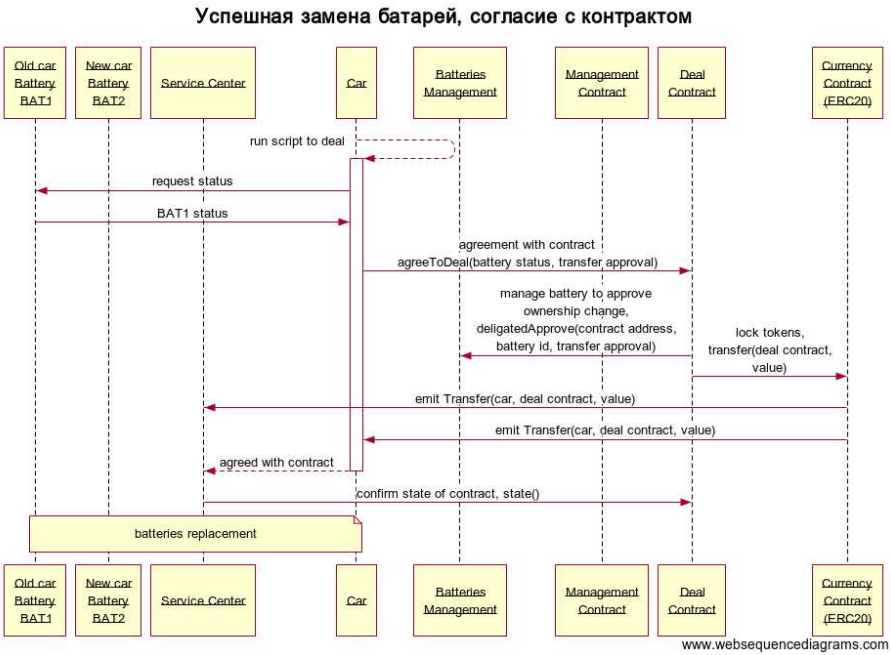


Рис. 9.6: Диаграмма подтверждения контракта

Использование скрипта
`car.py --deal <old battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для подтверждения согласия с условиями контракта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/ee4d92e6.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258 Confirmation of 124 tokens lock commited.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок. Контракт расчетных токенов может быть проинспектирован на изменение баланса контракта на замену батарей. Результатом проверки того самого же статуса батареи, что использовался в при согласии с условиями контракта, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. Нельзя создать еще один контракт на замену, который бы содержал одну из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/411b6845.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется отсутствие достаточного количества расчетных токенов на балансе электромобиля. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки баланса токенов перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/ee4d92e6.py 0x852A0bE16E3597bfD533a99c9aa083587B074434 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что аккаунт электромобиля, использующийся для подтверждения согласия с условиями контракта, не указан в качестве участника контракта (в качестве электромобиля). Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки участников контракта перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/11ef4074.py 0x852A0bE16E3597bfD533a99c9aa083587B074434 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что цифровая подпись статуса батареи, указывает на батарею, отличающуюся от того, что была использована для контракта. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/49b2523c.py 0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что временная метка, включенная в статус батареи, установленной в электромобиле, меньше либо равна той, что была использована для создания контракта. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/e45b2c31.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что батарея с данным идентификатором не принадлежит данному электромобилю. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/2c4fcd3f.py 0x80a7e048f37a50500351c204cb407766fa3bae7f Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

US-[3]USCounter Подтверждение готовности к физической замене батарей

Как только электромобиль подтвердил свое согласие с контрактом замены, сервисный центр должен иметь возможность получить информацию об этом, чтобы начать физическую замену батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py status 0x3597bfD533a99c9aa083587B074434E61Eb0A258 Battery replacement confirmed`
Комментарий: Статус определился успешно. В блокчейн нет транзакций для контракта 0x3597bfD533a99c9aa083587B074434E61Eb0A258.

EPIC-[2]EpicCounter Подтверждение выполнения условий контракта после замены батарей

После физической замены батарей автомобиль проверяет то, что новая батарея

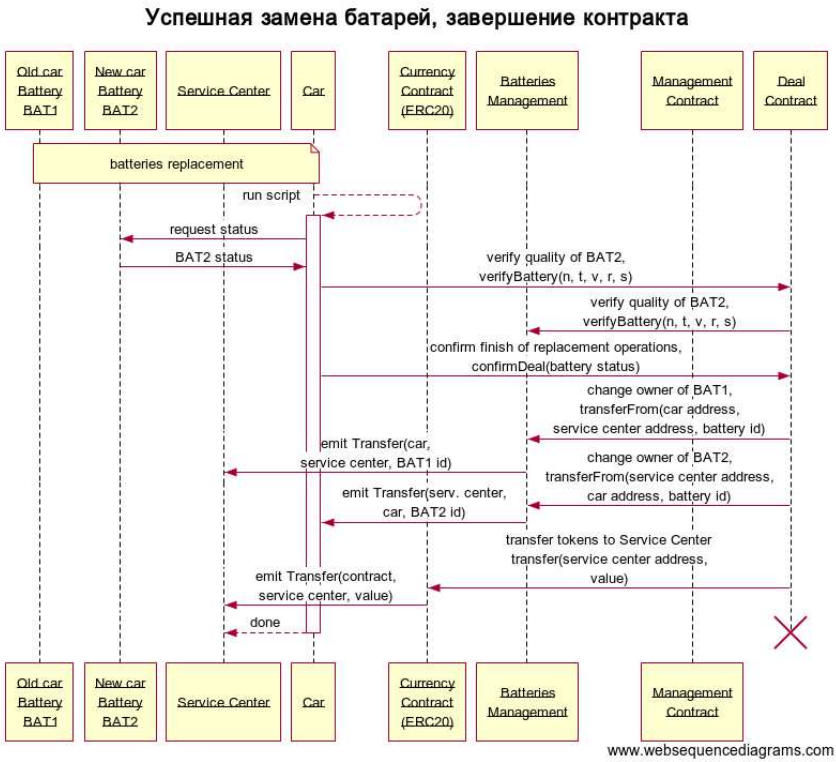


Рис. 9.7: Диаграмма завершения контракта

совпадает с той, что указана в контракте, и подтверждает завершение платежа к сервисному центру. Данное действие также меняет владельца батареи в блокчейн.

Использование скрипта
`car.py --confirm <new battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для завершения оплаты по контракту замены батарей.

US-[3]USCounter Подтверждение контракта

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/00360d2b.py 0x3597bfD533a99c9aa083587B074434
E61Eb0A258 124 tokens paid for battery replacement.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок. Контракт расчетных токенов может быть проинспектирован на увеличение баланса сервисного центра. `Battery Management Contract` можно проинспектировать, что владелец старой и новой батареи изменились. Результатом проверки того же самого статуса батареи, что использовался в при подтверждении контракта, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/d819ba81.py 0x3597bfD533a99c9aa083587B074434
E61Eb0A258 Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная батарея изначально не была указана для участия в замене. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/d819ba81.py 0x3597bfD533a99c9aa083587B074434
E61Eb0A258 Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что батарея с данным идентификатором и с таким же статусом уже участвовала в контракте замены. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/94018902.py 0x0d17628ea8a4f14c94142b18761a1b
26d42d1546 Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что временная метка, включенная в статус новой батареи, установленной в электромобиле, меньше либо равна той, что была использована для создания контракта. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --confirm firmware/d819ba81.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что аккаунт электромобиля, использующийся для подтверждения контракта не указан в качестве участника контракта (в качестве электромобиля). Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --confirm firmware/9cc3bf7a.py 0x6f7ded7500aff3be1015caa65c063bb3f764c581 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --confirm firmware/5444fbbc.py 0x32e90a83c861b9d81379ac13d063efe70e0dc327 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что электромобиль еще не послал подтверждение на согласие со сделкой. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Защита интересов участников сделки

Нельзя полагаться на то, что сделка будет всегда проходить по успешному сценарию. Если бы это было так, то и решение на платформе блокчейн было бы не нужно.

Следовательно, нужно предусмотреть такие варианты событий, когда участники по стечению обстоятельств (сбой системы питания электромобиля) или по злому умыслу (мошенничество) отходят от предписанного сценария.

EPIC-[2]EpicCounter Отказ сервисного центра в замене батареи

Возможна такая ситуация, когда электромобиль согласился со сделкой, но сервисный центр не производит замену. Это приводит к тому, что средства автомобиля (расчетные токены) блокируются на счету контракта сделки. Следовательно, нужно предоставить возможность электромобилю сделать запрос на их разблокирование.

Гарантией того, что замены не было, является новый статус батареи, которая все еще установлена в электромобиле. Чтобы указать, что замены произведено не было, нужно подтвердить контракту владение батарей через предъявление статуса, отличающегося от того, который использовался, чтобы подтвердить согласие с контрактом.

После разблокирования токенов, контракт считается отмененным.

При этом возможно мошенничество, когда автомобиль получает два статуса от батареи: один, чтобы подтвердить согласие со сделкой, и сразу же второй, чтобы даже при отсутствии старой батареи, можно было бы вернуть себе расчетные токены вместо завершения сделки. Следовательно, поскольку статус включает в себя временную метку, можно предусмотреть такую логику обработки запроса на разблокирование, когда между соседними статусами, пройдет не меньше определенного промежутка времени. Данное пороговое значение устанавливается в значение по-умолчанию (3600 секунд) при регистрации контракта `Battery Management Contract` и передается потом в контракт сделки при ее создании. `Battery Management Contract` должен предоставлять способ для изменения значения временного промежутка, в течение которого запросы на разблокирование токенов не будет применяться. Данное значение будет применяться только для вновь созданных сделок.

US-[3]USCounter Установка времени запрета разблокирование токенов

Использование скрипта
`setup.py --release <timeout>`
Подключается к узлу Ethereum и отправляет запрос к `Battery Management Contract` на изменение временного промежутка запрета разблокирования расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --release 600` `Updated successfully`
Комментарий: В блокчейн сеть отправляется транзакция. Транзакция успешно включена в блок. Сделки, созданные после этой транзакции, используют данный промежуток времени для разрешения разблокирования расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --release 600` `No permissions to change token release timeout`
Комментарий: Если попытка изменения временного промежутка запрета разблокирование токенов происходит не от аккаунта компании производителя ПО, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

US-[3]USCounter Запрос на разблокирование токенов

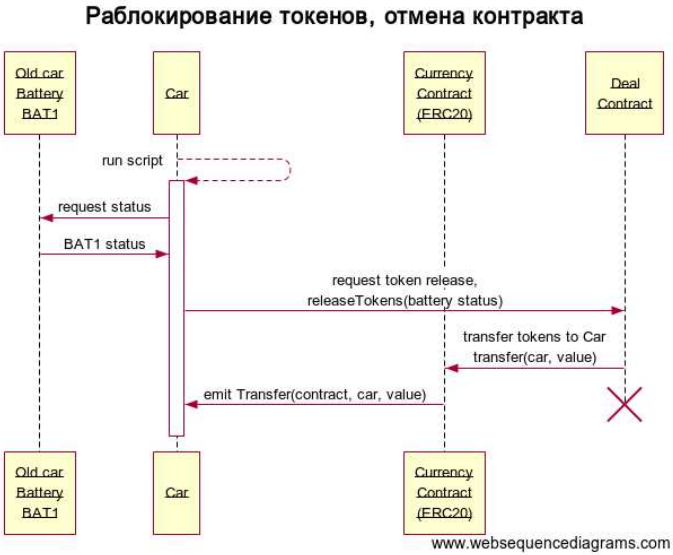


Рис. 9.8: Диаграмма отмены контракта

Использование скрипта
`car.py --release <old battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для разблокирование токенов. Выводит на экран количество разблокированных расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/10052a52.py 0x9d2d7931b0edf8709260e3f7ffce74 f2b3e569d6 124 tokens unlocked successfully.`
Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок. Контракт токенов может быть проинспектирован на увеличение баланса электромобиля. `Battery Management Contract` можно проинспектировать, что владелец старой и новой батареи не изменились. Результатом проверки того же самого статуса батареи, что использовался в при разблокировании токенов, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/2339a242.py 0x356dc7a098f26e720db68a7b1b0c62993585c504 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная батарея изначально не была указана для участия в замене. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/5c3cde51.py 0xe010407d8e423e7a374f6fc81136fa124e3b3d57 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что батарея с данным идентификатором и с таким же статусом уже участвовала в контракте замены. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/7db4ff5b.py 0x40d6ea55a436f814de4210d0de444fe52fef3946 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что временная метка, включенная в статус батареи, меньше, чем ожидаемая с учетом временного промежутка запрета разблокирования токенов. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/4998c293.py 0xca8d895baf2bf90810dc65507d2625c509b16c22 Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что аккаунт электромобиля, использующийся для разблокирования токенов, не указан в качестве участника контракта (в качестве электромобиля). Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/b3853f71.py 0xd43d5d0cc33f2832da514476441098abad1e241e Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

EPIC-[2]EpicCounter Разрешение споров после замены батарей

Следующая возможная ситуация, которая может возникнуть - это установка сервисным центром не той батареи, которая заявлялась при создании сделки, как неумышленно, так и злонамеренно. Причем это может и не оказаться правдой, когда злоумышленником выступает электромобиль, пытаясь сформировать отрицательную репутацию сервисного центра.

Тогда, после сообщения электромобилем о возникшей проблеме, возможны четыре возможных сценария:

1. Сервисный центр соглашается поставить верную батарею. Тогда сделка дальше движется по запланированному сценарию.
2. Сервисный центр признает свою ошибку и соглашается вернуть старую батарею. В этом случае замены не происходит, покрывать затраты сервисного центра не нужно. А электромобиль отправляет запрос на разблокирование токенов.
3. Электромобиль формирует атаку *Denial of Service* и блокирует работу сервисного центра, ложно сообщая о неправильной батарее (для этого годятся любые данные) и не отправляя запрос о завершении сделки. Один из возможных способов решения данной проблемы - введение страховых депозитов и системы скоринга.

   Тогда электромобиль за некорректное поведение получает снижение скоринга независимыми арбитрами (например, блокировка ведения бизнеса может рассматриваться полицией), что приводит к удорожанию страхового депозита электромобиля.

   Данный функционал не будет рассматриваться в рамках данной задачи.
4. Сервисный центр отказывается менять батарею, что может привести к тому, что из-за разряженной поставленной батареи, электромобиль не будет иметь возможность уехать из сервисного центра. Поскольку данная ситуация расценивается, как мошенничество со стороны сервисного центра, то к решению проблемы привлекаются независимые арбитры (например, полиция).

   Экономической защитой от такого поведения, может также служить система страховых депозитов при создании сделки и ведение скоринга всех сервисных центров. Снижение скоринга из-за неприемлемого поведения конкретного участника приводило бы к увеличению страхового депозита.

   Данный функционал не будет рассматриваться в рамках данной задачи.

US-[3]USCounter Отсутствие подтверждения замены батарей для завершения сделки

Еще одной ситуацией является отсутствие подтверждения согласия с заменой батарей от электромобиля для заверешения сделки.

Это приводит к тому, что на счет сервисного центра не перечисляются расчетные токены. Тогда, по аналогии с запросом на разблокирование токенов электромобилем, должен быть запрос на разблокирование токенов сервисным центром.

Разблокирование токенов сервисным центром также не должно позваляться в течение определенного промежутка времени. Поэтому для проверки времени запроса на разблокирование будет использоваться те же самые данные, что и для варианта с электромобилем.

После разблокирования токенов, контракт считается отмененным.

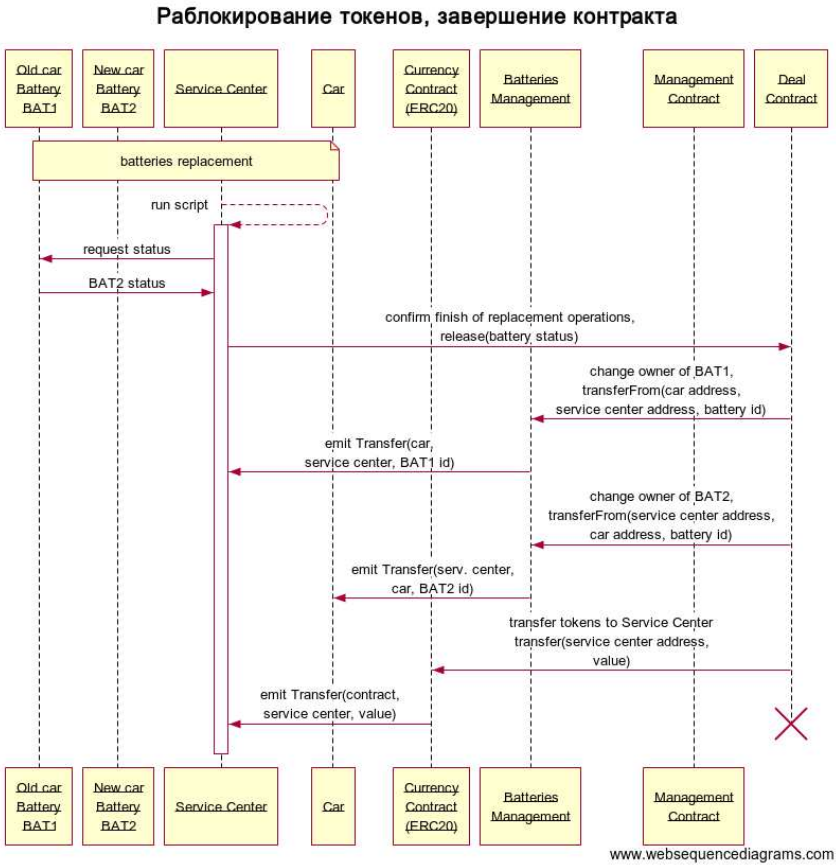


Рис. 9.9: Диаграмма разблокирования токенов и завершения контракта

Использование скрипта
`scenter.py --release <old battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для разблокирование токенов. Выводит на экран количество разблокированных расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/c412dbe6.py 0x9bb1fd0466a1a77714f7e0ce75c69740468ee56f 124 tokens unlocked successfully.`
Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок. Контракт токенов может быть проинспектирован на увеличение баланса сервисного центра. `Battery Management Contract` можно проинспектировать, что на изменение прав владения у старой и новой батарей. Результатом проверки того же самого статуса батареи, что использовался в при разблокировании токенов, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/2339a242.py 0x2e5990ecb06d4ef63400d7a4888db09c7a31d0b5 Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что данная батарея изначально не была указана для участия в замене, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/dca16490.py 0x9d2d7931b0edf8709260e3f7ffce74f2b3e569d6 Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что батарея с данным идентификатором и с таким же статусом уже участвовала в контракте замены, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/7db4ff5b.py 0xb1d1969e3932335eec67830d24268b0622ed7910 Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что временная метка, включенная в статус батареи, меньше, чем ожидаемая с учетом временного промежутка запрета разблокирования токенов, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/5fa8d151.py 0xca8d895baf2bf90810dc65507d2625c509b16c22 Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что аккаунт сервисного центра, использующийся для разблокирования токенов, не указан в качестве участника контракта (в роли сервисного центра), то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/7db4ff5b.py 0x40d6ea55a436f814de4210d0de444fe52fef3946 Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/4998c293.py 0xca8d895baf2bf90810dc65507d2625c509b16c22 Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что электромобиль еще не послал подтверждение на согласие со сделкой, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Ethereum контракты

Одно из требований к децентрализованному приложению - предоставлять ожидаемый интерфейс взаимодейтсвия. Это нужно потому, что клиентская часть приложения может быть реализована не обязательно в виде `Pyhton` скриптов. Она может быть выполнена, как приложение для десктопа, web-приложение или мобильное приложение.

Разработка всех частей может вестись независимо друг от друга в один и тот же момент, поэтому важно заранее описать интерфейсы взаимодействия и не менять их при дальнейшей разработке. Иначе неизбежны задержки в разработке ПО, срыв сроков, дополнительные расходы и поздний вывод продукта на рынок.

Ниже представлены задания, ответственные за реализацию отдельных методов Ethereum контрактов. Каждое задание будет считаться принятым, если совпадают все указанные в нем ожидания.

Для вызова методов перечисленных ниже контрактов будет использоваться ABI, который может быть получен из описания интерфейсов контрактов, представленных ниже

EPIC-[2]EpicCounter Management contract

Описание интерфейса контракта:

```
pragma solidity ^0.4.19;

contract ManagementContractInterface {
    // Для оповещения регистрации нового производителя
    // - адрес аккаунта из-под которого проходила регистрация
    // - идентификатор производителя
    event Vendor(address, bytes4);

    // Для оповещения о создании новой батареи
    // - идентификатор производителя
    // - идентификатор батареи
    event NewBattery(bytes4, bytes20);

    // Конструктор контракта
    // - адрес контракта, ответственного за накопление криптовалюты,
    //   перечисляемой в качестве депозита за использование сервиса.
    // - сумму сбора за выпуск одной батареи
    function ManagementContractInterface(address, uint256) public {}

    // Устанавливает адрес для контракта, ответственного за
    // управление информацией о батареях.
    // Доступен только создателю management контракта
    // - адрес контракта, управляющего инфорацией о батареях
    function setBatteryManagementContract(address) public;

    // Регистрирует вендора, если при вызове метода перечисляется
    // достаточно средств.
    // - наименование производителя
    function registerVendor(bytes) public payable;

    // Возвращает размер текущий депозит вендора
    function vendorDeposit(address) public view returns(uint256);

    // Регистрирует новые батареи, если при вызове метода на балансе
    // данного производителя достаточно средств. Во время регистрации
    // батарей баланс уменьшается соответственно количеству батареи и
    // цене, установленной для данного производителя на текущий момент.
    // - идентификаторы батарей
    function registerBatteries(bytes20[]) public payable;

    // Возвращает наименование производителя по его идентификатору
    function vendorNames(bytes4) public view returns(bytes);

    // Возвращает идентификатор производителя по адресу производителя
    function vendorId(address) public view returns(bytes4);

    // Возвращает адрес контракта, управляющего информацией о батареях,
    // установленного в данный момент.
    function batteryManagement() public view returns(address);

    // Устанавливает сумму сбора за выпуск одной батареи
    // - сумма сбора в wei
    function setFee(uint256) public;

    // Возвращает сумму сбора в wei, который будет списываться с депозита
    // за каждую выпущенную батарею. Если запрос идет от зарегистрированного
    // производителя батарей, то выдается та сумма сбора, которая была на момент
    // его регистрации.
    function batteryFee() public view returns(uint256);
```

```
    // Возвращает сумму сбора в wei, которую необходимо перечислить в качестве
    // депозита при регистрации производителя батарей. Сумма депозита
    // равна сумме за регистрацию 1000 батарей, что позволит защититься от
    // мошенников, поскольку требует серьезных вложений.
    function registrationDeposit() public view returns(uint256);

    // Возвращает адрес контракта, на котором происходит накопление
    // криптовалюты.
    function walletContract() public view returns(address);

    // Возвращает истину или ложь в зависимости от того, зарегистрирован сервис
    // центра с указанным адресом в системе или нет.
    function serviceCenters(address) public view returns(bool);

    // Регистрирует в системе адрес отправителя транзакции, как сервис центр.
    // Регистрация происходит только если данный адрес уже не был зарегистрирован
    // в качестве сервис центра или электромобиля.
    function registerServiceCenter() public;

    // Возвращает истину или ложь в зависимости от того, зарегистрирован
    // электромобиль с указанным адресом в системе или нет.
    function cars(address) public view returns(bool);

    // Регистрирует в системе адрес отправителя транзакции, электромобиль.
    // Регистрация происходит только если данный адрес уже не был зарегистрирован
    // в качестве сервис центра или электромобиля.
    function registerCar() public;
}
```

US-[3]USCounter метод setBatteryManagementContract()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции - аккаунт компании разработчика ПО.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `batteryManagement()` значения должны совпадать.

US-[3]USCounter метод registerVendor()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сумма транзакции отличная от нуля, если производитель батарей с таким названием еще не зарегистрирован, если с аккаунта отправителя транзакции не регистрировался еще не один производитель.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Vendor` с указанием адреса отправителя транзакции и идентификатора производителя

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Проверяется вызовом `vendorId()` и `vendorNames()`, которые позволяют получить адрес производителя по его идентификатору и название производителя по его адресу. Наименование производителя должно совпадать с тем, что было передано в метод `registerVendor()` при его вызове.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Депозит отправителя транзакции в `Management Contract` становится равным сумме транзакции. Результат можно проверить `vendorDeposit()`.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Изменяется баланс счета *Servide Provider Wallet* на сумму, указанную в транзакции

US-[3]USCounter метод registerBatteries()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если аккаунт отправитель уже зарегистрирован, как предводитель батарей, если депозит данного производителя достаточен для регистрации нужного количества батарей с учетом суммы транзакции.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
С депозита отправителя транзакции списывается сумма равная количеству батарей умноженному на сумму сбора за регистрацию одной батареи, которая была доступна на момент регистрации данного производителя. Результат можно проверить `vendorDeposit()`.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Для каждой батареи из переданного списка генерирует событие `NewBattery` с указанием идентификатора производителя и идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Каждый идентификатор батареи может провериться вызовом `ownerOf()`, `vendorOf()` в `Battery Management`, которые позволяют получить адрес владельца батареи и адрес ее производителя. Оба значения должны совпадать с адресом отправителя транзакции, вызвавшей `registerBatteries()`

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Для каждой зарегистрированной батареи можно сформировать вызовом скрипта, эмулирующего аппаратный ключ, такие данные, что `verifyBattery()` в `Battery Management` вернет 0.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Если в транзакции вызывающей метод была указана сумма транзакции, то баланс счета *Servide ProviderWallet* изменяется на эту сумму.

US-[3]USCounter метод setFee()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции - аккаунт компании разработчика ПО.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `batteryFee()` значения должны совпадать, при вызове из-под аккаунта, который не зарегистрирован в качестве производителя батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
При вызове из-под аккаунта, который зарегистрирован в качестве производителя батареи, то указывается сумма сборов, которая была установлена на момент регистрации производителя

US-[3]USCounter метод registerServiceCenter()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции не еще не зарегистрирован ни как сервисный центр, ни как элекромобиль.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `serviceCenters()` значения должны совпадать.

US-[3]USCounter метод registerCar()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции не еще не зарегистрирован ни как сервисный центр, ни как элекромобиль.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `cars()` значения должны совпадать.

EPIC-[2]EpicCounter Battery Management

Описание интерфейса контракта:

```
pragma solidity ^0.4.19;

contract BatteryManagementInterface {
    // Для оповещения о передаче прав владения батареей новому владельцу
    // - адрес предыдущего владельца
    // - адрес нового владельца
    // - индентификатор батареи
    event Transfer(address indexed, address indexed, bytes20);

    // Для оповещения, что какому-то аккунту делегировано право менять владельца
    // батареи
    // - владелец батареи
    // - адрес далегата
    // - идентификатор батареи, которой может распоряжаться делегат
    event Approval(address indexed, address indexed, bytes20);

    // Для оповещения после создания контракта на замену батарей
    // - адрес контракта
    event NewDeal(address);

    // Конструктор контракта
    // - адрес контракта, управляющего списком вендоров.
    // - адрес контракта, управляющего токенами, в которых будет
    //   происходить расчет за замену батарей.
    function BatteryManagementInterface(address, address) public {}

    // Создает новую батарею
    // Владельцем батареи назначается его текущий создатель.
    // Создание нового батареи может быть доступно только
    // management контракту
    // - адрес производителя батареи
    // - идентификатор батареи
```

```
function createBattery(address, bytes20) public;

// Меняет владельца батареи. Может быть вызвана только
// текущим владельцем
// - адрес нового владельца
// - идентификатор батареи
function transfer(address, bytes20) public;

// Меняет владельца для всех батарей перечисленных в списке. Может быть вызвана
// только текущим владельцем
// - адрес нового владельца
// - список идентификаторов батарей
function transfer(address, bytes20[]) public;

// Определяет по статусу батареи ее идентификатор, затем текущего владельца.
// И меняет владельца батаереи. Может быть вызвана только текущим владельцем
// - запакованные данные, хранящие статус батареи и нового получателя. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**124) + t*(2**192) + v*(2**160) + address
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function transfer(bytes32, bytes32, bytes32) public;

// Делегирует право аккунту изменять владельца батареи, которой на текущий
// момент владеет аккаунт-отправитель транзакции. Может выполняться, если
// аккаунт-отправитель тразанкции действительно владеет данной батареей. Для
// одной батареи может быть только один делегат. Для отзыва права необходимо
// послать 0 в качестве адреса.
// Генерирует событие Approval.
// - адрес делегата
// - идентификатор батареи
function approve(address, bytes20) public;

// Делегирует право аккунту изменять владельца батареи, которым на текущий
// момент владеет аккаунт, адрес которого определяется в ходе проверки цифровой
// подписи. Может выполняться, если аккаунт, восстанавливаемый из цифровой
// подписи, действительно владеет данной батареей. Для одной батареи может
// быть только один делегат. Для отзыва права необходимо
// послать 0 в качестве адреса.
// Цифровая подпись проверяется относительно адреса отправителя транзакции
// (msg.sender), поскольку подразумевается, что тем самым владелец батареи
// подтверждает, что делегирование права может быть выполнено с данного адреса.
// Генерирует событие Approval.
// - адрес делегата
// - идентификатор батареи
// - v, r, s компоненты цифровой подписи
function delegatedApprove(address, bytes20, uint8, bytes32, bytes32) public;

// Меняет владельца батареи. Может выполняться, только если отправителю
// транзакций делегировано право изменения владельца для данной батареи.
// Генерирует событие Transfer.
// - адрес владельца батареи
// - адрес получателя
// - идентификатор батареи
function transferFrom(address, address, bytes20) public;

// Возвращает адрес текущего зарегистрированного владельца батареи
// - индентификатор батареи
function ownerOf(bytes20) view public returns(address);
```

```
// Возвращает адрес производителя батареи
// - идентификатор батареи
function vendorOf(bytes20) view public returns(address);

// Проверяет цифровую подпись для данных и возвращает результат
// проверки и адрес вендора.
// Результат проверки: 0 - результат проверки цифровой подписи, показывает
// что она сделана зарегистрированной батареей; 1 - транзакция
// с таким статусом уже отправлялась в блокчейн, что указывает на возможную
// прослушку траффика; 2 - нет соответствующей зарегистрированной батареи;
// 999 - другая ошибка.
// - число зарядов
// - временная метка
// - v, r, s компоненты цифровой подписи
function verifyBattery(uint256 n, uint256 t, uint8 v, bytes32 r, bytes32 s)
    public view returns(uint256, address);

// Формирует новый контракт на выполнение операций по замене батарей.
// В контракт передаются идентификаторы батарей, участвующие в сделке,
// адрес аккаунта автомобиля, сумма на компенсацию амортизации батарей,
// сумма за выполенние работ по замене.
// Идентификаторы батарей становятся известны в ходе проверки цифровых
// подписей для статусов батарей.
// Сумма компенсации исчисляется в зависимости от количества заряда,
// переданного в статусе батарей.
// Контракт не должен создаваться, если для батареи с такими
// идентфикаторами не зарегистрированы в контракте, если одна из батарей
// уже участвует в каком-то другом контракте замены или участники контракта
// не являются владельцами батарей.
// - запакованные данные, хранящие статус сразу двух батарей. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n1*(2**160) + t1*(2**128) + v1*(2**96) + n2*(2**64) + t2*(2**32) + v2
// - r компонента подписи для данных старой батареи
// - s компонента подписи для данных старой батареи
// - r компонента подписи для данных новой батареи
// - s компонента подписи для данных новой батареи
// - адрес аккаунта автомобиля
// - количество расчетных токенов, определяющих выполнение работ по замене
function initiateDeal(uint256, bytes32, bytes32, bytes32, bytes32, address,
    uint256) public;

// Возвращает количество зарядов для конкретной батареи, зарегистрированных
// в данный момент
// - идентификатор батареи
function chargesNumber(bytes20) view public returns(uint256);

// Извлекает из переданных данных цифровую подпись текущего владельца батареи,
// адрес нового владельца батареи, ее статус. Изменяет владельца, если
// нет нарущений прав владения для восстановленного из подпсис адреса.
// - запакованные данные, хранящие статус батареи и нового получателя. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки (от старших
//   байт к младшим):
//    0..3   байт - число зарядов
//    4..7   байт - временная метка
//    8..75  байт - v, r, s компоненты цифровой подписи статуса батареи
//   76..95  байт - адрес нового владельца
//   96..163 байт - v, r, s компоненты цифровой подписи текущего владельца
```

```
    function approvedTransfer(bytes) public;

    // Проверяет переданный адрес, если он является контрактом сделки.
    // Возвращает 0 - контракт сделки существует и аккаунт, с которого
    // осуществляется запрос, является участником сделки; 1 - контракт сделки
    // существует, но аккаунт не является участников сделки; 2 - контракт
    // сделки не существует.
    function checkDealContract(address) view public returns(uint256);

    // Устанавливает верменной промежуток, после которого разрешено
    // разблокирование расчетных токенов контрактом сделки. Данная операция
    // разрешена только аккаунту производителя ПО.
    // - размер временного промежутка в секундах.
    function setTimeoutThreshold(uint256) public;

    // Возвращает временной промежуток, запрета разблокирования расчетных
    // токенов, установленный на текущий момент.
    function timeoutThreshold() view public returns(uint256);

    // Возвращает адрес контракта, который управляет регистрацией производителей
    // батарей.
    function managementContract() view public returns(address);

    // Возвращает адрес контракта, который управляет токенами, использующимися
    // в качестве внутренней валюты при совершении сделок по замене батарей.
    function erc20() view public returns(address);
}
```

US-[3]USCounter метод createBattery()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если вызывается из `Management Contract` и если батареи с таким идентификатором не существует еще.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если вызывается из `Management Contract`.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова провериться вызовом `ownerOf()`, `vendorOf()`, которые позволяют получить адрес владельца батареи и адрес ее производителя. Оба значения должны совпадать с адресом, указанным в вызовае `createBattery()`

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Для зарегистрированной батареи можно сформировать вызовом скрипта, эмулирующего аппаратный ключ, такие данные, что `verifyBattery()` вернет 0.

US-[3]USCounter метод transfer() с указанием одного идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции является владельцем батареи и если батарея с таким идентификатором существует

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если получатель права на владение батареей зарегистрирован, как электромобиль или сервисная станция

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Transfer` с указанием старого и нового владельцев и идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова провериться вызовом `ownerOf()`. Адрес возвращаемый методом должен совпадать с адресом указанным в `transfer()` для того же идентификатора батареи.

US-[3]USCounter метод delegatedApprove()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если адрес аккаунта, восстанавливаемый из цифровой подписи, указанной в параметрах, действительно владеет батарей с указанным идентификатором.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если адрес аккаунта, который вызывает данный метод, входит в то сообщение, для которого создана цифровая подпись.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Approval` с указанием владельца батареи, того кто может передавать права владения и идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова проверяется вызовом `transferFrom()` - аккаунт вызывающий данный метод, должен совпадать с аккаунтом, которому делегировалось право смены владельца.

US-[3]USCounter метод initiateDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции, является сервисным центром, и если он является владельцем одной из батарей, чей идентификатор восстанавливается из статуса

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если аккаунт, указанный в качестве параметра, является аккаунтом электромобиля и владельцем одной из батарей, чей идентификатор восстанавливается из статуса.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статусы обеих батарей еще не использовались при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если батареи с такими идентификаторами не участвуют ни в какой другой сделке.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `NewDeal` с указанием адреса контракта сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Approval` с указанием адреса сервисного центра, адреса контракта сделки и идентификатором батареи, которую сервисный центр предлагает к замене.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статусы, использовавшиеся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `chargesNumber()` с идентификатором каждой из батарей. Должно вернуться тоже значение выполненных зарядов, что и в статусе, который использовался для создания сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `oldBattery()` или `newBattery()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны вернуть те же идентификаторы батарей, что получатся при восстановлении из статусов, использовавшихся при создании сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `oldBatteryInfo()` или `newBatteryInfo()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны вернуть верную информацию о производителях батарей, участвующих в сделке.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `deprecationValue()` и `serviceFee()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны вернуть верную информацию о количестве токенов, необходимых для сделки (сумма компенсации при замене батарей должна соответствовать значению рассчитанному по Таблице 1.1.).

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `agreeToDeal()` или `cancelDeal()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны успешно выполняться (при соблюдении остальных условий их успешного выполнения).

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовашие в контракте, и послав их в другой вызов `initiateDeal()`, который завершится с ошибкой из-за того, что батареи уже используются в другой сделке.

US-[3]USCounter метод setTimeoutThreshold()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции - аккаунт компании разработчика ПО.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `timeoutThreshold()` значения должны совпадать.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `releaseTokensByCar()` или `releaseTokensByServiceCenter()` у контрактов сделки, созданных после установки нового временного промежутка. Разблокирование токенов должно происходить только после истечения установленного промежутка времени.

EPIC-[2]EpicCounter Deal contract

Описание интерфейса контракта:

```
pragma solidity ^0.4.19;

contract DealInterface {

    // Конструктор контракта
    // - идентификатор старой батареи
    // - идентификатор новой батареи
    // - адрес контракта расчетных токенов
    // - количество токенов, определяющих компенсацию амортизации батарей
    // - количество токенов, определяющих выполнение работ по замене
    // - временной промежуток, в ходе которого запрещено разблокирование
    //   расчетных токенов.
    function Deal(bytes20, bytes20, address, uint256, uint256, uint256) public {
    }

    // Возвращают идентификаторы батарей, вовлеченных в контракт.
    function oldBattery() view public returns(bytes20);
    function newBattery() view public returns(bytes20);

    // Возвращают информацию о батареях, вовлеченных в контракт.
    // Возвращаемые значения:
    // - количество зарядов
    // - идентификатор производителя
    // - наименование производителя
    function oldBatteryInfo() view public returns(uint256, bytes4, bytes);
    function newBatteryInfo() view public returns(uint256, bytes4, bytes);

    // Возвращает количество токенов, определяющих компенсацию амортизации
    // батарей
    function deprecationValue() view public returns(uint256);

    // Возвращает количество токенов, определяющих выполнение работ по
    // замене
    function serviceFee() view public returns(uint256);

    // Возвращает текущий статус контракта
    // 1 - Ожидание согласия со сделкой от автомобиля
    // 2 - Согласие со сделкой получено
    // 3 - Оплата работ выполнена
    function state() view public returns(uint256);

    // Проверяет, что переданный статус батареи получен от батареи, которая
    // зарегистрирована к замене (старая), что данный запрос идет от текущего
```

```
// владельца батареи, который участвует в контракте в роли получателя новой
// батареи. Если все условия выполняются, то токены идущие в оплату
// контракта переводятся на счет контракта.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function agreeToDeal(uint256, bytes32, bytes32) public;

// Проверяет, что переданный статус батареи получен от батареи, на которую
// должна происходить замена. При этом:
// 1) либо данный запрос идет от аккаунта, который участвует в контракте в роли
//    получателя новой батареи
// 2) либо запрос идет от аккаунта, который участвует в контракте в роли того,
//    кто предоставляет сервис по замене и с момента запроса согласия с
//    контрактом  прошло 24 часа
// Если все условия выполняются, то токены идущие в оплату контракта
// переводятся на счет сервисного центра, владельцы батарей изменяются.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function confirmDeal(uint256, bytes32, bytes32) public;

// Перепосылает аргументы в контракт BatteryManagement для проверки цифровой
// подписи. Возвращаемый результат проверки такой же, как и в оригинальной
// функции: 0 - результат проверки цифровой подписи, показывает
// что она сделана батареей, для которой существует токен; 1 - транзакция
// с таким статусом уже отправлялась в блокчейн, что указывает на возможную
// прослушку траффика; 2 - для  батареи нет соответствующего токена; 999 -
// другая ошибка.
// - число зарядов
// - временная метка
// - v, r, s компоненты цифровой подписи
function verifyBattery(uint256 n, uint256 t, uint8 v, bytes32 r, bytes32 s)
    public view returns(uint256, address);

// Обрабатывает запрос на разблокирование расчетных токенов и возвращает
// токены на счет электромобиля, владельцы батарей, участвующих в сделке,
// не изменяются.
// Выполняется только если отправитель запроса, зарегистрирован в качестве
// получателя новой батареи, если проверка подписи из статуса батареи указывает
// на то, что батарея не была заменена, и если время из статуса больше,
// чем разрешенное время разблокировки токенов.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function releaseTokensByCar(uint256, bytes32, bytes32) public;

// Обрабатывает запрос на разблокирование расчетных токенов и перечисляет
// токены на счет сервисного центра, происходит смена владельцев батарей,
// участвующих в сделке.
// Выполняется только если отправитель запроса, зарегистрирован в качестве
```

```
    // изначального владельца новой батареи, если проверка подписи из статуса
    // батареи указывает на то, что батарея действительно была заменена, и
    // если время из статуса больше, чем разрешенное время разблокировки токенов.
    // - запакованные данные, хранящие статус батареи. Упаковка
    //   данных используется для уменьшения используемого газа за данные,
    //   передаваемые в транзакции в метод контракта. Метод упаковки:
    //   p = n*(2**64) + t*(2**32) + v
    // - r компонента подписи для данных батареи
    // - s компонента подписи для данных батареи
    function releaseTokensByServiceCenter(uint256, bytes32, bytes32) public;

    // Отменяет сделку, если она получатель новой батареи, не согласился с
    // контрактом. Выполняется только если отправитель запроса - сервисный
    // центр.
    function cancelDeal() public;
}
```

US-[3]USCounter метод agreeToDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является электромобилем, и если он является владельцем батареи, чей идентификатор восстанавливается из статуса, использованного в транзакции

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является участником контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции имеет достаточное количество расчетных токенов на балансе.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Approval` с указанием адреса сервисного центра, адреса контракта сделки и идентификатором батареи, которую сервисный центр предлагает к замене.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Transfer` с указанием адресов электромобиля и контракта сделки, и тем же количеством токенов, что получаются в ходе сложения значений, которые выдаются `deprecationValue()` и `serviceFee()`, вызванные у данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если вызов `state()` вернет 2.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса контракта сделки в `Currency Token` контракте до и после транзакции с `agreeToDeal()`. Разница между значениями равна сумме сделки.

US-[3]USCounter метод confirmDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является электромобилем и он является участников контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи, восстановленный из статуса, использующегося в транзакции, указывает на то, что владельцем этой батареи является сервисный центр - участник данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки выполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод confirmDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется три события `Transfer` • два - с указанием адресов сервисного центра, электромобиля и идентификаторами батарей; • одно - с указанием контракта сделки, контракта сервисного центра, и суммы расчетных токенов, установленных в контракте на начало сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса сервисного центра сделки в `Currency Token` контракте до и после транзакции с `agreeToDeal()`. Разница между значениями равна сумме сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `ownerOf()` для идентификаторов батарей в `Battery Management` контракта до и после транзакции, вызывающей `agreeToDeal()`. Должен произойти обмен владельцев.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

US-[3]USCounter метод releaseTokensByCar()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является электромобилем, и если он является владельцем батареи, чей идентификатор восстанавливается из статуса, использованного в транзакции

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является участником контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта, и больше, чем временной промежуток в ходе которого запрещено разблокирование батарей.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента методы `confirmDeal()` и `cancelDeal` для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Transfer` с указанием адресов контракта сделки и электромобиля, и тем же количеством токенов, что получаются в ходе сложения значений, которые выдаются `deprecationValue()` и `serviceFee()`, вызванные у данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса электромобиля в `Currency Token` контракте до и после транзакции с `releaseTokensByCar()`. Разница между значениями равна сумме сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

US-[3]USCounter метод releaseTokensByServiceCenter()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является сервисным центром и он является участников контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи, восстановленный из статуса, использующегося в транзакции, указывает на то, что владельцем этой батареи является электромобиль - участник данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта, и больше, чем временной промежуток в ходе которого запрещено разблокирование батарей.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки выполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод confirmDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется три события `Transfer` • два - с указанием адресов сервисного центра, электромобиля и идентификаторами батарей; • одно - с указанием контракта сделки, контракта сервисного центра, и суммы расчетных токенов, установленных в контракте на начало сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса сервисного центра сделки в `Currency Token` контракте до и после транзакции с `releaseTokensByServiceCenter()`. Разница между значениями равна сумме сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `ownerOf()` для идентификаторов батарей в `Battery Management` контракта до и после транзакции, вызывающей `releaseTokensByServiceCenter()`. Должен произойти обмен владельцев.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

US-[3]USCounter метод cancelDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является сервисным центром и он является участников контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки выполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод confirmDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

Обмен батарей без участия сервисного центра

Для того, чтобы не ограничивать возможные сценарии обмена батареями только между сервисным центром и электромобилем, необходимо обеспечивать передачу батареи от одного владельца к другому (например, сценарий когда автовладелец продет батарею другому владельцу).

При этом должен поддерживаться вариант, когда подключение к блокчейн сети доступно и сторона, получающая батарею, может убедиться в передаче прав владения, так и вариант, когда подключение к блокчейн невозможно, т.е. передача прав владения может быть отложенная во времени.

В во втором случае, данная сделка не является рекомендованной, поскольку несёт очень большие риски, связанные с мошенничеством. Возможен такой вариант, когда предыдущий владелец создает фальшивую батарею и программирует ее на выдачу нескольких статусов, считанных с оригинальной батареи. Т.е. статусы с фальшивой батареи будут выглядеть так, как будто они подписаны реальной цифровой подписью и выдаются зарегистрированной в блокчейн батереей.

US-[3]USCounter Передача прав на батарею новому владельцу

Данный сценарий используется при доступе к блокчейн во время передачи прав владения. При этом новый владелец может убедиться, в том что передаваемая батарея оригинальная через получение и проверки статуса батареи.

Использование скрипта
`car.py owner <path to battery> <new_owner>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и отправляет запрос к `Battery Management Contract` на смену владельца батареи. В терминал выводится статус выполнения команды.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py owner firmware/00360d2b.py 0x50dfbef376d01e05e134a9b3a322afef50111ec1 Success`
Комментарий: В блокчейн сеть отправляется транзакция на изменение владельца батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py owner firmware/d819ba81.py 0x0dcd2f752394c41875e259e00bb44fd505297caf Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи выдается ошибка, поскольку данная батарея не принадлежит тому, кто пытается сменить владельца. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py owner firmware/d819ba81.py 0x0dcd2f752394c41875e259e00bb44fd505297caf Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи может выдается ошибка, поскольку данная батарея не зарегистрирована в блокчейн. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

EPIC-[2]EpicCounter Замена батарей при отсутствии подключения к сети блокчейн

US-[3]USCounter Отложенная передача прав владения

Использование скрипта
`car.py offline <path to battery> <new_owner>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подписывает приватным ключем текущего владельца батареи статус (включая цифровую подпись) и адрес нового владельца. В терминал выводится в шестнадцатеричном формате доверенность, включающая статус батареи, адрес нового владельца, подпись текущего владельца.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py offline firmware/00360d2b.py 0x50dfbef376d01e05e134a9b3a322afef50111ec10c164f.....8e51bc`
Комментарий: На экран выведена доверенность - строка из шестнадцатеричных символов, которая потом может использоваться для отложенного подтверждения владения.

US-[3]USCounter Регистрация прав владения

Использование скрипта
`car.py register <certificate>`
Подключается к узлу Ethereum и отправляет данные доверенности к `Battery Management Contract` на передачу прав владения адресу, указанному в доверенности. В терминал выводится статус выполнения команды.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py register 0c164f.....8e51bc` `Success`
Комментарий: В блокчейн сеть отправляется транзакция на изменение владельца батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py register 0c164f.....8e51bc` `Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи может выдается ошибка, поскольку данная батарея не принадлежит тому, кто пытается сменить владельца. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py register 0c164f.....8e51bc` `Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи может выдается ошибка, поскольку данная батарея не зарегистрирована в блокчейн. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

Оптимизация системы

Поскольку вычислительные мощности, затрачиваемые валидаторами блоков, на включение транзакций блок так или иначе оплачивают пользователи системы, то, очевидно, что решение, предоставляемое командой разработчика ПО должно быть конкурентным не только по функционалу, но и по стоимости пользования.

US-[3]USCounter Оптимизация по использованию вычислительных ресурсов

Чем оптимальнее логика работы контрактов, тем меньше будут платить пользователи за вызов того или иного функционала децентрализованного приложения.

Поэтому необходимо реализовывать методы контрактов таким образом, чтобы потреблять минимально возможное количества газа.

Для проверки решения данной части задания, будет проверяться, что методы, указанные ниже в таблице потребляют не больше указанного количества газа. При этом каждый метод засчитывается отдельно, т.е. допускается частичное выполнения задания. Метод будет проверяться только если успешно проверено задание (или

конкретная часть его), указанное в последнем столбце.

AC	method	gas, max	depends on
AC-[3]USCounter-01	`Management Contract:registerVendor`	—	
AC-[3]USCounter-02	`Battery Management:createBattery`	—	
AC-[3]USCounter-03	`Battery Management:transfer(address, bytes20)`	—	
AC-[3]USCounter-04	`Battery Management:transfer(bytes32, bytes32, bytes32)`	—	
AC-[3]USCounter-05	`Battery Management:delegatedApprove`	—	
AC-[3]USCounter-07	`Deal:agreeToDeal`	—	
AC-[3]USCounter-08	`Deal:confirmDeal`	—	
AC-[3]USCounter-09	`Deal:cancelDeal`	—	
AC-[3]USCounter-10	`Deal:releaseTokensByCar`	—	
AC-[3]USCounter-11	`Deal:releaseTokensByServiceCenter`	—	

US-[3]USCounter Оптимизация комиссии за валидацию транзакций

Скорость валидации транзакций и включения в блок влияет на удобство пользования системой (время присутствия электромобиля на сервисном центре для замены батареи должно быть сильно меньше, чем при зарядке этой батареи). Высокое вознаграждение валидатору транзакции, естественно ускорит включению ее в блок. Но с другой стороны, высокая стоимость оттолкнет пользователей системы. Следовательно, нужно предлагать пользователю использовать такой размер вознаграждения, который был бы оптимальным по скорости проведения транзакций и не приводил бы к явной переплате.

Для получения оптимальной стоимости газа предлагается использовать оракул `https://github.com/poanetwork/gasprice` доступный по адресу `https://gasprice.poa.network/`.

Для проверки решения данной части задания, будет проверяться, для транзакций, изменяющих состояние блокчейн, скрипты `setup.py`, `vendor.py`, `scenter.py` и `car.py` выставляет такую стоимость газа, которая по мнению оракула приводит к вероятности в 90 процентов включения транзакции в следующий блок. Задание считается выполненным, если каждый скрипт выставляет стоимость газа согласно требованиям выше.

Анализ использования системы

Поскольку одним из инструментов привлечения новых участников системы является демонстрация ее популярности, то вместе с децентрализованным приложением должен предоставляться инструмент сбора статистики.

Имя скрипта
`stat.py [command]`

US-[3]USCounter Сбор статистики

Использование скрипта
`stat.py [--tokens]`
Подключается к узлу Ethereum и, по адресу `Management Contract` из `database.json` определяет первый блок, в котором была регистрация контракта, после чего блокчейн исследуется на предмет нужных данных. В терминал выводится статистика по использованию системы.

Пример вызова команды (AC-058-01)

`# stat.py`
`Registered vendors: aaa`
`Registered batteries: bbb`
`Vendor and batteries income: ccc ether`
`Registered deals: ddd`
`Batteries transfers: eee`
`Canceled deals: fff`

Комментарий: Собрана статистика по использованию системы, где • `Registered vendors` - количество зарегистрированных вендоров; • `Registered batteries` - общее количество зарегистрированных батарей; • `Vendor registration income` - средства полученные от регистрации вендоров и принятые в качестве дополнительной оплаты за регистрацию батарей; • `Registered deals` - общее количество зарегистрированных сделок; • `Batteries transfers` - общее количество операций по изменению владельца батареи; • `Canceled deals` - общее количество сделок, которые были отменены и не привели к оплате за заменую батарей.

Пример вызова команды (AC-058-02)

`# stat.py --tokens`
`Registered vendors: aaa`
`Registered batteries: bbb`
`Vendor and batteries income: ccc ether`
`Registered deals: ddd`
`Batteries transfers: eee`
`Canceled deals: fff`
`Token flows:`
`* XXX tokens used in deals`
`* YYY tokens paid for service`
`* ZZZ tokens locked on deal contracts`

Комментарий: Собрана статистика по использованию системы, где • `tokens used in deals` - количество расчетных токенов, которые были пересланы на счета сделок; • `tokens paid for service` - количество расчетных токенов, переведенных за совершенные замены батарей; • `tokens locked on deal contracts` - количество расчетных токенов, заблокированных в текущий момент на незавершенных сделках.

10.5. Решение

./battery_ template.py

```
import sys, os, datetime as dt
from py_ecc.secp256k1 import ecdsa_raw_sign
from json import load, dump, dumps
from sha3 import keccak_256
```

```
import argparse

def _getDatabaseName(_args):
    return os.path.splitext(_args)[0] + ".json"

def _writeDataBase(_data, _file):
    with open(_file, 'w') as out:
        dump(_data, out)

def _openDataBase(_file):
    if os.path.exists(_file):
        with open(_file) as file:
            return load(file)
    else:
        return None

def _zpad(_x, _l, _s):
    return _s * (_l - len(_x)) + _x

def _getBatteryStatus(_file):
    file = _getDatabaseName(_file)

    databaseName = _getDatabaseName(file)
    data = _openDataBase(databaseName)
    if data is None:
        n = 0
    else:
        n = data['n']

    t = int(dt.datetime.utcnow().timestamp())
    priv = int(_privateKey, 16).to_bytes(32, 'big')
    m = n * 2 ** 32 + t

    message = m.to_bytes(32, byteorder='big')
    data = keccak_256(message).digest()
    batteryId = ecdsa_raw_sign(data, priv)

    v = batteryId[0]
    r = _zpad(hex(batteryId[1])[2:], 64, '0')
    s = _zpad(hex(batteryId[2])[2:], 64, '0')

    return n, t, v, r, s

def get(_file):
    n, t, v, r, s = _getBatteryStatus(_file)
    print(n, t, v, r, s, sep = '\n')

def charge(_file):
    databaseName = _getDatabaseName(_file)
    data = _openDataBase(databaseName)
    if data is None:
        data = {'n': 0}

    data['n'] = data['n'] + 1

    _writeDataBase(data, databaseName)

def getJson(_file):
    n, t, v, r, s = _getBatteryStatus(_file)
    data = {'n': n, 't': t, 'v': v, 'r': r, 's':s}
```

```
    print(dumps(data))

_privateKey = 'cafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafe'

def create_parser():
    parser = argparse.ArgumentParser(
        description='Service provider tool',
        epilog="""
    It is expected that Web3 provider specified by WEB3_PROVIDER_URI
    environment variable. E.g.
    WEB3_PROVIDER_URI=file:///path/to/node/rpc-json/file.ipc
    WEB3_PROVIDER_URI=http://192.168.1.2:8545
    """
    )

    parser.add_argument(
        '--get', action='store_true', required=False,
        help='Return battery status'
    )

    parser.add_argument(
        '--charge', action='store_true', required=False,
        help='Increase counter of charges'
    )

    parser.add_argument(
        '--getJson', action='store_true', required=False,
        help='Return battery status in JSON format'
    )

    return parser

def main():
    parser = create_parser()
    args = parser.parse_args()

    if args.get:
        get(sys.argv[0])
    elif args.charge:
        charge(sys.argv[0])
    elif args.getJson:
        getJson(sys.argv[0])
    else:
        sys.exit("No parameters provided")

if __name__ == '__main__':
```

./car.py

```
import warnings

warnings.simplefilter("ignore", category=DeprecationWarning)

# install web3 >=4.0.0 by `pip install --upgrade 'web3==4.0.0b11'`
from web3 import Web3
from web3.middleware import geth_poa_middleware
from web3.utils.transactions import wait_for_transaction_receipt

import datetime as dt
from random import randint
```

```python
import argparse
import sys, os

from utils import writeDataBase, openDataBase, getActualGasPrice, compileContracts, \
    initializeContractFactory, runCmd, ManagementContract, BatteryContract, DealContract

from verify_battery import verify, return_ntvrs_from, batteryExists

web3 = Web3()

# configure provider to work with PoA chains
web3.middleware_stack.inject(geth_poa_middleware, layer=0)

accountDatabaseName = 'car.json'
mgmtContractDatabaseName = 'database.json'

gasPrice = getActualGasPrice(web3)

registrationRequiredGas = 50000


def _createCarAccountDatabase(_carPrivateKey):
    data = {'key': _carPrivateKey}
    writeDataBase(data, accountDatabaseName)


# generate private key using current time and random int
def _generatePrivateKey():
    t = int(dt.datetime.utcnow().timestamp())
    k = randint(0, 2 ** 16)
    privateKey = web3.toHex(web3.sha3((t + k).to_bytes(32, 'big')))
    return _delHexPrefix(privateKey)


# delete hex prefix from string
def _delHexPrefix(_s: str):
    if _s[:2] == '0x':
        _s = _s[2:]
    return _s


def new():
    privateKey = _generatePrivateKey()
    data = {"key": privateKey}
    writeDataBase(data, accountDatabaseName)
    print(web3.eth.account.privateKeyToAccount(data['key']).address)


def account():
    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    privKey = data['key']
    print(web3.eth.account.privateKeyToAccount(privKey).address)


def reg():
```

```
    mgmtContract = ManagementContract(web3)

    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    privateKey = data['key']
    carAddress = web3.eth.account.privateKeyToAccount(privateKey).address

    if registrationRequiredGas * gasPrice > web3.eth.getBalance(carAddress):
        print("No enough funds to send transaction")
        return

    nonce = web3.eth.getTransactionCount(carAddress)
    tx = {'gasPrice': gasPrice, 'nonce': nonce}

    regTx = mgmtContract.functions.registerCar().buildTransaction(tx)
    signTx = web3.eth.account.signTransaction(regTx, privateKey)
    txHash = web3.eth.sendRawTransaction(signTx.rawTransaction)
    receipt = wait_for_transaction_receipt(web3, txHash)

    if receipt.status == 1:
        print('Registered successfully')
    else:
        print('Already registered')


CarExe = 'car.py'


def AgreeToDeal(_actor, _path, _addressDeal):
    retval = False
    if not os.path.exists(_path):
        print('Not found path ')
        return retval

    mgmtContract = ManagementContract(web3)
    _addressBat = mgmtContract.functions.batteryManagement.call()
    BatContract = BatteryContract(web3, _addressBat)
    dealCont = DealContract(web3, _addressDeal)
    if not batteryExists(_path):
        print('Cannot access battery')
        return
    _n, _t, _v, _r, _s = return_ntvrs_from(_path)
    _tokenId = BatContract.functions.getBatteryIdFromSignature(_n, _t, _v, _r, _s)
    txHash1 = BatContract.functions.delegatedApprove(_addressDeal, _tokenId, _v, _r, _s)
    _p = _n * (2 ** 64) + _t * (2 ** 32) + _v
    txHash2 = dealCont.functions.agreeToDeal(_p, _r, _s)
    receipt1 = wait_for_transaction_receipt(web3, txHash1)
    receipt2 = wait_for_transaction_receipt(web3, txHash2)
    if receipt1.status == 1:
        if receipt2.status == 1:
            retval = True
        else:
            print("AgreeToDeal from Deal contract did not work")
    else:
        print("getBatteryIdFromSignature from Battery contract did not work")
    return retval
```

```
def GetInfoAboutContract(_actor, _addressDeal):
    dealCont = DealContract(web3, _addressDeal)
    forma = """Total charges: {}\nVendor ID: {}\nVendor Name: '{}'"""
    try:
        old_bat_info = dealCont.functions.oldBatteryInfo().call()
        new_bat_info = dealCont.functions.oldBatteryInfo().call()
    except BaseException as error:
        return False
    print("OLD BATTERY:")
    print(forma.format(old_bat_info))
    print("NEW BATTERY:")
    print(forma.format(new_bat_info))
    return True


def create_parser():
    parser = argparse.ArgumentParser(
        description='Service provider tool',
        epilog="""
    It is expected that Web3 provider specified by WEB3_PROVIDER_URI
    environment variable. E.g.
    WEB3_PROVIDER_URI=file:///path/to/node/rpc-json/file.ipc
    WEB3_PROVIDER_URI=http://192.168.1.2:8545
    """
    )

    parser.add_argument(
        '--new', action='store_true', required=False,
        help='Add new car account'
    )

    parser.add_argument(
        '--account', action='store_true', required=False,
        help='Return address of car'
    )

    parser.add_argument(
        '--reg', action='store_true', required=False,
        help='Register car in the chain'
    )

    parser.add_argument(
        '--verify', type=str, required=False,
        help='Verify battery on validity'
    )

    parser.add_argument(
        '--deal', nargs=2, required=False,
        help='Confirm deal contract'
    )
    parser.add_argument(
        '--info', type=str, required=False,
        help='Show information about contract for change batteries'
    )
    return parser


def main():
    parser = create_parser()
```

```
    args = parser.parse_args()

    web3.eth.enable_unaudited_features()

    if args.new:
        new()
    elif args.account:
        account()
    elif args.reg:
        reg()
    elif args.verify:
        if batteryExists(args.verify):
            result = verify(web3, args.verify)
            if result[0] == 0:
                n = result[1]
                vendorId = result[2]
                vendorName = result[3]
                print("Verified successfully.")
                print('Total charges: {}'.format(n))
                print('Vendor ID: {}'.format(vendorId))
                print('Vendor Name: {}'.format(vendorName))
            elif result[0] == 1:
                print('Battery with the same status already replaced. Probably the battery
            forged.')
                elif result[0] == 2:
                print('Verification failed. Probably the battery forged.')
            else:
                print('Error. Cannot verify battery.')
        else:
            print('Cannot access battery')
    elif args.info:
        pass
    else:
        sys.exit("No parameters provided")


if __name__ == '__main__':
    main()
```

./contracts/BatteryManagement.sol

```
pragma solidity ^0.4.19;

import "./NFT/BasicNFToken.sol";
import "./ManagementContractInterface.sol";
import "./ERC20TokenInterface.sol";
import "./lib/Ownable.sol";
import "./lib/SafeMath.sol";
import "./Fee.sol";
import "./Deal.sol";

contract BatteryManagement is BasicNFToken, Ownable {
    // Для проверки на повторное использование транзакции
    mapping (bytes32 => bool) reused;

    // Для проверки на передачу прав на какую-либо батарею
    // для данного пользователя
    mapping (address => bool) hasBattery;

    // Для проверки на участие батареи в другом контракте
    mapping (bytes20 => bool) isUsed;
```

```
// Временной промежуток запрета разблокирования токенов
uint256 timeout = 3600;

// Контракт для хранения амортизационной комиссии
Fee fees;

// Для оповещения о передаче прав владения батареей новому владельцу
// - адрес предыдущего владельца
// - адрес нового владельца
// - индентификатор батареи
event Transfer(address indexed, address indexed, bytes20);


// Для оповещения после создания контракта на замену батарей
// - адрес контракта
event NewDeal(address newDeal);

ManagementContractInterface public managementContract;
ERC20TokenInterface public erc20;
Deal deal;

// Конструктор контракта
// - адрес контракта, управляющего списком вендоров.
// - адрес контракта, управляющего токенами, в которых будет
//   происходить расчет за замену батарей.
function BatteryManagement(address _mgmt, address _erc20) public {
    managementContract = ManagementContractInterface(_mgmt);
    erc20 = ERC20TokenInterface(_erc20);
    fees = new Fee();
}

// Создает новую батарею
// Владельцем батареи назначается его текущий создатель.
// Создание нового батареи может быть доступно только
// management контракту
// - адрес производителя батареи
// - идентификатор батареи
function createBattery(address _vendor, bytes20 _tokenId) public {
    require(msg.sender == address(managementContract));
    require(!batteryExists(_tokenId));
    _setTokenWithID(_tokenId, _vendor);
    _transfer(0, _vendor, _tokenId);
}

// Меняет владельца токена. Может быть вызвана только
// текущим владельцем
// - адрес нового владельца
// - идентификатор батареи
function transfer(address _to, bytes20 _tokenId) public{
    require(msg.sender == ownerOf[_tokenId]);
    _transfer(msg.sender,_to,_tokenId);
    hasBattery[_to] = true;
    emit Transfer(msg.sender,_to,_tokenId);
}

// Меняет владельца для всех токенов перечисленных в списке. Может быть вызвана
// только текущим владельцем
// - адрес нового владельца
```

```
// - список идентификаторов батарей
function transfer(address _to, bytes20[] _tokenIds) public{
    for (uint i = 0 ; i < _tokenIds.length; i++){
        transfer(_to,_tokenIds[i]);
    }
}

// Определяет по статусу батареи ее идентификатор, затем текущего владельца.
// И меняет владельца токена, отвечающего за батаерею. Может быть вызвана
// только текущим владельцем
// - запакованные данные, хранящие статус батареи и нового получателя. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**124) + t*(2**192) + v*(2**160) + address
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
//function transfer(bytes32, bytes32, bytes32) public;

// Делегирует право аккаунту изменять владельца для токена, которым на текущий
// момент владеет аккаунт-отправитель транзакции. Может выполняться, если
// аккаунт-отправитель тразанкции действительно владеет данным токенов. Для
// одного токена может быть только один делегат. Для отзыва права необходимо
// послать 0 в качестве адреса.
// Генерирует событие Approval.
// - адрес делегата
// - идентификатор токена
function approve(address _to, bytes20 _tokenId) public{
    require(ownerOf[_tokenId] == msg.sender);
    _approve(_to,_tokenId);
}

// Делегирует право аккаунту изменять владельца для токена, которым на текущий
// момент владеет аккаунт, адрес которого определяется в ходе проверки цифровой
// подписи. Может выполняться, если аккаунт, восстанавливаемый из цифровой
// подписи, действительно владеет данным токеном. Для
// одного токена может быть только один делегат. Для отзыва права необходимо
// послать 0 в качестве адреса.
// Цифровая подпись проверяется относительно адреса отправителя транзакции
// (msg.sender), поскольку подразумевается, что тем самым владелец токена
// подтверждает, что делегирование права может быть выполнено с данного адреса.
// Генерирует событие Approval.
// - адрес делегата
// - идентификатор токена
// - v, r, s компоненты цифровой подписи
function delegatedApprove(address _to, bytes20 _tokenId, uint8 v, bytes32 r, bytes32
s) public{
    address _owner = ecrecover(keccak256(msg.sender),v,r,s);
    require(ownerOf[_tokenId] == _owner);
    _approve(_to,_tokenId);
}
// Меняет владельца токена. Может выполняться, только если отправителю
// транзакций делегировано право изменения владельца для данного токена.
// Генерирует событие Transfer.
// - адрес владельца токена
// - адрес получателя
// - идентификатор токена
//function transferFrom(address, address, bytes20) public;

// Возвращает адрес текущего зарегистрированного владельца батареи
// - индентификатор батареи
```

```
//function ownerOf(bytes20) view public returns(address);

// Возвращает адрес производителя батареи
// - идентификатор батареи
function vendorOf(bytes20 _batteryId) view public returns(address) {
    return tokenID[_batteryId];
}

// Проверяет цифровую подпись для данных и возвращает результат
// проверки и адрес вендора.
// Результат проверки: 0 - результат проверки цифровой подписи, показывает
// что она сделана батареей, для которой существует токен; 1 - транзакция
// с таким статусом уже отправлялась в блокчейн, что указывает на возможную
// прослушку траффика; 2 - для  батареи нет соответствующего токена; 999 -
// другая ошибка.
// - число зарядов
// - временная метка
// - v, r, s компоненты цифровой подписи
function verifyBattery(uint256 n, uint256 t, uint8 v, bytes32 r, bytes32 s) public view
returns(uint256, address) {
    address batteryId = getBatteryIdFromSignature(n, t, v, r, s);
    bytes32 ch = generateStatusUsageHash(batteryId, n, t);
    uint256 retval = 999;
    if (batteryExists(bytes20(batteryId))) {
        if (reused[ch]) {
            retval = 1;
        }
        else {
            retval = 0;
        }
    } else {
        retval = 2;
    }
    address vendorId = vendorOf(bytes20(batteryId));
    return (retval, vendorId);

}
// Формирует новый контракт на выполнение операций по замене батарей.
// В контракт передаются идентификаторы батарей, участвующие в сделке,
// адрес аккаунта автомобиля, сумма на компенсацию амортизации батарей,
// сумма за выполенние работ по замене.
// Идентификаторы батарей становятся известны в ходе проверки цифровых
// подписей для статусов батарей.
// Сумма компенсации исчисляется в зависимости от количества заряда,
// переданного в статусе батарей.
// Контракт не должен создаваться, если для батарей с такими
// идентфикаторами нет соответствующего токена, если одна из батарей
// уже участвует в каком-то другом контракте замены или участники контракта
// не являются владельцами батарей.
// - запакованные данные, хранящие статус сразу двух батарей. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n1*(2**160) + t1*(2**128) + v1*(2**96) + n2*(2**64) + t2*(2**32) + v2
// - r компонента подписи для данных старой батареи
// - s компонента подписи для данных старой батареи
// - r компонента подписи для данных новой батареи
// - s компонента подписи для данных новой батареи
// - адрес аккаунта автомобиля
// - количество расчетных токенов, определяющих выполнение работ по замене
function initiateDeal(
    uint256 _p,
```

```
    bytes32 _r1,
    bytes32 _s1,
    bytes32 _r2,
    bytes32 _s2,
    address _car,
    uint256 _scenterFee
)
public
{
  uint256 p = _p;
  bytes20 oldBatteryId = verifyOldBattery(_p, _r1, _s1, _car);
  bytes20 newBatteryId = verifyNewBattery(_p%2**96, _r2, _s2);
  uint256 amortizedFee = getAmortizedFee(getValueFromPack(p, 2**160),
  getValueFromPack(p%2**96, 2**64));
  deal = new Deal(oldBatteryId, newBatteryId, erc20, amortizedFee, _scenterFee, timeout);
  isUsed[oldBatteryId] = true;
  isUsed[newBatteryId] = true;
  reused[generateStatusUsageHash(address(oldBatteryId), getValueFromPack(p, 2**160),
  getValueFromPack(p%2**   160, 2**128))] = true;
  reused[generateStatusUsageHash(address(newBatteryId), getValueFromPack(p%2**96, 2**64),
    p%2**64, 2**32))] = true;
  getValueFromPack( approve(address(deal), newBatteryId);
  emit NewDeal(address(deal));
}
// Возвращает количество зарядов для конкретной батареи, зарегистрированных
// в данный момент
// - идентификатор батареи
//function chargesNumber(bytes20) view public returns(uint256);

// Извлекает из переданных данных цифровую подпись текущего владельца батареи,
// адрес нового владельца батареи, ее статус. Изменяет владельца, если
// нет нарущений прав владения для восстановленного из подпсис адреса.
// - запакованные данные, хранящие статус батареи и нового получателя. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки (от старших
//   байт к младшим):
//    0..3   байт - число зарядов
//    4..7   байт - временная метка
//    8..75  байт - v, r, s компоненты цифровой подписи статуса батареи
//   76..95  байт - адрес нового владельца
//   96..163 байт - v, r, s компоненты цифровой подписи текущего владельца
//function approvedTransfer(bytes) public;

// Проверяет переданный адрес, если он является контрактом сделки.
// Возвращает 0 - контракт сделки существует и аккаунт, с которого
// осуществляется запрос, является участником сделки; 1 - контракт сделки
// существует, но аккаунт не является участников сделки; 2 - контракт
// сделки не существует.
//function checkDealContract(address) view public returns(uint256);

// Устанавливает верменной промежуток, после которого разрешено
// разблокирование расчетных токенов контрактом сделки. Данная операция
// разрешена только аккаунту производителя ПО.
// - размер временного промежутка в секундах.
function setTimeoutThreshold(uint256 _timeout) public onlyOwner {
    timeout = _timeout;
}

// Возвращает временной промежуток, запрета разблокирования расчетных
// токенов, установленный на текущий момент.
//function timeoutThreshold() view public returns(uint256);
```

```
// Устанавливает значения минимальной комиссии
// за амортизационную стоимость батарей
// - значения в зависимости от разности батарей
// - отступ, с которого необходимо устанавливать комиссию
function setMinimalFees(uint256[] _fees, uint counter) public {
    fees.setMinimalFees(_fees, counter);
}

// Устанавливает значения максимальной комиссии
// за амортизационную стоимость батарей
// - значения в зависимости от разности батарей
// - отступ, с которого необходимо устанавливать комиссию
function setMaximalFees(uint256[] _fees, uint counter) public {
    fees.setMaximalFees(_fees, counter);
}

// TODO: only for testing. Delete before release
function getMaximalFees(uint256 n) public returns(uint256) {
    return fees.maximalFees(n);
}

// Высчитывает хэш, используемый для проверки на повторное
// использование транзакции
// - адрес, полученный из подписи и хэша транзакции
// - количество заряда батареи
// - время получения информации о батарее
function generateStatusUsageHash(address _addr, uint256 _n, uint256 _t) internal pure
returns (bytes32) {
    return keccak256(_addr, _n, _t);
}
// Вычисляет Id батареи по информации из её прошивки
// - количество зарядок
// - время получения информации
// - v компонента подписи для батареи
// - r компонента подписи для батареи
// - s компонента подписи для батареи
function getBatteryIdFromSignature(uint256 _n, uint256 _t, uint8 _v, bytes32 _r,
bytes32 _s)
    public
    pure
    returns (address batteryId)
{
    uint256 message = 2**32 * _n + _t;
    bytes32 h = keccak256(message);
    batteryId = ecrecover(h, _v, _r, _s);
}

// Проверяет зарегистрирован ли токен с таким идентификатором любым из производителей.
function batteryExists(bytes20 _batteryId) internal view returns (bool) {
    return tokenID[_batteryId] != address(0);
}

// Расчитывает амортизационную стоимость при замене батареи
function getAmortizedFee(uint256 _oldBattery, uint256 _newBattery) internal view
returns(uint256)
    if (_newBattery >= 300) {
        return 0;
    } else if (_oldBattery >= 300) {
        return 1000;
    } else if (_newBattery >= 150) {
        return _oldBattery - _newBattery;
    } else if (_newBattery >= 50) {
```

```
        if (_oldBattery >= 150) {
            return fees.minimalFees(_oldBattery - _newBattery);
        }
        else {
            return _oldBattery - _newBattery;
        }
    } else if (_oldBattery >= 150) {
        return fees.maximalFees(_oldBattery - _newBattery);
    } else if (_oldBattery >= 50) {
        return fees.minimalFees(_oldBattery - _newBattery);
    } else {
        return _oldBattery - _newBattery;
    }
}

// Проверяет батарею предоставленную машиной
function verifyOldBattery(
    uint256 _p,
    bytes32 _r1,
    bytes32 _s1,
    address _car
)
internal view returns(bytes20) {
    uint256 n = SafeMath.div(_p, 2**160);
    _p %= 2**160;
    uint256 t = SafeMath.div(_p, 2**128);
    _p %= 2**128;
    uint256 v = SafeMath.div(_p, 2**96);
    _p %= 2**96;

    require(managementContract.cars(_car));
    uint256 oldBatteryValidity;
    address oldBatteryVendorAddress;
    (oldBatteryValidity, oldBatteryVendorAddress) = verifyBattery(n, t, uint8(v), _r1, _s1);
    require(oldBatteryValidity == 0);
    address oldBatteryId = getBatteryIdFromSignature(n, t, uint8(v), _r1, _s1);
    require(ownerOf[bytes20(oldBatteryId)] == _car);
    require(hasBattery[_car]);
    require(!isUsed[bytes20(oldBatteryId)]);

    return(bytes20(oldBatteryId));
}

// Проверяет батарею предоставленную сервисным центром
function verifyNewBattery(
    uint256 _p,
    bytes32 _r2,
    bytes32 _s2
)
internal view returns(bytes20) {
    uint256 n = SafeMath.div(_p, 2**64);
    _p %= 2**64;
    uint256 t = SafeMath.div(_p, 2**32);
    _p %= 2**32;
    uint256 v = _p;

    require(managementContract.serviceCenters(msg.sender));
    uint256 newBatteryValidity;
    address newBatteryVendorAddress;
    (newBatteryValidity, newBatteryVendorAddress) = verifyBattery(n, t, uint8(v), _r2, _s2);
```

```
        require(newBatteryValidity == 0);
        address newBatteryId = getBatteryIdFromSignature(n, t, uint8(v), _r2, _s2);
        require(ownerOf[bytes20(newBatteryId)] == msg.sender);
        require(hasBattery[msg.sender]);
        require(!isUsed[bytes20(newBatteryId)]);

        return bytes20(newBatteryId);
    }

    // Получает значение из упакованных данных с необходимым сдвигом
    function getValueFromPack(uint256 _p, uint256 _power) internal pure returns(uint256) {
        return SafeMath.div(_p, _power);
    }

}
```

./contracts/Deal.sol

```
pragma solidity ^0.4.19;

import "./BatteryManagementInterface.sol";
import "./ManagementContract.sol";

contract Deal {

    BatteryManagementInterface batteryContract;
    ManagementContract managementContract;

    bytes20 oldBat;
    bytes20 newBat;
    address erc20;
    uint256 ammortizedFee;
    uint256 serviceFeeValue;
    uint256 timeout;
    uint256 deal_state;

    // Конструктор контракта
    // - идентификатор старой батареи
    // - идентификатор новой батареи
    // - адрес контракта расчетных токенов
    // - количество токенов, определяющих компенсацию амортизации батарей
    // - количество токенов, определяющих выполнение работ по замене
    // - временной промежуток, в ходе которого запрещено разблокирование
    //   расчетных токенов.
    function Deal(bytes20 _oldBat, bytes20 _newBat, address _erc20, uint256 _ammorFee,
    uint256 _serviceFee, uint256 _timeout) public {
        oldBat = _oldBat;
        newBat = _newBat;
        erc20 = _erc20;
        ammortizedFee = _ammorFee;
        serviceFeeValue = _serviceFee;
        timeout = _timeout;
        deal_state = 1;
        batteryContract = BatteryManagementInterface(msg.sender);
        managementContract = ManagementContract(batteryContract.managementContract());
    }

    // Возвращают идентификаторы батарей, вовлеченных в контракт.
    function oldBattery() view public returns(bytes20) {
        return oldBat;
    }
```

```
function newBattery() view public returns(bytes20) {
    return newBat;
}

// Возвращают информацию о батареях, вовлеченных в контракт.
// Возвращаемые значения:
// - количество зарядов
// - идентификатор производителя
// - наименование производителя
function oldBatteryInfo() view public returns(uint256, bytes4, bytes)
{
    uint256 n = batteryContract.chargesNumber(oldBat);
    address vendorAddress = batteryContract.vendorOf(oldBat);
    bytes4 vendorId = managementContract.vendorId(vendorAddress);
    //bytes vendorName = ;

    return (n, vendorId, getVendorNamesByChunks(vendorId));
}

function newBatteryInfo() view public returns(uint256, bytes4, bytes) {
    uint256 n = batteryContract.chargesNumber(newBat);
    address vendorAddress = batteryContract.vendorOf(newBat);
    bytes4 vendorId = managementContract.vendorId(vendorAddress);
    //bytes vendorName = getVendorNamesByChunks(vendorId);

    return (n, vendorId, getVendorNamesByChunks(vendorId));
}

// Возвращает количество токенов, определяющих компенсацию амортизации
// батарей
function deprecationValue() view public returns(uint256) {
    return ammortizedFee;
}

// Возвращает количество токенов, определяющих выполнение работ по
// замене
function serviceFee() view public returns(uint256) {
    return serviceFeeValue;
}

// Возвращает текущий статус контракта
// 1 - Ожидание согласия со сделкой от автомобиля
// 2 - Согласие со сделкой получено
// 3 - Оплата работ выполнена
function state() view public returns(uint256) {
    return deal_state;
}

// Проверяет, что переданный статус батареи получен от батареи, которая
// зарегистрирована к замене (старая), что данный запрос идет от текущего
// владельца батареи, который участвует в контракте в роли получателя новой
// батареи. Если все условия выполняются, то токены идущие в оплату
// контракта переводятся на счет контракта.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
// function agreeToDeal(uint256, bytes32, bytes32) public;
```

```
// Проверяет, что переданный статус батареи получен от батареи, на которую
// должна происходить замена. При этом:
// 1) либо данный запрос идет от аккаунта, который участвует в контракте в роли
//    получателя новой батареи
// 2) либо запрос идет от аккаунта, который участвует в контракте в роли того,
//    кто предоставляет сервис по замене и с момента запроса согласия с
//    контрактом  прошло 24 часа
// Если все условия выполняются, то токены идущие в оплату контракта
// переводятся на счет сервисного центра, владельцы батарей изменяются.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
// function confirmDeal(uint256, bytes32, bytes32) public;

// Перепосылает аргументы в контракт BatteryManagement для проверки цифровой
// подписи. Возвращаемый результат проверки такой же, как и в оригинальной
// функции: 0 - результат проверки цифровой подписи, показывает
// что она сделана батареей, для которой существует токен; 1 - транзакция
// с таким статусом уже отправлялась в блокчейн, что указывает на возможную
// прослушку траффика; 2 - для  батареи нет соответствующего токена; 999 -
// другая ошибка.
// - число зарядов
// - временная метка
// - v, r, s компоненты цифровой подписи
function verifyBattery(uint256 n, uint256 t, uint8 v, bytes32 r, bytes32 s) public
view returns(uint256,
    return batteryContract.verifyBattery(n, t, v, r, s);
}

// Обрабатывает запрос на разблокирование расчетных токенов и возвращает
// токены на счет электромобиля, владельцы батарей, участвующих в сделке,
// не изменяются.
// Выполняется только если отправитель запроса, зарегистрирован в качестве
// получателя новой батареи, если проверка подписи из статуса батареи указывает
// на то, что батарея не была заменена, и если время из статуса больше,
// чем разрешенное время разблокировки токенов.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
// function releaseTokensByCar(uint256, bytes32, bytes32) public;

// Обрабатывает запрос на разблокирование расчетных токенов и перечисляет
// токены на счет сервисного центра, происходит смена владельцев батарей,
// участвующих в сделке.
// Выполняется только если отправитель запроса, зарегистрирован в качестве
// изначального владельца новой батареи, если проверка подписи из статуса
// батареи указывает на то, что батарея действительно была заменена, и
// если время из статуса больше, чем разрешенное время разблокировки
токенов. // - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
```

```
    // function releaseTokensByServiceCenter(uint256, bytes32, bytes32) public;

    // Отменяет сделку, если она получатель новой батареи, не согласился с
    // контрактом. Выполняется только если отправитель запроса - сервисный
    // центр.
    // function cancelDeal() public;

    function getVendorNamesByChunks(bytes4 _id) public view returns(bytes){
        uint256 l = managementContract.vendorNamesDataLen(_id);
        bytes memory data = new bytes(l);

        uint256 pos = 0;
        for(uint256 i=0; i<(l/32)+1; i++) {
            bytes32 d = managementContract.vendorNamesDataChunk(_id, i);
            uint256 j=0;
            while(j<32 && j+pos<l) {
                data[pos+j] = d[j];
                j++;
            }
            pos = pos + j;
        }

        return(data);
    }
}
```

./contracts/ERC20Token.sol

```
pragma solidity ^0.4.21;

import "./erc20/MintableToken.sol";

contract ERC20Token is MintableToken {

    function ERC20Token() public {
        mint(owner, 100 ether);
    }
}
```

./contracts/Fee.sol

```
pragma solidity ^0.4.19;

contract Fee {
    // Для хранения минимальных компенсаций в зависимости
    // от разницы количества зарядок батарей
    mapping (uint256 => uint256) public minimalFees;

    // Для хранения минимальных компенсаций в зависимости
    // от разницы количества зарядок батарей
    mapping (uint256 => uint256) public maximalFees;

    function setMinimalFees(uint256[] _fees, uint counter) public {
        for (uint i = counter; i<_fees.length+counter; i++) {
            minimalFees[i] = _fees[i-counter];
        }
    }

    function setMaximalFees(uint256[] _fees, uint counter) public {
```

```
        for (uint i = counter; i<_fees.length+counter; i++) {
            maximalFees[i] = _fees[i-counter];
        }
    }
```

./contracts/ManagementContract.sol

```
pragma solidity ^0.4.19;

import "./ServiceProviderWallet.sol";
import "./BatteryManagementInterface.sol";
import "./lib/Ownable.sol";

contract ManagementContract is Ownable {

    //Депозит для каждого вендора
    mapping (address => uint256) public vendorDeposit;

    //По адресу отправителя определяет имя производителя которое ему принадлежит
    mapping (address => bytes4) public vendorId;

    //По индефикатору возвращает имя производителя
    mapping (bytes4 => bytes) public vendorNames;

    //Проверка уже зарегестрированного имени
    mapping (bytes => bool) registeredVendor;

    //Проверка уже зарегестрированного адреса производителя
    mapping (address => bool) public registeredVendorAddress;

    //По адресу возвращает batteryFee для данного производителя
    mapping (address => uint256) vendorBatteryFee;

    // Возвращает истину или ложь в зависимости от того, зарегистрирован
    // электромобиль с указанным адресом в системе или нет.
    mapping (address => bool) public serviceCenters;

    // Возвращает истину или ложь в зависимости от того, зарегистрирован
    // электромобиль с указанным адресом в системе или нет.
    mapping (address => bool) public cars;

    // Контракт управляющий информацией о батареях
    BatteryManagementInterface public batteryManagement;
    // Контракт кошелька
    ServiceProviderWallet public walletContract;

    //Цена за создание одной батареи
    uint256 BatFee;
    // Для оповещения регистрации нового производителя
    // - адрес аккаунта из-под которого проходила регистрация
    // - идентификатор производителя
    event Vendor(address owner, bytes4 tokenId);

    // Для оповещения о создании новой батареи
    // - идентификатор производителя
    // - идентификатор батареи
    event NewBattery(bytes4 vendorId, bytes20 tokenId);

    // Конструктор контракта
    // - адрес контракта, ответственного за накопление криптовалюты,
```

```
//   перечисляемой в качестве депозита за использование сервиса.
// - сумму сбора за выпуск одной батареи
function ManagementContract(address _wallet, uint256 _batfee) public{
    BatFee = _batfee;
    walletContract = ServiceProviderWallet(_wallet);
}

// Устанавливает адрес для контракта, ответственного за
// управление информацией о батареях.
// Доступен только создателю management контракта
// - адрес контракта, управляющего инфорацией о батареях
function setBatteryManagementContract(address _batmgmt) onlyOwner public {
    batteryManagement = BatteryManagementInterface(_batmgmt);
}

// Регистрирует вендора, если при вызове метода перечисляется
// достаточно средств.
// - наименование производителя
function registerVendor(bytes _name) public payable{
    require(msg.value >= BatFee*1000);
    require(!registeredVendor[_name]);
    registeredVendor[_name] = true;
    require(!registeredVendorAddress[msg.sender]);
    registeredVendorAddress[msg.sender] = true;
    vendorBatteryFee[msg.sender] = BatFee;

    bytes4 _nameSym = bytes4(keccak256(msg.sender,_name,block.number));
    vendorDeposit[msg.sender] += msg.value;
    address(walletContract).transfer(msg.value);

    vendorId[msg.sender] = _nameSym;
    vendorNames[_nameSym] = _name;

    Vendor(msg.sender,_nameSym);
}

// Регистрирует новые батареи, если при вызове метода на балансе
// данного производителя достаточно средств. Во время регистрации
// батарей баланс уменьшается соответственно количеству батареи и
// цене, установленной для данного производителя на текущий момент.
// - идентификаторы батарей
function registerBatteries(bytes20[] _ids) public payable{
    uint _n = _ids.length;
    require(msg.value + vendorDeposit[msg.sender] >= _n * vendorBatteryFee[msg.sender]);
    bytes4 _tokenId = vendorId[msg.sender];
    require(_tokenId != "");
    vendorDeposit[msg.sender] += msg.value - (_n * vendorBatteryFee[msg.sender]);
    if (msg.value > 0){
        address(walletContract).transfer(msg.value);
    }
    for (uint i=0; i < _n; i++){
        batteryManagement.createBattery(msg.sender, _ids[i]);
        NewBattery(_tokenId, _ids[i]);
    }

}

// Устанавливает сумму сбора за выпуск одной батареи
// - сумма сбора в wei
function setFee(uint256 pr) public onlyOwner {
```

```
        BatFee = pr;
    }

    // Возвращает сумму сбора в wei, который будет списываться с депозита
    // за каждую выпущенную батарею. Если запрос идет от зарегистрированного
    // производителя батарей, то выдается та сумма сбора, которая была на момент
    // его регистрации.
    function batteryFee() public view returns(uint256){
        if (vendorBatteryFee[msg.sender] != 0) {
            return vendorBatteryFee[msg.sender];
        }
        return BatFee;
    }

    // Возвращает сумму сбора в wei, которую необходимо перечислить в качестве
    // депозита при регистрации производителя батарей. Сумма депозита
    // равна сумме за регистрацию 1000 батарей, что позволит защититься от
    // мошенников, поскольку требует серьезных вложений.
    function registrationDeposit() public view returns(uint256){
        return 1000 * BatFee;
    }

    // Регистрирует в системе адрес отправителя транзакции, как сервис центр.
    // Регистрация просиходит только если данный адрес уже не был зарегистрирован
    // в качестве сервис центра или электромобиля.
    function registerServiceCenter() public{
        require(!cars[msg.sender]);
        require(!serviceCenters[msg.sender]);
        serviceCenters[msg.sender] = true;
    }

    // Регистрирует в системе адрес отправителя транзакции, электромобиль.
    // Регистрация просиходит только если данный адрес уже не был зарегистрирован
    // в качестве сервис центра или электромобиля.
    function registerCar() public{
        require(!cars[msg.sender]);
        require(!serviceCenters[msg.sender]);
        cars[msg.sender] = true;
    }

    function vendorNamesDataLen(bytes4 _id) public view returns(uint256) {
        return(vendorNames[_id].length);
    }

    function vendorNamesDataChunk(bytes4 _id, uint256 n) public view returns(bytes32){
        uint256 res = 0;
        uint256 i = 0;
        while((i<32) && (i<(vendorNamesDataLen(_id)-n*32))) {
            res = res + uint256(vendorNames[_id][i+n*32]) * 2 ** ((31-i)*8);
            i++;
        }
        return(bytes32(res));
    }

}
```

./contracts/NFT/BasicNFToken.sol

```
pragma solidity ^0.4.19;
```

```
contract BasicNFToken {
    /// @dev A mapping from Token IDs to the address that owns them.
    ///      Even new token is created with a non-zero owner.
    mapping (bytes20 => address) public ownerOf;

    /// @dev A mapping from Token IDs to an address that has been approved to call
    ///      transferFrom(). Each Token can only have one approved address for transfer
    ///      at any time. A zero value means no approval is outstanding.
    mapping (bytes20 => address) public tokenIdToApproved;

    // @dev A mapping from Token IDs to vendor ID
    mapping (bytes20 => address) public tokenID;

    // Для оповещения, что какому-то аккунту делегировано право менять владельца
    // токена
    // - владелец токена
    // - адрес далегата
    // - идентификатор токена, которым может распоряжаться делегат
    event Approval(address indexed _owner, address indexed _approved, bytes20 _tokenId);

    function _setTokenWithID(bytes20 _tokenId, address _data) internal {
        tokenID[_tokenId] = _data;
    }

    /// @dev Assigns ownership of a specific token to an address.
    function _transfer(address _from, address _to, bytes20 _tokenId) internal {
        // transfer ownership
        ownerOf[_tokenId] = _to;
        // When creating new token _from is 0x0, but we can't account that address.
        if (_from != address(0)) {
            // clear any previously approved ownership exchange
            delete tokenIdToApproved[_tokenId];
        }
    }

    function _approve(address _to, bytes20 _tokenId) internal {
        tokenIdToApproved[_tokenId] = _to;
        // Emit approval event.
        emit Approval(msg.sender, _to, _tokenId);
    }
}
```

./contracts/ServiceProviderWallet.sol

```
pragma solidity ^0.4.21;

import './lib/Ownable.sol';

contract ServiceProviderWallet is Ownable {
    // Для оповещения поступлении новых средств
    // - адрес, с которого перечислены средства
    // - сумма поступления
    event Received(address, uint256);

    // Для оповещения о выдачи части средств
    // - адрес, какой инициировал выдачу средств
    // - адрес, на который направлены средства
    // - сумма к выдаче
    event Withdraw(address, address, uint256);
```

```
    function () public payable {
        require(msg.value > 0);
        emit Received(msg.sender, msg.value);
    }

    // Используется для выдачи части средств с адреса контаркта
    function withdraw(address _to, uint256 _value) onlyOwner external {
        _to.transfer(_value);
        emit Withdraw(msg.sender, _to, _value);
    }

    // Позволяет удалить контракт
    function kill() onlyOwner external {
        selfdestruct(owner);
    }
}
```

./contracts/erc20/BasicToken.sol

```
pragma solidity ^0.4.21;

import "./ERC20Basic.sol";
import "../lib/SafeMath.sol";

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  uint256 totalSupply_;

  /**
  * @dev total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

    // SafeMath.sub will throw if there is not enough balance.
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  /**
  * @dev Gets the balance of the specified address.
```

```
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }

}
```

./contracts/erc20/ERC20.sol

```
pragma solidity ^0.4.21;

import "./ERC20Basic.sol";

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

./contracts/erc20/ERC20Basic.sol

```
pragma solidity ^0.4.21;

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}
```

./contracts/erc20/MintableToken.sol

```
pragma solidity ^0.4.21;

import "../lib/Ownable.sol";
import "./StandardToken.sol";

/**
 * @title Mintable token
 * @dev Simple ERC20 Token example, with mintable token creation
 * @dev Issue: * https://github.com/OpenZeppelin/zeppelin-solidity/issues/120
 * Based on code by TokenMarketNet: https://github.com/TokenMarketNet/ico/blob/master/
 contracts/MintableToken. */ sol
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();
```

```
  bool public mintingFinished = false;

  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  function finishMinting() onlyOwner canMint public returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
  }
}
```

./contracts/erc20/StandardToken.sol

```
pragma solidity ^0.4.21;

import "./ERC20.sol";
import "./BasicToken.sol";

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 * @dev Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/
 smart_contract/*/FirstBloodToken.sol

contract StandardToken is ERC20, BasicToken {
  mapping (address => mapping (address => uint256)) internal allowed;


  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[_from]);
```

```
  require(_value <= allowed[_from][msg.sender]);

  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  emit Transfer(_from, _to, _value);
  return true;
}

/**
 * @dev Approve the passed address to spend the specified amount of tokens on behalf of
 msg.sender.
 * Beware that changing an allowance with this method brings the risk that someone may use
   both the old
 * and the new allowance by unfortunate transaction ordering. One possible solution to
 mitigate this
 * race condition is to first reduce the spender's allowance to 0 and set the desired value
 afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param _spender The address which will spend the funds.
 * @param _value The amount of tokens to be spent.
 */
function approve(address _spender, uint256 _value) public returns (bool) {
  allowed[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
}
/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param _owner address The address which owns the funds.
 * @param _spender address The address which will spend the funds.
 * @return A uint256 specifying the amount of tokens still available for the spender.
 */
function allowance(address _owner, address _spender) public view returns (uint256) {
  return allowed[_owner][_spender];
}
/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the allowance by.
 */
function increaseApproval(address _spender, uint _addedValue) public returns (bool) {
  allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
  emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
  return true;
}
/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to decrease the allowance by.
```

```
   */
  function decreaseApproval(address _spender, uint _subtractedValue) public returns (bool) {
    uint oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}
```

./contracts/lib/Ownable.sol

```
pragma solidity ^0.4.21;

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 * Ref: Taken from
 * https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/ownership/
 Ownable.sol
 * /
contract Ownable {
  address public owner;

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  function Ownable() public {
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }
}
```

./contracts/lib/SafeMath.sol

```
pragma solidity ^0.4.21;
```

```
/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  /**
  * @dev Substracts two numbers, throws on overflow (i.e. if subtrahend is greater than
  minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }
  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {

    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

./fee_generation.py

```
minimalFeeValue = 1.003
maximalFeeValue = 1.0035

def generateMinFee():
    min_fee = []
    for delta in range(101, 300):
        min_fee.append(int(delta * (maximalFeeValue ** delta)))
    return min_fee

def generateMaxFee():
    max_fee = []
    for delta in range(1, 250):
        max_fee.append(int(delta * (minimalFeeValue ** delta)))
```

./scenter.py

```
import warnings

warnings.simplefilter("ignore", category=DeprecationWarning)

# install web3 >=4.0.0 by `pip install --upgrade 'web3==4.0.0b11'`
from web3 import Web3
from web3.middleware import geth_poa_middleware
from web3.utils.transactions import wait_for_transaction_receipt

from json import loads

import argparse, subprocess
import sys, os

from utils import createNewAccount, openDataBase, unlockAccount, getActualGasPrice, \
    compileContracts, initializeContractFactory, ManagementContract, BatteryContract,
    DealContract
from verify_battery import verify, return_ntvrs_from, batteryExists

web3 = Web3()

# configure provider to work with PoA chains
web3.middleware_stack.inject(geth_poa_middleware, layer=0)

accountDatabaseName = 'scenter.json'
mgmtContractDatabaseName = 'database.json'

gasPrice = getActualGasPrice(web3)

registrationRequiredGas = 50000


def _getBatteryStatus(_path):
    if not os.path.exists(_path):
        return

    args = ['python', _path, '--getJson']

    batteryOut = subprocess.Popen(args, stdout=subprocess.PIPE)
    batteryStatus = loads(batteryOut.communicate()[0].decode())

    return batteryStatus


def reg():
    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access management contract database")
        return


    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    actor = data["account"]

    tx = {'from': actor, 'gasPrice': gasPrice}
```

```
    mgmtContract = ManagementContract(web3)

    if registrationRequiredGas * gasPrice > web3.eth.getBalance(actor):
        print("No enough funds to send transaction")
        return

    unlockAccount(web3, actor, data["password"])

    try:
        txHash = mgmtContract.functions.registerServiceCenter().transact(tx)
    except ValueError:
        print("Already registered")
        return

    receipt = wait_for_transaction_receipt(web3, txHash)
    if receipt.status == 1:
        print("Registered successfully")


def createDeal(_bat_new, _bat_old, _car, _service_price):
    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    actor = data["account"]

    tx = {'from': actor, 'gasPrice': gasPrice}

    unlockAccount(web3, actor, data["password"])

    if not batteryExists(_bat_old):
        print('Cannot access battery')
        return
    n1, t1, v1, r1, s1 = return_ntvrs_from(_bat_old)

    if not batteryExists(_bat_new):
        print('Cannot access battery')
    return
    n2, t2, v2, r2, s2 = return_ntvrs_from(_bat_new)

    p = n1 * (2 ** 160) + t1 * (2 ** 128) + v1 * (2 ** 96) + n2 * (2 ** 64) + t2 *
    (2 ** 32) + v2
    mgmtContract = ManagementContract(web3)
    _bat_contract_address = mgmtContract.functions.batteryManagement().call()
    batContract = BatteryContract(web3, _bat_contract_address)

    try:
        txHash = batContract.functions.initiateDeal(p, web3.toBytes(hexstr=r1),
        web3.toBytes(hexstr=s1), web3.toBytes(hexstr=r2), web3.toBytes(hexstr=s2),
        _car, _service_price).transact(tx)
        receipt = wait_for_transaction_receipt(web3, txHash)
        if receipt.status == 0:
            print("Deal was not created.")
            return
    except BaseException as error:
        if error.args[0]['message'] == 'insufficient funds for gas * price + value':
            print('No enough funds to send transaction')
            return
        print("Deal was not created.")
        return
```

```
    # Contract.events.<event name>.createFilter(fromBlock=block, [toBlock=block,
    argument_filters={"arg1": "value"}
    logs = batContract.events.NewDeal().processReceipt(receipt)[::-1]
    for log in logs:
        try:
            if log['args']['newDeal']:
                address_new_deal = log['args']['newDeal']
                print(f"Deal: {address_new_deal}")
                return
        except:
            return

def create_parser():
    parser = argparse.ArgumentParser(
        description='Service provider tool',
        epilog="""
    It is expected that Web3 provider specified by WEB3_PROVIDER_URI
    environment variable. E.g.
    WEB3_PROVIDER_URI=file:///path/to/node/rpc-json/file.ipc
    WEB3_PROVIDER_URI=http://192.168.1.2:8545
    """
    )

    parser.add_argument(
        '--new', type=str, required=False,
        help='Add new service scenter account'
    )

    parser.add_argument(
        '--reg', action='store_true', required=False,
        help='Register service scenter in the chain'
    )

    parser.add_argument(
        '--verify', type=str, required=False,
        help='Verify battery on validity'
    )

    parser.add_argument(
        '--contract', nargs=4, required=False,
        help='Create deal between car and service center'
    )

    return parser


def main():
    parser = create_parser()
    args = parser.parse_args()

    if args.new:
        print(createNewAccount(web3, args.new, accountDatabaseName))
    elif args.reg:
        reg()
    elif args.verify:
        if batteryExists(args.verify):
            result = verify(web3, args.verify)
            if result[0] == 0:
                n = result[1]
```

```
                vendorId = result[2]
                vendorName = result[3]
                print("Verified successfully.")
                print('Total charges: {}'.format(n))
                print('Vendor ID: {}'.format(vendorId))
                print('Vendor Name: {}'.format(vendorName))
            elif result[0] == 1:
                print('Battery with the same status already replaced. Probably the
            battery forged.')
            elif result[0] == 2:
                print('Verification failed. Probably the battery forged.')
            else:
                print('Error. Cannot verify battery.')
        else:
            print('Cannot access battery')
    elif args.contract:
        bat_old = args.contract[0]
        bat_new = args.contract[1]
        car = args.contract[2]
        service_price = int(args.contract[3])
        createDeal(bat_old, bat_new, car, service_price)
    else:
        sys.exit("No parameters provided")


if __name__ == '__main__':
    main()
```

./setup.py

```
import warnings

warnings.simplefilter("ignore", category=DeprecationWarning)

# install web3 >=4.0.0 by `pip install --upgrade 'web3==4.0.0b11'`
from web3 import Web3
from web3.middleware import geth_poa_middleware
from web3.utils.transactions import wait_for_transaction_receipt

from web3.utils.threads import (
    Timeout,
)
from random import random

import argparse
import sys

from utils import createNewAccount, writeDataBase, openDataBase, unlockAccount,
getActualGasPrice
from utils import compileContracts, initializeContractFactory
from fee_generation import generateMaxFee, generateMinFee
from utils import createNewAccount, writeDataBase, openDataBase, unlockAccount

web3 = Web3()

# configure provider to work with PoA chains
web3.middleware_stack.inject(geth_poa_middleware, layer=0)

accountDatabaseName = 'account.json'
mgmtContractDatabaseName = 'database.json'
```

```
gasPrice = getActualGasPrice(web3)

requiredGasForSetting = 30000

# contract files
contracts = {'token': ('contracts/ERC20Token.sol', 'ERC20Token'),
             'wallet': ('contracts/ServiceProviderWallet.sol', 'ServiceProviderWallet'),
             'mgmt': ('contracts/ManagementContract.sol', 'ManagementContract'),
             'battery': ('contracts/BatteryManagement.sol', 'BatteryManagement')}


def _deployContractAndWait(_actor, _contrSourceFile, _contractName, args=None):
    txHash = _deployContract(_actor, _contrSourceFile, _contractName, args)
    receipt = wait_for_transaction_receipt(web3, txHash)
    return receipt.contractAddress


def _deployContract(_actor, _contrSourceFile, _contractName, args=None):
    compiled = compileContracts(_contrSourceFile)
    contract = initializeContractFactory(web3, compiled, _contrSourceFile + ":" +
    _contractName)
    tx = {'from': _actor, 'gasPrice': gasPrice}
    return contract.deploy(transaction=tx, args=args)



def _waitForValidation(_w3, _txdict, _tmout=120):
    receiptslist = {}
    for i in list(_txdict):
        receiptslist[i] = [_txdict[i], None]
    confirmations = len(list(_txdict))

    with Timeout(_tmout) as timeout:
        while (confirmations > 0):
            for i in list(_txdict):
                if receiptslist[i][1] is None:
                    txn_receipt = _w3.eth.getTransactionReceipt(receiptslist[i][0])
                    if txn_receipt is not None:
                        receiptslist[i][1] = txn_receipt
                        confirmations = confirmations - 1
                timeout.sleep(random())

    return receiptslist


def _createMgmtContractDatabase(_contractAddress):
    data = {'mgmtContract': _contractAddress}
    writeDataBase(data, mgmtContractDatabaseName)

def _setupMaximalAndMinimalFees(_actor, _contract):
    tx = {'from': _actor, 'gasPrice': gasPrice}
    minFee = generateMinFee()
    for i in range(0, len(minFee), 100):
        _contract.functions.setMinimalFees(minFee[i:100 + i], i + 1).transact(tx)
    maxFee = generateMaxFee()
    for i in range(0, len(maxFee), 100):
        _contract.functions.setMaximalFees(maxFee[i:100 + i], i + 1).transact(tx)

def setup(_serviceFee):
```

```
serviceFee = web3.toWei(_serviceFee, 'ether')
data = openDataBase(accountDatabaseName)
if data is None:
    print("Cannot access account database")
    return
actor = data["account"]
unlockAccount(web3, actor, data["password"])
try:
    # deploy token and wallet in one block (hopefully)
    txd = {}
    for i in ['token', 'wallet']:
        txd[i] = _deployContract(actor, contracts[i][0], contracts[i][1])

    # wait for deployment transactions validation
    receiptd = _waitForValidation(web3, txd)

    currencyTokenContractAddress = receiptd['token'][1].contractAddress
    serviceProviderWalletAddress = receiptd['wallet'][1].contractAddress

    if (receiptd['token'][1] is not None) and (receiptd['wallet'][1] is not None):
        serviceProviderWalletAddress = receiptd['wallet'][1].contractAddress
        currencyTokenContractAddress = receiptd['token'][1].contractAddress

        if serviceProviderWalletAddress is not None:
            # deploy management contract
            mgmtContractAddress = _deployContractAndWait(actor, contracts['mgmt'][0],
            contracts['mgmt '][1 ],
            [serviceProviderWalletAddress, serviceFee])


            if mgmtContractAddress is not None:
                _createMgmtContractDatabase(mgmtContractAddress)

                # deploy battery management
                batteryMgmtContractAddress = _deployContractAndWait(actor,
                contracts['battery'][0],
                contracts['battery'][1],
                [mgmtContractAddress, currencyTokenContractAddress])
                if batteryMgmtContractAddress is not None:
                    compiledContract = compileContracts(contracts['mgmt'][0])
                    mgmtContract = initializeContractFactory(web3, compiledContract,
                    contracts['mgmt'][0] + ':' + contracts['mgmt'][1],
                    mgmtContractAddress)
                    txHash = mgmtContract.functions.setBatteryManagementContract
                    (batteryMgmtContractAddress).transact (
                    {'from': actor, 'gasPrice': gasPrice})
                    receipt = wait_for_transaction_receipt(web3, txHash)

                    if receipt.status == 1:
                        print('Management contract:', mgmtContractAddress, sep=' ')
                        print('Wallet contract:', serviceProviderWalletAddress, sep=' ')
                        print('Currency contract:', currencyTokenContractAddress, sep=' ')

                        compiledContract = compileContracts(contracts['battery'][0])
                        batteryMgmtContract = initializeContractFactory(web3, compiledContract,
                        contracts['battery'][0]+':'+contracts['battery'][1],
                        batteryMgmtContractAddress)
                        _setupMaximalAndMinimalFees(actor, batteryMgmtContract)
```

```
                            return
    except ValueError as error:
        if error.args[0]['message'] == 'insufficient funds for gas * price + value':
            print('No enough funds to send transaction')
            return
    print('Contracts deployment and configuration failed')


def setFee(_serviceFee):
    serviceFee = web3.toWei(_serviceFee, 'ether')

    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    mgmtContractAddress = data['mgmtContract']

    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    actor = data["account"]

    tx = {'from': actor, 'gasPrice': gasPrice}

    compiledContract = compileContracts(contracts['mgmt'][0])
    mgmtContract = initializeContractFactory(web3, compiledContract, contracts['mgmt']
    [0] + ':' + contracts['mgmt '][1],  mgmtContractAddress)

    if web3.eth.getBalance(actor) < requiredGasForSetting * gasPrice:
        print("No enough funds to send transaction")
        return

    unlockAccount(web3, actor, data["password"])

    try:
        txHash = mgmtContract.functions.setFee(serviceFee).transact(tx)
    except ValueError:
        print("No permissions to change the service fee")
        return

    receipt = wait_for_transaction_receipt(web3, txHash)
    if receipt.status == 1:
        print("Updated successfully")


def release(_timeout):
    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    mgmtContractAddress = data['mgmtContract']

    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return
```

```
    actor = data["account"]

    tx = {'from': actor, 'gasPrice': gasPrice}

    compiledContracts = compileContracts([contracts['mgmt'][0], contracts['battery'][0]])
    mgmtContract = initializeContractFactory(web3, compiledContracts, contracts['mgmt'][0]
     + ':' + ][1],
    contracts['mgmt
    mgmtContractAddress)
    batteryMgmtContractAddress = mgmtContract.functions.batteryManagement().call()
    batteryMgmtContract = initializeContractFactory(web3, compiledContracts,
    contracts['battery'][0] + ':' + contracts['battery'][1],
    batteryMgmtContractAddress)
    if web3.eth.getBalance(actor) < requiredGasForSetting * gasPrice:
        print("No enough funds to send transaction")
        return

    unlockAccount(web3, actor, data["password"])

    try:
        txHash = batteryMgmtContract.functions.setTimeoutThreshold(_timeout).transact(tx)
    except ValueError:
        print("No permissions to change the service fee")
        return

    receipt = wait_for_transaction_receipt(web3, txHash)
    if receipt.status == 1:
        print("Updated successfully")


def create_parser():
    parser = argparse.ArgumentParser(
        description='Service provider tool',
        epilog="""
It is expected that Web3 provider specified by WEB3_PROVIDER_URI
environment variable. E.g.
WEB3_PROVIDER_URI=file:///path/to/node/rpc-json/file.ipc
WEB3_PROVIDER_URI=http://192.168.1.2:8545
"""
    )

    parser.add_argument(
        '--new', type=str, required=False,
        help='Add new software developer'
    )

    parser.add_argument(
        '--setup', type=float, required=False,
        help='Deploy contract(s) to the chain. Set fee (in ether) for handling one
        vendor registration request'
    )

    parser.add_argument(
        '--setfee', type=float, required=False,
        help='Set fee (in ether) for handling one vendor registration request'
    )

    parser.add_argument(
        '--release', type=int, required=False,
        help='Set timeout (in seconds) for unblocking the tokens'
```

```
    )

    return parser


def main():
    parser = create_parser()
    args = parser.parse_args()

    if args.new:
        print(createNewAccount(web3, args.new, accountDatabaseName))
    elif args.setup:
        setup(args.setup)
    elif args.setfee:
        setFee(args.setfee)
    elif args.release:
        release(args.release)
    else:
        sys.exit("No parameters provided")


if __name__ == '__main__':
    main()
```

./test_script.py

```
import subprocess, json, os, sys

accounts = {}
contracts = {}
batteries = []

password = 'hackathon180317'
setup = 'setup.py'
vendor = 'vendor.py'
car = 'car.py'
scenter = 'scenter.py'
fauc = 'testnet/faucet.py'

py = ['python']

def faucet(_account):
    print()
    args = py + [fauc, _account]
    print(*args)
    subprocess.Popen(args, stdout=subprocess.PIPE).communicate()

def createManagementAccount():
    global accounts
    args = py + [setup, '--new', password]
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    carry = out[0].decode().splitlines()
    accounts['managementAccount'] = carry[0]
    faucet(carry[0])

def setupSystem(_fee):
    global contracts
    print()
    args = py + [setup, '--setup',  _fee]
```

```
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    carry = out[0].decode().splitlines()
    contracts['mgmtContractAddress'] = carry[0].split(': ')[1]
    contracts['serviceProviderWalletAddress'] = carry[1].split(': ')[1]
    contracts['currencyTokenContractAddress'] = carry[2].split(': ')[1]

def createVendorAccount():
    print()
    args = py + [vendor, '--new', password]
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    carry = out[0].decode().splitlines()
    accounts['vendorAccount'] = carry[0]
    faucet(carry[0])

def registerNewVendor():
    global accounts
    print()
    args = py + [vendor, '--reg', 'InnoCars', '0.1']
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    accounts['vendorId'] = out[0].decode().splitlines()[1].split(': ')[1]

def createNewBatteries(_batNunber):
    global batteries
    print()
    args = py + [vendor, '--bat', _batNunber]
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    out = out[0].decode().splitlines()
    for i in range(len(out)):
        batteries.append(out[i].split(': ')[1])

def createNewScenterAccount():
    global accounts
    print()
    args = py + [scenter, '--new', password]
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    out = out[0].decode().splitlines()
    faucet(out[0])
    accounts['scenterAccount'] = out[0]

def registerScenterAccount():
    print()
    args = py + [scenter, '--reg']
    print(*args)
    subprocess.Popen(args, stdout=subprocess.PIPE).communicate()

def createNewCarAccount():
    global accounts
    print()
    args = py + [car, '--new']
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    out = out[0].decode().splitlines()
    faucet(out[0])
    accounts['carAccount'] = out[0]
```

```
def registerCarAccount():
    print()
    args = py + [car, '--reg']
    print(*args)
    subprocess.Popen(args, stdout=subprocess.PIPE).communicate()

def saleNewBattery(_batteryId, _newOwner):
    print()
    args = py + [vendor, '--owner', _batteryId, _newOwner]
    print(*args)
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    print(out[0].decode())

def createContract(_newBattery, _oldBattery, _serviceFee):
    print()
    args = py + [scenter, '--contract', f'firmware/{_newBattery[2:10]}.py',
    f'firmware/{_oldBattery[2: print(*args) 10]}.py', accounts['carAccount'], _serviceFee]
    out = subprocess.Popen(args, stdout=subprocess.PIPE).communicate()
    print(out[0].decode())


def main():
    createManagementAccount()
    setupSystem('0.0001')
    createVendorAccount()
    registerNewVendor()
    createNewBatteries('5')
    createNewScenterAccount()
    registerScenterAccount()
    createNewCarAccount()
    registerCarAccount()
    saleNewBattery(batteries[0], accounts['scenterAccount'])
    saleNewBattery(batteries[1], accounts['carAccount'])
    createContract(batteries[0], batteries[1], '1')

    print()
    for i in list(contracts.keys()):
        print(i + ': ' + contracts[i])
    print()
    for i in list(accounts.keys()):
        print(i + ': ' + accounts[i])
    print()
    print("Batteries:")
    for i in batteries:
        print(i)

if __name__ == '__main__':
    # contracts = {'mgmtContractAddress': '0x50835cc7c28Da19878415758e6F562D46c324F4F',
    #     'serviceProviderWalletAddress': '0x2c5b6db368F2df6D5a6902f45C9A006f46DF7bA1',
    #     'currencyTokenContractAddress': '0x1c741C179aD7611658b6cd7f9296ef301578FebF'}
    #
    # accounts = {'managementAccount': '0x806FA8996d43316D8be233BEF54e6b1CeDb9Ed79',
    #             'vendorAccount': '0x7d2B0DcF57209bc8495F2a9b1bE41E4A9864B40A',
    #             'vendorId': '875e6b38',
    #             'scenterAccount': '0xdB4df1cA65967b5455FD6B3eDE01fcdEA563F3f6',
    #             'carAccount': '0x433Eac9aE288ac86eF548512bC2db29Be41BBf1B'}
    #
    # batteries = ['0xaA3AbBee1718bFb14407e7e848469F5ea3FFbE14',
    #             '0xF84e7ddcD1bE0452d3B39fab0a2B1b6c23a98dBC',
    #             '0xeC891e1Ce6fb92b33e87a7d2e9961F3B83711a96',
    #             '0xa86d1Ce0Cb1B5A4be27Ee44d9e3a335bf542e634',
```

```
    #               '0xd4f605B79874851A57E34f165Aa636F089bF0cE6']
```

./utils.py

```
from json import dump, load
import os

# install solc by `pip install py-solc`
from solc import compile_files


def delHexPrefix(_s: str):
    if _s[:2] == '0x':
        _s = _s[2:]
    return _s


def writeDataBase(_data, _file):
    with open(_file, 'w') as out:
        dump(_data, out)


def createNewAccount(_w3, _password, _file):
    if os.path.exists(_file):
        os.remove(_file)
    acc = _w3.personal.newAccount(_password)
    data = {"account": acc, "password": _password}
    writeDataBase(data, _file)
    return data['account']


def unlockAccount(_w3, _actor, _pwd):
    _w3.personal.unlockAccount(_actor, _pwd, 60)


def openDataBase(_file):
    if os.path.exists(_file):
        with open(_file) as file:
            return load(file)
    else:
        return None


def getAccountFromDB(_file):
    data = openDataBase(_file)
    if data is not None:
        return data["account"]
    else:
        return None


# TODO: ask an oracle for gas price
def getActualGasPrice(_w3):
    return _w3.toWei(1, 'gwei')


def compileContracts(_files):
    t = type(_files)
    if t == str:
        contracts = compile_files([_files])
```

```
    if t == list:
        contracts = compile_files(_files)
    return contracts


def initializeContractFactory(_w3, _compiledContracts, _key, _address=None):
    if _address == None:
        contract = _w3.eth.contract(
            abi=_compiledContracts[_key]['abi'],
            bytecode=_compiledContracts[_key]['bin']
        )
    else:
        contract = _w3.eth.contract(
            abi=_compiledContracts[_key]['abi'],
            address=_address
        )
    return contract


def runCmd(_args, _workdir=''):
    executed = False
    out = ""
    ext_args = _args[:]
    # this will initialize python_intrp when runCmd() will be run first time
    if not hasattr(runCmd, "python_intrp"):
        runCmd.python_intrp = getPythonInterpreter()
    ext_args.insert(0, runCmd.python_intrp)

    if len(_workdir) > 0:
        cwd = os.getcwd()
        os.chdir(_workdir)

    try:
        tmpout = check_output(ext_args, stderr=STDOUT, timeout=50)
    except CalledProcessError as ret:
        tmpout = ret.stdout
    except TimeoutExpired:
        tmpout = ''

    if len(_workdir) > 0:
        os.chdir(cwd)
    if len(tmpout) > 0:
        executed = True
        out = tmpout.decode().splitlines()

    return (executed, out)


accountDatabaseName = 'car.json'
mgmtContractDatabaseName = 'database.json'


def ManagementContract(_w3):
    mgmtContractDatabaseName = 'database.json'
    data = openDataBase(mgmtContractDatabaseName)

    contractFileMgmt = 'contracts/ManagementContract.sol'
    mgmtContractName = 'ManagementContract'
    mgmtContractABI = compileContracts(contractFileMgmt)[f'{contractFileMgmt}:
    {mgmtContractName}']['abi'] _address = data['mgmtContract']
```

```
    return _w3.eth.contract(abi=mgmtContractABI, address=_address)


def BatteryContract(_w3, _address):
    contractFileBat = 'contracts/BatteryManagement.sol'
    BatContractName = 'BatteryManagement'
    batteryMgmtABI = compileContracts(contractFileBat)[f'{contractFileBat}:
    {BatContractName}']['abi']
    return _w3.eth.contract(abi=batteryMgmtABI, address=_address)

def DealContract(_w3, _address):
    contractFileDeal = 'contracts/DealInterface.sol'
    DealContractName = 'DealInterface'
    DealContractABI = compileContracts(contractFileDeal)[f'{contractFileDeal}:
    {DealContractName}']['abi']
    return _w3.eth.contract(abi=DealContractABI, address=_address)
```

./vendor.py

```
# install web3 >=4.0.0 by `pip install --upgrade 'web3==4.0.0b11'`
from web3 import Web3, IPCProvider
from web3.middleware import geth_poa_middleware
from web3.utils.transactions import wait_for_transaction_receipt

# install solc by `pip install py-solc`
from solc import compile_files

from json import load, dump, JSONDecodeError
import argparse
import sys, os
from utils import ManagementContract, BatteryContract, delHexPrefix

web3 = Web3()

# configure provider to work with PoA chains
web3.middleware_stack.inject(geth_poa_middleware, layer=0)
try:
    with open('account.json') as file:
        config = load(file)
    actor = web3.toChecksumAddress(config['account'])
    gasPrice = web3.toWei(1, 'gwei')
    txTmpl = {'from': actor, 'gasPrice': gasPrice}
except:
    pass

databaseFile = 'database.json'

try:
    with open(databaseFile) as file:
        database = load(file)
except FileNotFoundError:
    database = {}
except JSONDecodeError:
    database = {}


def unlockAccount(_w3):
    _w3.personal.unlockAccount(actor, config['password'], 300)


def compileContracts(_files):
```

```
    t = type(_files)
    if t == str:
        contracts = compile_files([_files])
    if t == list:
        contracts = compile_files(_files)
    return contracts


def initializeContractFactory(_w3, _compiledContracts, _key, _address=None):
    if _address == None:
        contract = _w3.eth.contract(
            abi=_compiledContracts[_key]['abi'],
            bytecode=_compiledContracts[_key]['bin']
        )
    else:
        contract = _w3.eth.contract(
            abi=_compiledContracts[_key]['abi'],
            address=_address
        )
    return contract


def writeDatabase(databaseFile='database.json'):
    with open(databaseFile, 'w') as file:
        dump(database, file)


def safetySendTx(_w3, _tx, _args):
    mgmtContract = ManagementContract(_w3)
    _gasLimit = _w3.eth.getBlock('latest')['gasLimit']
    _batSuccess = []
    # _args is list of batIds
    start, end = 0, len(_args)
    flag = True

    while flag:
        try:
            Gas = mgmtContract.functions.registerBatteries(_args[start:end]).estimateGas(_tx)

            if Gas < int(_gasLimit / 2):
                flag = False
        except:
            pass
        end //= 2
    end = max(end, 1)
    for it in range(0, len(_args), end):
        _to = min(len(_args), it + end)
        txHash = mgmtContract.functions.registerBatteries(_args[it:_to]).transact(_tx)
        _tx['value'] = 0
        receipt = wait_for_transaction_receipt(_w3, txHash)
        if receipt.status == 1:
            _batSuccess += _args[it:_to]
        else:
            break

    # Total
    status = 0 if (len(_batSuccess) == 0) else 1
    status = 2 if (len(_batSuccess) == len(_args)) else status

    return status, _batSuccess
```

```
def regfee(_w3):
    mgmtContract = ManagementContract(_w3)
    result = mgmtContract.functions.registrationDeposit().call()
    return _w3.fromWei(result, 'ether')


def batfee(_w3):
    tx = txTmpl
    mgmtContract = ManagementContract(_w3)
    result = mgmtContract.functions.batteryFee().call({'from': tx['from']})
    return _w3.fromWei(result, 'ether')


def deposit(_w3):
    tx = txTmpl
    mgmtContract = ManagementContract(_w3)
    if mgmtContract.functions.registeredVendorAddress(tx['from']).call():
        result = mgmtContract.functions.vendorDeposit(tx['from']).call()
    else:
        return -1
    return _w3.fromWei(result, 'ether')


def registerVendor(_w3, _name, _value=1):
    mgmtContract = ManagementContract(_w3)
    tx = txTmpl
    if _value < mgmtContract.functions.registrationDeposit().call():
        return "Failed. Payment is low."
    elif _value > _w3.eth.getBalance(tx['from']):
        return "Failed. No enough funds to deposit."
    elif mgmtContract.functions.registeredVendorAddress(tx['from']).call():
        return "Failed. The vendor address already used."
    else:
        try:
            tx['value'] = _value
            txHash = mgmtContract.functions.registerVendor(_name.encode()).transact(tx)
            receipt = wait_for_transaction_receipt(_w3, txHash)
            if receipt.status == 1:
                return mgmtContract.events.Vendor().processReceipt(receipt)[0]['args']
        ['tokenId']
        except ValueError:
            return "Failed. The vendor name is not unique."

def regBat(_w3, _count, _value=0):
    _tx = txTmpl
    _tx['value'] = _value
    _batkeys = []
    _args = []
    if deposit(_w3) + web3.fromWei(_value, "ether") < batfee(_w3) * _count:
        return -1
    for i in range(_count):
        _privKey = _w3.sha3(os.urandom(20))
        _batkeys.append((_privKey, _w3.eth.account.privateKeyToAccount(_privKey).address))
    for i in range(len(_batkeys)):
        _args.append(_w3.toBytes(hexstr=_batkeys[i][1]))

    result, _batkeysSuccess = safetySendTx(_w3, _tx, _args)
    form = 'Created battery with ID: {}'
```

```
    resultText = ''
    if result != 0:
        for _batId in _batkeys:
            createBatteryScript(_batId)
            print(form.format(_batId[1]))
    return result


def createBatteryScript(_key):
    _path = './battery_template.py'
    with open(_path, 'r') as file:
        code = file.read()
    code = code.replace("cafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafe",
                        delHexPrefix(web3.toHex(_key[0])), 1)
    with open('./firmware/' + delHexPrefix(_key[1][2:10]) + '.py', 'w') as file:
        file.write(code)
    return True


def newowner(_w3, _batids, _newOwner, pacTransf=False):
    _tx = txTmpl
    mgmtContract = ManagementContract(_w3)
    _batteryContract = mgmtContract.functions.batteryManagement().call()
    _BatContract = BatteryContract(_w3, _batteryContract)
    if pacTransf:
        with open(_batids, 'r') as file:
            _batids = file.read().split('\n')
    else:
        _batids = [_batids]
    _args = []
    for i in range(len(_batids)):
        _args.append(_w3.toBytes(hexstr=_batids[i]))
    if mgmtContract.functions.serviceCenters(_newOwner).call() or mgmtContract.functions.
    cars(_newOwner).call():

        try:
            txHash = _BatContract.functions.transfer(_newOwner, _args).transact(_tx)
            receipt = wait_for_transaction_receipt(_w3, txHash)
            result = receipt.status
        except:
            result = 0
    else:
        result = 0

    return result == 1

def create_parser():
    parser = argparse.ArgumentParser(
        description='Simple management tool for didgital assests',
        epilog="""
It is expected that Web3 provider specified by WEB3_PROVIDER_URI
environment variable. E.g.
WEB3_PROVIDER_URI=file:///path/to/node/rpc-json/file.ipc
WEB3_PROVIDER_URI=http://192.168.1.2:8545
"""
    )

    parser.add_argument(
        '--regfee', action='store_true', required=False,
        help='Get fee (in ether) for registration request'
    )
```

```
    parser.add_argument(
        '--batfee', action='store_true', required=False,
        help='Get fee (in ether) for registration request'
    )

    parser.add_argument(
        '--new', nargs=1, required=False,
        help='Add address to the account.json'
    )

    parser.add_argument(
        '--reg', nargs=2, required=False,
        help='Register a vendor'
    )
    parser.add_argument(
        '--bat', nargs="+", required=False,
        help='Register batteries'
    )

    parser.add_argument(
        '--owner', nargs=2, required=False,
        help='Transfer batteries'
    )

    parser.add_argument(
        '--deposit', action='store_true', required=False,
        help='Get deposit for vendor account'
    )

    return parser


def main():
    parser = create_parser()

    args = parser.parse_args()

    web3.eth.enable_unaudited_features()

    # print(args)
    if args.new:
        global database
        database = {}
        database['account'] = web3.personal.newAccount(args.new[0])
        database['password'] = args.new[0]
        if len(database['account']) == 42:
            writeDatabase('account.json')
            print('{}'.format(database['account']))
        else:
            sys.exit("Cannot get address")
    elif args.regfee:
        if 'mgmtContract' in database:
            unlockAccount(web3)
            result = regfee(web3)
            print('Vendor registration fee: {}'.format(result))
    elif args.batfee:
        if 'mgmtContract' in database:
            unlockAccount(web3)
            result = batfee(web3)
            print('Production fee per one battery: {}'.format(result))
```

```
    elif args.reg:
        if 'mgmtContract' in database:
            unlockAccount(web3)
            result = registerVendor(web3, args.reg[0], web3.toWei(args.reg[1], 'ether'))
            if isinstance(result, bytes):
                print('Success.\nVendor ID: {}'.format(delHexPrefix(web3.toHex(result))))
            else:
                sys.exit(result)

        else:
            sys.exit("Management contract address not found in the database")
    elif args.bat:
        if 'mgmtContract' in database:
            unlockAccount(web3)
            if len(args.bat) > 1:
                result = regBat(web3, int(args.bat[0]), web3.toWei(args.bat[1], 'ether'))
            else:
                result = regBat(web3, int(args.bat[0]))
            # if result >= 1:
            #     print('Success')
            # else:
            #     print('Failed')
            if result < 1:
                print('Failed')
        else:
            sys.exit("Management contract address not found in the database")

    elif args.owner:
        if 'mgmtContract' in database:
            unlockAccount(web3)
            isFilePathExist = os.access(args.owner[0], os.F_OK)
            if isFilePathExist:
                result = newowner(web3, args.owner[0], args.owner[1], pacTransf=True)
            else:
                result = newowner(web3, args.owner[0], args.owner[1], pacTransf=False)
            if result:
                print('Success')
            else:
                print('Failed. Not allowed to change ownership.')
    elif args.deposit:
        if 'mgmtContract' in database:
            unlockAccount(web3)
            result = deposit(web3)
            if result != -1:
                print('Deposit: {}'.format(result))
            else:
                print("Vendor account is not registered.")
    else:
        sys.exit("No parameters provided")


if __name__ == '__main__':
    main()
```

./verify_battery.py

```
from utils import openDataBase, ManagementContract, BatteryContract, DealContract
import json, subprocess, os
```

```
def _delHexPrefix(_s: str):
    if _s[:2] == '0x':
        _s = _s[2:]
    return _s

def batteryExists(_path):
    return os.path.exists(_path)


def verify(_w3, _path):
    n, t, v, r, s = return_ntvrs_from(_path)
    mgmtContract = ManagementContract(_w3)
    validity, vendorAddress = _verifyBatteryByContract(_w3, mgmtContract, n, t, v, r, s)

    vendorId = _w3.toHex(mgmtContract.functions.vendorId(vendorAddress).call())
    vendorName = mgmtContract.functions.vendorNames(vendorId).call()
    if validity == 0:
        return validity, n, _delHexPrefix(vendorId), vendorName.decode()
    else:
        return [validity]


def return_ntvrs_from(_path):
    args = ['python', _path, '--getJson']

    batteryOut = subprocess.Popen(args, stdout=subprocess.PIPE)
    batteryInformation = json.loads(batteryOut.communicate()[0].decode())
    n, t, v, r, s = batteryInformation['n'], batteryInformation['t'],
    batteryInformation['v'], \
    batteryInformation['r'], batteryInformation['s']
    return [n, t, v, r, s]

def _verifyBatteryByContract(_w3, _mgmtContract, _n, _t, _v, _r, _s):
    batteryContractAddress = _mgmtContract.functions.batteryManagement().call()

    batteryContract = BatteryContract(_w3, batteryContractAddress)

    validity, address = batteryContract.functions.verifyBattery(_n, _t, _v, _w3.toBytes
    (hexstr=_r), _w3.toBytes(hexstr=_s)).call()

    return validity, address
```

Критерий определения победителей и призеров заключительного этапа

В заключительном этапе олимпиады баллы участника складываются из двух частей: он получает баллы за индивидуальное решение задач по предметам (математика и информатика) и за командное решение практической задачи.

$$S = S_1 + S_2$$

где S_1 — количество баллов, набранное в рамках индивидуальной части заключительного этапа (максимум 200 баллов); S_2 — количество баллов, набранное в рамках командной части заключительного этапа (максимум 400 баллов).

Критерий определения победителей и призеров:

	Призеры	**Победители**
Набранные баллы	от $97,5$ до 200 баллов	от 200 баллов и выше