Работа победителя/призера заключительного этапа

командной инженерной олимпиады школьников

Олимпиада Национальной технологической инициативы

Профиль «Программная инженерия финансовых технологий»

Плевако Матвей Дмитриевич

Класс: 11

Город: Благовещенск

Школа: Лицей ФБГБОУ ВО БГПУ

Регион: Амурская область

Уникальный номер участника: 8

Команда на заключительном этапе: IT`s more than a team

Параллель: 10-11 класс

Результаты заключительного этапа:

№	индивидуальная			Командная часть				
	Математика	Информатика	итого	1 день	2 день	3 день	общий	результат
8	0	45	45	4	28,5	20	52,5	97,5

Индивидуальная часть

Персональный лист участника с номером 8:

Олимпиада НТИ

ФИО Гилевко Матвей Дмитриевич

Город г. Благовещенск

Школа № лицей БГПУ

Математика

Лист 1:

Командная инженерная олимпиада «Олимпиада НТИ»

Направление _____

Предмет _____

Номер участника 180209

Информатика

Задача A1:

```
n, m = map(int, input().split())

graph = {}

for i in range(m):
    a, b = map(int, input().split())
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)


s_a, s_b = map(int, input().split())

stack = [s_a]
passed = {s_a}

while len(stack) > 0:
    v = stack.pop()
    if v not in graph:
        continue
    for each in graph[v]:
        if each not in passed:
            stack.append(each)
            passed.add(each)


if s_b in passed:
    print("Yes")
else:
    print("No")
```

Задача A2:

```
n, m = map(int, input().split())

graph = {}

for i in range(m):
    a, b = map(int, input().split())
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)


s_a, s_b = map(int, input().split())

stack = [s_a]
passed = {s_a}

while len(stack) > 0:
    v = stack.pop()
    if v not in graph:
```

```
        continue
    for each in graph[v]:
        if each not in passed:
            stack.append(each)
            passed.add(each)


if s_b in passed:
    print("Yes")
else:
    print("No")
```

Задача A Full:

```
n, m = map(int, input().split())

graph = {}

for i in range(m):
    a, b = map(int, input().split())
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)


s_a, s_b = map(int, input().split())

stack = [s_a]
passed = {s_a}

while len(stack) > 0:
    v = stack.pop()
    if v not in graph:
        continue
    for each in graph[v]:
        if each not in passed:
            stack.append(each)
            passed.add(each)


if s_b in passed:
    print("Yes")
else:
    print("No")
```

Задача B1:

```
INF = 10**4

n, m = map(int, input().split())

graph = {}
cost = [[INF for ji in range(n)] for ij in range(n)]

for i in range(m):
    a, b, c = map(int, input().split())
    a -= 1
```

```
    b -= 1
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)
    cost[a][b] = c

s_a, s_b = map(int, input().split())
s_a -= 1
s_b -= 1

q, q0 = map(int, input().split())

for i in range(q):
    a, b, c = map(int, input().split())
    a -= 1
    b -= 1
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)
    cost[a][b] = min(c + q0, cost[a][b])



cost_to_transfer = [INF for i in range(n)]
cost_to_transfer[s_a] = 0

stack = [s_a]
passed = {s_a}

while len(stack) > 0:
    v = stack.pop()
    if v not in graph:
        continue
    for each in graph[v]:
        cost_to_transfer[each] = min(cost_to_transfer[each],
cost_to_transfer[v] + cost[v][each])
        if each not in passed:
            stack.append(each)
            passed.add(each)

if cost_to_transfer[s_b] == INF:
    print(-1)
else:
    print(cost_to_transfer[s_b])
```

Задача B2:

```
INF = 10**100

n, m = map(int, input().split())

graph = {}
cost = [[INF for ji in range(n)] for ij in range(n)]

for i in range(m):
    a, b, c = map(int, input().split())
    a -= 1
```

```
    b -= 1
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)
    cost[a][b] = c

s_a, s_b = map(int, input().split())
s_a -= 1
s_b -= 1

q, q0 = map(int, input().split())

for i in range(q):
    a, b, c = map(int, input().split())
    a -= 1
    b -= 1
    if a not in graph:
        graph[a] = [b]
    elif b not in graph[a]:
        graph[a].append(b)
    cost[a][b] = min(c + q0, cost[a][b])



cost_to_transfer = [INF for i in range(n)]
cost_to_transfer[s_a] = 0

stack = [s_a]
passed = {s_a}

while len(stack) > 0:
    v = stack.pop()
    if v not in graph:
        continue
    for each in graph[v]:
        if cost_to_transfer[each] > cost_to_transfer[v] + cost[v][each]:
            cost_to_transfer[each] = cost_to_transfer[v] + cost[v][each]
            stack.append(each)
        if each not in passed:
            stack.append(each)
            passed.add(each)

if cost_to_transfer[s_b] == INF:
    print(-1)
else:
    print(cost_to_transfer[s_b])
```

Задача B Full:

```
INF = 10 ** 10

from queue import deque


n, m = map(int, input().split())

graph = {}
cost = [[INF for ji in range(n)] for ij in range(n)]
```

```
for i in range(m):
    a, b, c = map(int, input().split())
    a -= 1
    b -= 1
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)
    cost[a][b] = c

s_a, s_b = map(int, input().split())
s_a -= 1
s_b -= 1

q, q0 = map(int, input().split())

for i in range(q):
    a, b, c = map(int, input().split())
    a -= 1
    b -= 1
    if a not in graph:
        graph[a] = [b]
    elif b not in graph[a]:
        graph[a].append(b)
    cost[a][b] = min(c + q0, cost[a][b])

cost_to_transfer = [INF for i in range(n)]
cost_to_transfer[s_a] = 0

stack = deque()
stack.append(s_a)
passed = {s_a}

while len(stack) > 0:
    v = stack.pop()
    if v not in graph:
        continue
    for each in graph[v]:
        cost_to_transfer[each] = min(cost_to_transfer[each],
cost_to_transfer[v] + cost[v][each])
        # if cost_to_transfer[each] > cost_to_transfer[v] + cost[v][each]:
        #     cost_to_transfer[each] = cost_to_transfer[v] + cost[v][each]
        #     if cost_to_transfer[each] != INF:
        #         stack.append(each)
        if each not in passed:
            stack.appendleft(each)
            passed.add(each)

if cost_to_transfer[s_b] == INF:
    print(-1)
else:
    print(cost_to_transfer[s_b])
```

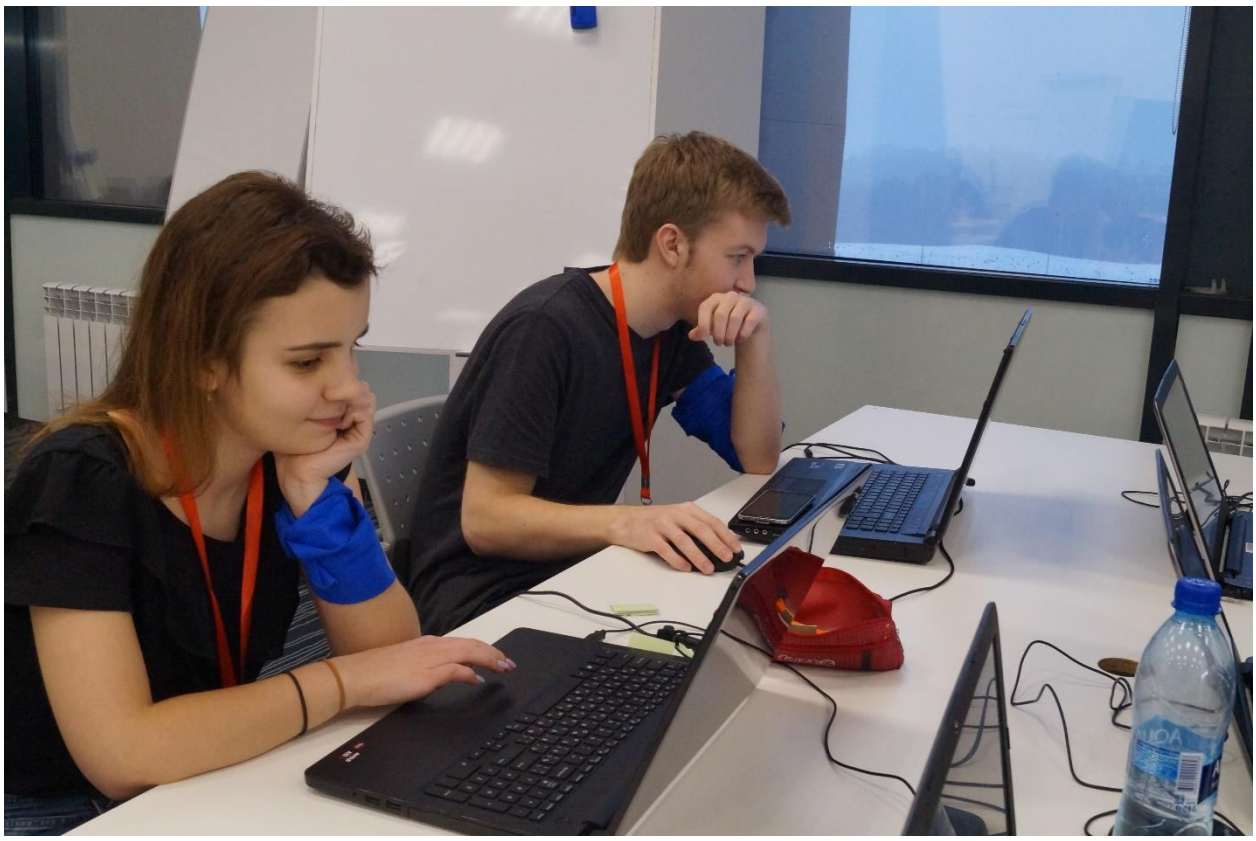
Задача C:

Решение не было приложено.

Командная часть

Результаты были получены в рамках выступления команды: IT`s more than a team

Фотография команды за работой:



Личный состав команды:

- Плевако Матвей Дмитриевич
- Лепёхина Анфиса Владимировна

Протокол выполнения заданий:

Первый этап:

№ задания	**Критерий**	**Возможные баллы**	Полученные баллы
US-001	Регистрация аккаунта компании производителя ПО	2	0
US-002	Регистрация контрактов	12	0

US-005	Создание аккаунта производителя батарей	2	2
US-007	Регистрация нового производителя (только AC-007-01)	8	0
US-010	Поштучная регистрация производимых батарей	6	0
US-020	Создание аккаунта сервисного центра	2	2
US-021	Регистрация аккаунта сервисного центра	6	0
US-022	Создание аккаунта электромобиля	2	0
US-023	Получение аккаунта электромобиля	2	0
US-024	Регистрация аккаунта электромобиля	6	0
Всего за этап		**48**	**4**

Второй этап:

№ задания	**Критерий**	**Возможные баллы**	**Полученные баллы**
US-003	Установка сборов за регистрацию одной батареи	6	6
US-006	Получение информации о необходимом депозите за регистрацию производителя батарей	4	4
US-007	Регистрация нового производителя (все)	4	4
US-008	Защита от повторной регистрации нового производителя	4	4
US-009	Получение информации о сборе за регистрацию одной батареи	4	4
US-011	Регистрация батарей с закрепленным сбором за регистрацию батарей	4	4
US-012	Получение размера остатка по депозиту	4	4
US-013	Пакетная регистрация производимых батарей	6	6
US-015	Поштучная продажа новых батарей	6	6
US-017	Генерация прошивок аппаратных ключей батарей	4	4

US-018	Изменение количества заряда батареи	3	3
US-019	Получение информации о батарее из прошивки	8	8
US-025	Проверка подлинности батареи сервисным центром	9	0
US-026	Проверка подлинности батареи электромобилем	9	0
US-027	Получение адреса контракта для замены батареи	16	0
US-029	Получение информации, что контракт замены в ожидании подтверждения	4	0
US-030	Получение условий контракта	8	0
US-031	Подтверждение согласия с условиями контракта	16	0
US-032	Подтверждение готовности к физической замене батарей	4	0
US-033	Подтверждение контракта	16	0
Всего за этап		**139**	**57**

Третий этап:

№ задания	**Критерий**	**Возможные баллы**	**Полученные баллы**
US-004	Регистрация контрактов с постоянными адресами	20	0
US-014	Безопасная пакетная регистрация производимых батарей	6	0
US-016	Пакетная продажа новых батарей	6	0
US-028	Отправка запроса на отмену сделки	8	0
US-034	Установка времени запрета разблокировки токенов	6	0
US-035	Запрос на разблокирование токенов	10	0
US-036	Отсутствие подтверждения замены батарей для завершения сделки	10	0
US-037	метод setBatteryManagementContract()	2	2
US-038	метод registerVendor()	4	4
US-039	метод registerBatteries()	3	3
US-040	метод setFee()	2	2
US-041	метод registerServiceCenter()	2	2
US-042	метод registerCar()	2	2

US-043	метод createBattery()	3	3
US-044	метод transfer() с указанием одного идентификатора батареи	2	2
US-045	метод delegatedApprove()	4	0
US-046	метод initiateDeal()	6	0
US-047	метод setTimeoutThreshold()	2	0
US-048	метод agreeToDeal()	6	0
US-050	метод releaseTokensByCar()	4	0
US-051	метод releaseTokensByServiceCenter()	4	0
US-052	метод cancelDeal()	4	0
US-053	Передача прав на батарею новому владельцу	8	0
US-054	Отложенная передача прав владения	12	0
US-055	Регистрация прав владения	8	0
US-056	Оптимизация по использованию вычислительных ресурсов	33	0
US-057	Оптимизация комиссии за валидацию транзакций	6	0
US-058	Сбор статистики	30	0
Всего за этап		**213**	**20**

Программа команды для решения задач:

Задача Car:

```
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

import argparse
import sys

from utils import writeDataBase, openDataBase, getActualGasPrice,
compileContracts, initializeContractFactory

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

# contract files
contracts = {'mgmt':   ('contracts/ManagementContract.sol',
'ManagementContract')}

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():
    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access management contract database")
        return

    mgmtContractAddress = data['mgmtContract']

    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    privateKey = data['key']

    compiledContract = compileContracts(contracts['mgmt'][0])
    mgmtContract = initializeContractFactory(web3, compiledContract,
contracts['mgmt'][0] + ':' + contracts['mgmt'][1],
```

```
                                                mgmtContractAddress)

    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')


def Manager():
    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access management contract database")
        return
    mgmtContractAddress = data['mgmtContract']
    compiledContract = compileContracts(contracts['mgmt'][0])
    mgmtContract = initializeContractFactory(web3, compiledContract,
contracts['mgmt'][0] + ':' + contracts['mgmt'][1],
                                                mgmtContractAddress)
    return mgmtContract

def Bat_Manager():
    contractFileBatm = './contracts/BatteryManagement.sol'
    batManager = 'BatteryManagement'

    _compiled = compileContracts(contractFileBatm)
    _BatManager = initializeContractFactory(web3, _compiled,
contractFileBatm + ":" + batManager,

Manager().functions.batteryManagement().call() )
    return _BatManager

def verify(path):
    from subprocess import check_output
    from web3.auto import w3

    data = check_output(('python3 ' + path[0] + ' --
get').split()).decode('utf-8').split()
    n, t, v, r, s = data
    v = hex(int(v))[2:]
    n = int(n)
    t = int(t)
    signature = '0x' + r + s + v
    msg = n * 2 ** 32 + t
    msg = "\x19Ethereum Signed Message:\n" + str(msg)
    try:
```

```
        addr = w3.eth.account.recoverMessage(text=msg,
signature=signature)
        vendor =
Bat_Manager().functions.bat_to_vendor(addr.lower()).call()
        id = Manager().functions.vendorId(vendor).call()
        name = Manager().functions.vendorNames(id).call()

        assert vendor != "0x0000000000000000000000000000000000000000"
        if Bat_Manager().functions.is_used(signature).call():
            print("Battery with the same status already replaced. Probably
the battery forged.")
            exit()
        print("Verified successfully.")
        print("Total charges: " + str(n))
        print("Vendor ID: " + id.hex())
        print("Vendor Name: " + name.decode('utf-8'))
    except:
        print("Verification failed. Probably the battery forged.")
        exit()

def create_parser():
    parser = argparse.ArgumentParser(
        description='Autonomous Ground Vehicle software',
        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='Generate a new account for the particular AGV'
    )

    parser.add_argument(
        '--account', action='store_true', required=False,
        help='Get identificator (Ethereum address) of AGV from the private
key stored in car.json'
    )

    parser.add_argument(
        '--reg', action='store_true', required=False,
        help='Register the vehicle in the chain'
    )

    return parser

def main():
    parser = create_parser()
    args = parser.parse_args()

    if args.new:
        new()
    elif args.account:
        account()
    elif args.reg:
        reg()
    else:
        sys.exit("No parameters provided")
```

```
if __name__ == '__main__':
    main()
```

Задача Scenter:

```
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

import argparse
import sys

from utils import writeDataBase, openDataBase, getActualGasPrice,
compileContracts, initializeContractFactory

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

# contract files
contracts = {'mgmt':   ('contracts/ManagementContract.sol',
'ManagementContract')}

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():
    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access management contract database")
        return

    mgmtContractAddress = data['mgmtContract']

    data = openDataBase(accountDatabaseName)
    if data is None:
        print("Cannot access account database")
        return

    privateKey = data['key']

    compiledContract = compileContracts(contracts['mgmt'][0])
    mgmtContract = initializeContractFactory(web3, compiledContract,
contracts['mgmt'][0] + ':' + contracts['mgmt'][1],
                                             mgmtContractAddress)

    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')


def Manager():
    data = openDataBase(mgmtContractDatabaseName)
    if data is None:
        print("Cannot access management contract database")
        return
    mgmtContractAddress = data['mgmtContract']
```

```
    compiledContract = compileContracts(contracts['mgmt'][0])
    mgmtContract = initializeContractFactory(web3, compiledContract,
contracts['mgmt'][0] + ':' + contracts['mgmt'][1],
                                             mgmtContractAddress)
    return mgmtContract

def Bat_Manager():
    contractFileBatm = './contracts/BatteryManagement.sol'
    batManager = 'BatteryManagement'

    _compiled = compileContracts(contractFileBatm)
    _BatManager = initializeContractFactory(web3, _compiled,
contractFileBatm + ":" + batManager,

Manager().functions.batteryManagement().call() )
    return _BatManager

def verify(path):
    from subprocess import check_output
    from web3.auto import w3

    data = check_output(('python3 ' + path[0] + ' --
get').split()).decode('utf-8').split()
    n, t, v, r, s = data
    v = hex(int(v))[2:]
    n = int(n)
    t = int(t)
    signature = '0x' + r + s + v
    msg = n * 2 ** 32 + t
    msg = "\x19Ethereum Signed Message:\n" + str(msg)
    try:
        addr = w3.eth.account.recoverMessage(text=msg,
signature=signature)
        vendor =
Bat_Manager().functions.bat_to_vendor(addr.lower()).call()
        id = Manager().functions.vendorId(vendor).call()
        name = Manager().functions.vendorNames(id).call()

        assert vendor != "0x0000000000000000000000000000000000000000"
        if Bat_Manager().functions.is_used(signature).call():
            print("Battery with the same status already replaced. Probably
the battery forged.")
            exit()
        print("Verified successfully.")
        print("Total charges: " + str(n))
        print("Vendor ID: " + id.hex())
        print("Vendor Name: " + name.decode('utf-8'))
    except:
        print("Verification failed. Probably the battery forged.")
        exit()

def create_parser():
    parser = argparse.ArgumentParser(
        description='Autonomous Ground Vehicle software',
        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='Generate a new account for the particular AGV'
    )

    parser.add_argument(
        '--account', action='store_true', required=False,
        help='Get identificator (Ethereum address) of AGV from the private
key stored in car.json'
    )

    parser.add_argument(
        '--reg', action='store_true', required=False,
        help='Register the vehicle in the chain'
    )

    return parser

def main():
    parser = create_parser()
    args = parser.parse_args()

    if args.new:
        new()
    elif args.account:
        account()
    elif args.reg:
        reg()
    else:
        sys.exit("No parameters provided")


if __name__ == '__main__':
    main()
```

Задача Send_car:

```
from sys import argv
from web3 import Web3
import json
from subprocess import check_output
from shlex import split

args = argv

def execute(comand):
    print("******")
    print("******")
    try:
        response = check_output(['python3'] +
split(comand)).rstrip(b'\n').decode('utf-8')
        print(response)
        return True
    except:
        return False

w3 = Web3()
if not w3.isConnected():
```

```
    print('No connection to node')
    exit()

with open('car.json') as file:
    account = json.loads(file.read())
    account_address = str(check_output(['python3', 'car.py', '--
account'])[:-1])[2:-1]
    print(account_address)


w3.eth.sendTransaction({'from':w3.eth.accounts[0], 'to': account_address,
'value':100 * 10**18, 'gas':10**5})
print('Sended')
```

Задача Send_money_to_json_acc:

```
from sys import argv
from web3 import Web3
import json

args = argv

w3 = Web3()
if not w3.isConnected():
    print('No connection to node')
    exit()

with open('account.json') as file:
    account = json.loads(file.read())
    account_address = account['account']

w3.eth.sendTransaction({'from':w3.eth.accounts[0], 'to': account_address,
'value':100 * 10**18, 'gas':10**5})
print('Sended')
```

Задача Send_scenter:

```
from sys import argv
from web3 import Web3
import json

args = argv

w3 = Web3()
if not w3.isConnected():
    print('No connection to node')
    exit()

with open('scenter.json') as file:
    account = json.loads(file.read())
    account_address = account["account"]

w3.eth.sendTransaction({'from': w3.eth.accounts[0], 'to': account_address,
'value': 100 * 10 ** 18, 'gas': 10 ** 5})
print('Sended')
```

Задача Setup:

```
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
from json import load, dump, JSONDecodeError
import argparse
import sys

from utils import createNewAccount, writeDataBase, openDataBase,
unlockAccount, getActualGasPrice
from utils import compileContracts, initializeContractFactory

web3 = Web3()

databaseFile = 'database.json'
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, 'gas':5* 10**5}
except:
    pass

try:
    with open(databaseFile) as file:
        database = load(file)
except FileNotFoundError:
    database = {}
except JSONDecodeError:
    database = {}

contractFileMgmt = './contracts/ManagementContract.sol'
mgmtContractName = 'ManagementContract'

contractFileMgmt = './contracts/ManagementContract.sol'
mgmtContractName = 'ManagementContract'

def ManagementContract(_w3):
    _compiled = compileContracts(contractFileMgmt)
    _mgmtContract = initializeContractFactory(_w3, _compiled,
contractFileMgmt + ":" + mgmtContractName,
                                              database['mgmtContract'])
    return _mgmtContract

# 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)

# 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 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"])

    # 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(batteryMgmtContractAdd
ress).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=' ')

                        return
    print('Contracts deployment and configuration failed')
```

```
def setfee(_serviceFee):
    serviceFee = web3.toWei(_serviceFee, 'ether')
    Manager = ManagementContract(web3)
    try:
        tx = Manager.functions.setFee(serviceFee).transact(txTmpl)
        receipt = wait_for_transaction_receipt(web3, tx)
        assert receipt.status == 1
    except:
        print("No permissions to change the service fee")
        exit()
    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 account for software development company'
    )

    parser.add_argument(
        '--setup', type=float, required=False,
        help='Deploy contract(s) to the chain. Set fee (in ether) for
registration of one battery, which reflects vendor registration fee'
    )

    parser.add_argument(
        '--setfee', type=float, required=False,
        help='set fee for 1 bat'
    )

    return parser

def main():
    parser = create_parser()
    args = parser.parse_args()

    if args.new:
        print(createNewAccount(web3, args.new, accountDatabaseName))
    elif args.setup:
        try:
            setup(args.setup)
        except:
            print("No enough funds to send transaction")
    elif args.setfee:
        setfee(args.setfee)
    else:
        sys.exit("No parameters provided")
```

```
if __name__ == '__main__':
    main()
```

Задача Utils:

```
from json import dump, load
import os

# install solc by `pip install py-solc`
from solc import compile_files

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
```

Задача Vendor:

```
# 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

from json import load, dump, JSONDecodeError
import argparse
import sys, os
import math

from utils import compileContracts, writeDataBase,
initializeContractFactory

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)

contractFileMgmt = './contracts/ManagementContract.sol'
mgmtContractName = 'ManagementContract'

contractFileBatm = './contracts/BatteryManagement.sol'
batManager = 'BatteryManagement'

def ManagementContract(_w3):
    _compiled = compileContracts(contractFileMgmt)
    _mgmtContract = initializeContractFactory(_w3, _compiled,
contractFileMgmt + ":" + mgmtContractName,
                                              database['mgmtContract'])
```

```
    return _mgmtContract

def Bat_Manager(_w3):
    _compiled = compileContracts(contractFileBatm)
    _BatManager = initializeContractFactory(_w3, _compiled,
contractFileBatm + ":" + batManager,

ManagementContract(web3).functions.batteryManagement().call() )
    return _BatManager


def registerVendor(_w3, _name, _value=1):
    mgmtContract = ManagementContract(_w3)
    try:
        not_fee = mgmtContract.functions.isAllowed().call({'from':actor})
        assert not not_fee
    except:
        return "Failed. The vendor address already used."
    tx = txTmpl
    if _value > _w3.eth.getBalance(tx['from']):
        return "Failed. No enough funds to deposit."
    else:
        try:
            tx['value'] = _value
            if _value <
mgmtContract.functions.registrationDeposit().call():
                return "Failed. Payment is low."
            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."

body = """_privkey = '{}'
from sys import argv
from json import loads, dump
import os.path
from web3.auto import w3

if argv[1] == '--charge':
    counter = dict()
    counter['n'] = 1
    if os.path.isfile('{}.json'):
        with open('{}.json') as file:
            counter = loads(file.read())
            counter['n'] += 1
    with open('{}.json', 'w') as file:
        dump(counter, file)

if argv[1] == '--get':


    if not os.path.isfile('{}.json'):
        with open('{}.json', 'w') as file:
            counter = dict()
            counter['n'] = 0
            dump(counter, file)
    with open('{}.json') as file:
```

```
        n = loads(file.read())['n']
    import calendar, time
    t = calendar.timegm(time.gmtime())
    m = n * 2 ** 32 + t
    msg = "\\x19Ethereum Signed Message:\\n" + str(m)

    signed_message = w3.eth.account.sign(message_text=msg,
private_key=_privkey)
    print(n)
    print(t)
    print(signed_message['v'])
    print(hex(signed_message['r'])[2:])
    print(hex(signed_message['s'])[2:])
"""

def regBat(_w3, _count, _value=0):
    _tx = txTmpl
    _tx['value'] = _value
    _batkeys = []
    _args = []

    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]))

    mgmtContract = ManagementContract(_w3)

    if mgmtContract.functions.vendorDeposit(actor).call() <
mgmtContract.functions.calcRegCost(_count).call():
        print("Failed. No enough funds to register object.")
        exit()
    gasLimit = web3.eth.getBlock("latest").gasLimit



    txHash = mgmtContract.functions.registerBatteries(_args).transact(_tx)
    receipt = wait_for_transaction_receipt(_w3, txHash)
    result = receipt.status
    form = 'Created battery with ID: {}'
    if not os.path.exists('firmware'):
        os.makedirs('firmware')
    if result >= 1:
        for batId in _batkeys:
            with
open('firmware/{}.py'.format(delHexPrefix(batId[1]).lower()[:8]), 'w') as
file:
                file.write(body.format(batId[0].hex(),
                                       delHexPrefix(batId[1]).lower()[:8],
                                       delHexPrefix(batId[1]).lower()[:8],
                                       delHexPrefix(batId[1]).lower()[:8],
                                       delHexPrefix(batId[1]).lower()[:8],
                                       delHexPrefix(batId[1]).lower()[:8],

delHexPrefix(batId[1]).lower()[:8]))
            print(form.format(delHexPrefix(batId[1]).lower()))
    else:
        print('Batteries registration failed')
```

```
def delHexPrefix(_s: str):
    if _s[:2] == '0x':
        _s = _s[2:]
    return _s

def getFee():
    Manager = ManagementContract(web3)
    fee = Manager.functions.registrationDeposit().call({'from':actor})
    fee = math.ceil(fee / 10 ** 12)
    fee = fee / 10 ** 6
    fee = ("%.10f" % fee).rstrip('0').rstrip('.')

    print("Vendor registration fee: " + fee)

def BatFee():
    Manager = ManagementContract(web3)
    fee = Manager.functions.batteryFee().call({'from':actor})
    fee = math.ceil(fee / 10 ** 12)
    fee = fee / 10 ** 6
    fee = ("%.10f" % fee).rstrip('0').rstrip('.')

    print("Production fee per one battery: " + fee)

def deposit():
    Manager = ManagementContract(web3)

    not_fee = Manager.functions.isAllowed().call({'from': actor})
    if not not_fee:
        print( "Vendor account is not registered.")
        exit()
    fee = Manager.functions.vendorDeposit(actor).call()
    fee = math.ceil(fee / 10 ** 12)
    fee = fee / 10 ** 6
    fee = ("%.10f" % fee).rstrip('0').rstrip('.')
    print("Deposit: " + fee)


def owner(arg):
    tokenID, newOwner = arg[0], arg[1]
    tokenID = bytes.fromhex(tokenID)
    Bat = Bat_Manager(web3)
    try:
        txHash = Bat.functions.transfer(newOwner,
tokenID).transact({'from': actor, 'gas': 5 * 10 ** 5 })
        receipt = wait_for_transaction_receipt(web3, txHash)
        result = receipt.status
        assert result == 1
    except:
        print("Failed. Not allowed to change ownership.")
        exit()
    print("Success")



def create_parser():
    parser = argparse.ArgumentParser(
        description='Battery vendor management 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', 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(
        '--regfee',  action='store_true', required=False
    )

    parser.add_argument(
        '--batfee', action='store_true', required=False
    )

    parser.add_argument(
        '--deposit', action='store_true', required=False
    )

    parser.add_argument(
        '--owner', nargs=2, required=False
    )

    return parser




def main():
    parser = create_parser()

    args = parser.parse_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(database, 'account.json')
            print('{}'.format(database['account']))
        else:
            sys.exit("Cannot get address")
    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]))
        else:
            sys.exit("Management contract address not found in the
database")
    elif args.regfee:
        if 'mgmtContract' in database:
            getFee()
    elif args.batfee:
        if 'mgmtContract' in database:
            BatFee()
    elif args.deposit:
        if 'mgmtContract' in database:
            deposit()
    elif args.owner:
        owner(args.owner)

    else:
        sys.exit("No parameters provided")

if __name__ == '__main__':
    main()
```

Программы для решения полной финальной задачи нет.