Работа победителя/призера заключительного этапа

командной инженерной олимпиады школьников

Олимпиада Национальной технологической инициативы

Профиль «Программная инженерия финансовых технологий»

Байков Аяз Ильдарович

Класс: 11

Город: Уфа

Школа: ГБОУ РИЛИ

Регион: Республика Башкортостан

Уникальный номер участника: 7

Команда на заключительном этапе: Deus Vult

Параллель: 10-11 класс

Результаты заключительного этапа:

№	индивидуальная			Командная часть				
	Математика	Информатика	итого	1 день	2 день	3 день	общий	результат
7	6	69,38	75,38	4	3	16	23	98,38

Индивидуальная часть

Персональный лист участника с номером 7:

Олимпиада НТИ

ФИО Байков Аяз Ильдарович

Город Уфа

Школа № ГБОУ РГИЛИ

Математика

Лист 1:

Командная инженерная олимпиада «Олимпиада НТИ»

Направление Физтех

Предмет Математика

Номер участника 1802005

№ 2

а) ~~$\beta = p \cdot \varepsilon$~~.

Боб получит ~~такой~~ i бит равным 0 в двух случаях:

1) Когда у Алисы i бит равен 0 и после прохождение кабель бит не поменяется. Вероятность такого исхода: $p \cdot (1-\varepsilon)$

2) Когда у Алисы i бит равен 1 и после прохождения кабеля бит поменяется. Вероятность такого исхода: $(1-p) \cdot \varepsilon$.

Значит:

$\beta = p \cdot (1-\varepsilon) + (1-p) \cdot \varepsilon$.

$P_{Боб}(0) = p \cdot (1-\varepsilon) + (1-p) \cdot \varepsilon$

$P_{Боб}(1) = 1 - p \cdot (1-\varepsilon) \pm (1-p) \cdot \varepsilon$?

Ответ: $\beta = p \cdot (1-\varepsilon) + (1-p) \cdot \varepsilon$.

Информатика

Задача A1:

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

using namespace std;

ifstream in("input.txt");
ofstream out("output.txt");

vector<vector<int>>a;
vector<bool>used;

void dfs(int i) {
	used[i] = true;
	for (int j = 0; j < a[i].size(); j++) {
	if (!used[a[i][j]]) {
		dfs(a[i][j]);
	}
	}
}

int main() {
	int n, m;
	cin >> n >> m;
```

```
	a.resize(n);
	used.resize(n);
	for (int i = 0; i < m; i++) {
	int f, t;
	cin >> f >> t;
	f--;
	t--;
	a[f].push_back(t);
	}
	int start, finish;
	cin >> start >> finish;
	start--;
	finish--;
	dfs(start);
	if (used[finish])cout << "Yes";
	else cout << "No";
	return 0;
```

Задача A2:

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

using namespace std;

ifstream in("input.txt");
```

```
ofstream out("output.txt");

vector<vector<int>>a;
vector<bool>used;

void dfs(int i) {
	used[i] = true;
	for (int j = 0; j < a[i].size(); j++) {
	if (!used[a[i][j]]) {
		dfs(a[i][j]);
	}
	}
}

int main() {
	int n, m;
	cin >> n >> m;
	a.resize(n);
	used.resize(n);
	for (int i = 0; i < m; i++) {
	int f, t;
	cin >> f >> t;
	f--;
	t--;
	a[f].push_back(t);
	}
	int start, finish;
	cin >> start >> finish;
```

```
	start--;
	finish--;
	dfs(start);
	if (used[finish])cout << "Yes";
	else cout << "No";
	return 0;
```

Задача A Full:

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

using namespace std;

ifstream in("input.txt");
ofstream out("output.txt");

vector<vector<int>>a;
vector<bool>used;

void dfs(int i) {
	used[i] = true;
	for (int j = 0; j < a[i].size(); j++) {
	if (!used[a[i][j]]) {
		dfs(a[i][j]);
	}
```

```
	}
}

int main() {
	int n, m;
	cin >> n >> m;
	a.resize(n);
	used.resize(n);
	for (int i = 0; i < m; i++) {
	int f, t;
	cin >> f >> t;
	f--;
	t--;
	a[f].push_back(t);
	}
	int start, finish;
	cin >> start >> finish;
	start--;
	finish--;
	dfs(start);
	if (used[finish])cout << "Yes";
	else cout << "No";
	return 0;
}
```

Задача B1:

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

```
#include <map>
#include <algorithm>
#include <set>

using namespace std;

ifstream in("input.txt");
ofstream out("output.txt");

vector<vector<pair<int, int>>>a;
vector<bool>used;



int main() {
	int n, m;
	cin >> n >> m;
	a.resize(n);
	for (int i = 0; i < m; i++) {
	int f, t, c;
	cin >> f >> t >> c;
	f--;
	t--;
	a[f].push_back(make_pair(t,c));
	}
	int s, f;
	cin >> s >> f;
	s--;
	f--;
```

```
	int q, q0;
	cin >> q >> q0;
	for (int i = 0; i < q; i++) {
	int fr, to, c;
	cin >> fr >> to >> c;
	fr--;
	to--;
	a[fr].push_back(make_pair(to,c+q0));
	}
	vector<int>dist(n,99999999);
	dist[s] = 0;
	set<pair<int, int>>st;
	st.insert(make_pair(0,s));
	while (!st.empty()) {
	int i = st.begin()->second;
	int d = st.begin()->first;
	st.erase(make_pair(d, i));
	for (int j = 0; j < a[i].size(); j++) {
		if (d + a[i][j].second < dist[a[i][j].first]) {
			st.erase(make_pair(dist[a[i][j].first],
a[i][j].first));
			dist[a[i][j].first] = d + a[i][j].second;
			st.insert(make_pair(dist[a[i][j].first],
a[i][j].first));
		}
	}
	}
	if(dist[f] != 99999999)cout << dist[f];
	else cout << -1;
```

```
	return 0;
}
```

Задача B2:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
#include <set>

using namespace std;

ifstream in("input.txt");
ofstream out("output.txt");

vector<vector<pair<int, int>>>a;
vector<bool>used;


int main() {
	int n, m;
	cin >> n >> m;
	a.resize(n);
	for (int i = 0; i < m; i++) {
	int f, t, c;
	cin >> f >> t >> c;
	f--;
```

```
	t--;
	a[f].push_back(make_pair(t,c));
	}
	int s, f;
	cin >> s >> f;
	s--;
	f--;
	int q, q0;
	cin >> q >> q0;
	for (int i = 0; i < q; i++) {
	int fr, to, c;
	cin >> fr >> to >> c;
	fr--;
	to--;
	a[fr].push_back(make_pair(to,c+q0));
	}
	vector<int>dist(n,99999999);
	dist[s] = 0;
	set<pair<int, int>>st;
	st.insert(make_pair(0,s));
	while (!st.empty()) {
	int i = st.begin()->second;
	int d = st.begin()->first;
	st.erase(make_pair(d, i));
	for (int j = 0; j < a[i].size(); j++) {
		if (d + a[i][j].second < dist[a[i][j].first]) {
			st.erase(make_pair(dist[a[i][j].first],
a[i][j].first));
```

```
                dist[a[i][j].first] = d + a[i][j].second;
                st.insert(make_pair(dist[a[i][j].first],
a[i][j].first));
           }
      }
      }
      if(dist[f] != 99999999)cout << dist[f];
      else cout << -1;
      return 0;
}
```

Задача B Full:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
#include <set>

using namespace std;

ifstream in("input.txt");
ofstream out("output.txt");

vector<vector<pair<int, int>>>a;
vector<bool>used;
```

```
int main() {
	int n, m;
	cin >> n >> m;
	a.resize(n);
	for (int i = 0; i < m; i++) {
	int f, t, c;
	cin >> f >> t >> c;
	f--;
	t--;
	a[f].push_back(make_pair(t,c));
	}
	int s, f;
	cin >> s >> f;
	s--;
	f--;
	int q, q0;
	cin >> q >> q0;
	for (int i = 0; i < q; i++) {
	int fr, to, c;
	cin >> fr >> to >> c;
	fr--;
	to--;
	a[fr].push_back(make_pair(to,c+q0));
	}
	vector<int>dist(n,99999999);
	dist[s] = 0;
	set<pair<int, int>>st;
	st.insert(make_pair(0,s));
```

```
      while (!st.empty()) {
      int i = st.begin()->second;
      int d = st.begin()->first;
      st.erase(make_pair(d, i));
      for (int j = 0; j < a[i].size(); j++) {
            if (d + a[i][j].second < dist[a[i][j].first]) {
                  st.erase(make_pair(dist[a[i][j].first],
a[i][j].first));
                  dist[a[i][j].first] = d + a[i][j].second;
                  st.insert(make_pair(dist[a[i][j].first],
a[i][j].first));
            }
      }
      }
      if(dist[f] != 99999999)cout << dist[f];
      else cout << -1;
      return 0;
```

Задача C:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
#include <set>

using namespace std;

ifstream in("input.txt");
ofstream out("output.txt");
```

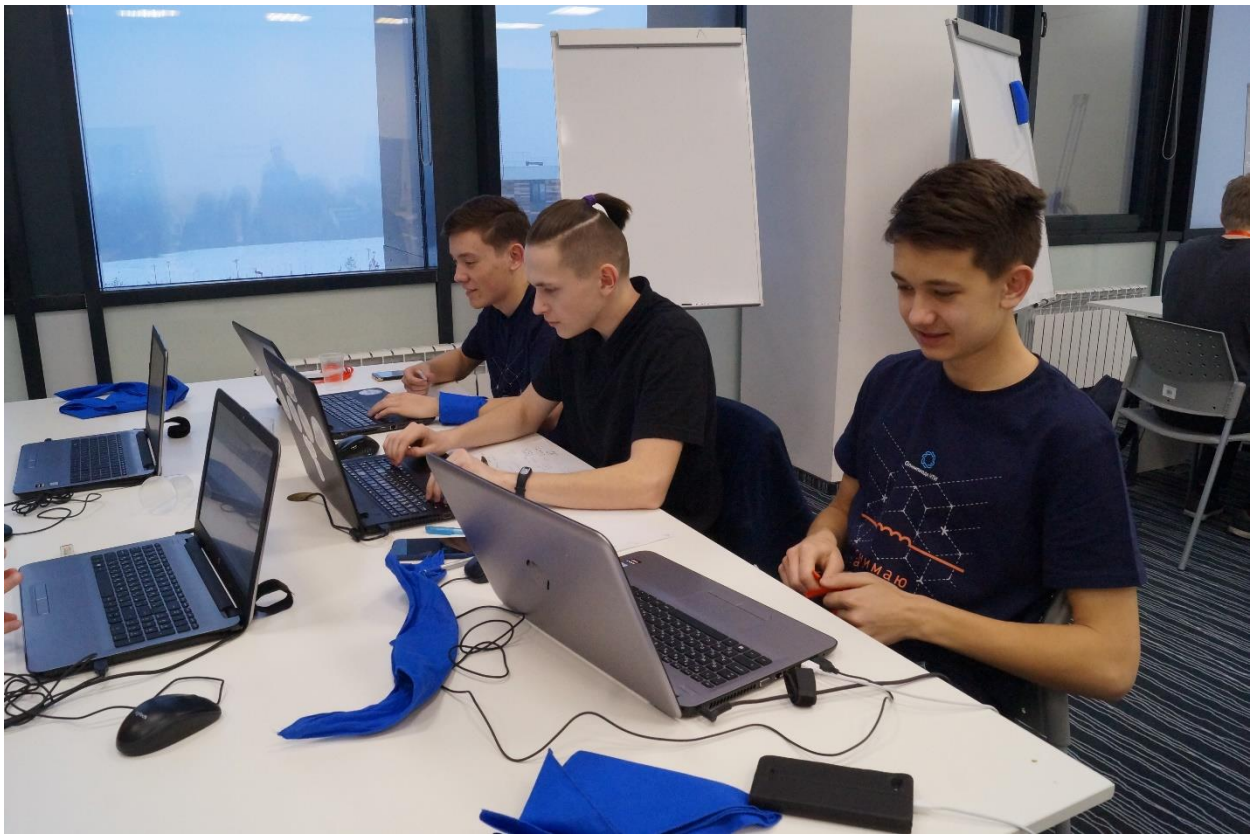
```
map<pair<int,long long>,long long>mp;
vector<vector<pair<int, long long>>>a;
vector<bool>used;

int beg = 0;

void dfs(int i, long long xo) {
	used[i] = true;
	for (int j = 0; j < a[i].size(); j++) {
	if (!used[a[i][j].first]) {
		mp[make_pair(beg, (xo ^ a[i][j].second))]++;
		dfs(a[i][j].first, (xo ^ a[i][j].second));
	}
	}
}

int main() {
	int n, q;
	cin >> n >> q;
	a.resize(n);
	used.resize(n);
	for (int i = 0; i < n - 1; i++) {
	int f, t;
	long long v;
	cin >> f >> t >> v;
	f--;
	t--;
```

```
        a[f].push_back(make_pair(t,v));
        //a[t].push_back(make_pair(f, v));
        }
        for (int i = 0; i < n; i++) {
        used.assign(n,false);
        beg = i;
        dfs(i,0);
        }
        for (int i = 0; i < q; i++) {
        int u;
        long long qi;
        cin >> u >> qi;
        u--;
        cout << mp[make_pair(u,qi)] << '\n';
        }
        return 0;
}
```

Командная часть

Результаты были получены в рамках выступления команды: Deus Vult

Фотография команды за работой:



Личный состав команды:

- Хилажев Линар Рафилевич;
- Шакиров Ильяс Саматович
- Байков Аяз Ильдарович

Протокол выполнения заданий:

Первый этап:

№ задания	Критерий	Возможные баллы	Полученные баллы
US-001	Регистрация аккаунта компании производителя ПО	2	2

US-002	Регистрация контрактов	12	0
US-005	Создание аккаунта производителя батарей	2	0
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	3
US-006	Получение информации о необходимом депозите за регистрацию производителя батарей	4	0
US-007	Регистрация нового производителя (все)	4	0
US-008	Защита от повторной регистрации нового производителя	4	0
US-009	Получение информации о сборе за регистрацию одной батареи	4	0
US-011	Регистрация батарей с закрепленным сбором за регистрацию батарей	4	0
US-012	Получение размера остатка по депозиту	4	0
US-013	Пакетная регистрация производимых батарей	6	0
US-015	Поштучная продажа новых батарей	6	0

US-017	Генерация прошивок аппаратных ключей батарей	4	0
US-018	Изменение количества заряда батареи	3	0
US-019	Получение информации о батарее из прошивки	8	0
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**	**3**

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	0
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	0
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**	**16**

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 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 argparse
import sys
import os

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

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

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

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

    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 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 verify(path):
    os.system('python3 {} --get'.format(path))
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

    )

    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:
```

```
        verify(args.verify)

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

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

Задача Setup:

```
import warnings
warnings.filterwarnings("ignore")
 #!/usr/bin/env python -W ignore::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 *
from web3.utils import *
from argparse import *
from solc import *
import sys
import json

web = Web3()

# configure provider to work with PoA chains
web.middleware_stack.inject(geth_poa_middleware, layer=0)

account_address = ''
account_pass = ''

try:
    with open('account.json', 'r') as file:
        d = json.load(file)
        account_address = d['account']#.lower()
        account_pass = d['password']
except FileNotFoundError:
    account_address = ''
    account_pass = ''
def deploy(path, contract):
    compiled = compile_files([path])
    contract_abi = compiled[path+':'+contract]['abi']
    contract_bin = compiled[path+':'+contract]['bin']
    contract = web.eth.contract(abi = contract_abi, bytecode=contract_bin)
    web.personal.unlockAccount(account_address, account_pass)
    fullhash = contract.deploy({'from': account_address})
    wait_for_transaction_receipt(web,fullhash)
    return web.eth.getTransactionReceipt(fullhash)['contractAddress']

def deploy_bat_man(path, contract, mgmt_addr, erc20_address):
    compiled = compile_files([path])
    contract_abi = compiled[path+':'+contract]['abi']
    contract_bin = compiled[path+':'+contract]['bin']
    contract = web.eth.contract(abi = contract_abi, bytecode=contract_bin)
    web.personal.unlockAccount(account_address, account_pass)
    fullhash = contract.deploy({'from': account_address},
[mgmt_addr,erc20_address])
    wait_for_transaction_receipt(web,fullhash)
    return web.eth.getTransactionReceipt(fullhash)['contractAddress']
```

```
def deploy_managementContract(path,contract,start_fee, wallet_address):
    compiled = compile_files([path])
    contract_abi = compiled[path+':'+contract]['abi']
    contract_bin = compiled[path+':'+contract]['bin']
    contract = web.eth.contract(abi = contract_abi, bytecode=contract_bin)
    web.personal.unlockAccount(account_address, account_pass)
    fullhash = contract.deploy({'from': account_address},
[wallet_address,web.toWei(start_fee,'ether')])
    wait_for_transaction_receipt(web,fullhash)
    return web.eth.getTransactionReceipt(fullhash)['contractAddress']

def new_account(parametr):
    account_p = parametr
    account_a = web.personal.newAccount(account_p)
    d = {'account': account_a, 'password': account_p}
    account_address = account_a
    account_pass = parametr
    print(account_a)
    with open('account.json', 'w') as file:
        json.dump(d,file)

def set_battery_address(bat_adr):
    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    man_con = web.eth.contract(address = man_addr['mgmtContract'], abi =
con_abi)
    web.personal.unlockAccount(account_address, account_pass)
    full_hash =
man_con.functions.setBatteryManagementContract(bat_adr).transact({'from':
account_address})
    wait_for_transaction_receipt(web, full_hash)

def get_battery_address():
    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    man_con = web.eth.contract(address = man_addr['mgmtContract'], abi =
con_abi)
    web.personal.unlockAccount(account_address, account_pass)
    return man_con.functions.batteryManagement().call({'from':
account_address})

def setup(start_fee):
    #Deploying Wallet Contract
    wallet_address = deploy('contracts/ServiceProviderWallet.sol',
'ServiceProviderWallet')
    #Deploying Managment Contract
    address =
deploy_managementContract('contracts/ManagementContract.sol',
'ManagementContract', start_fee, wallet_address)
    d = {'mgmtContract': address}
    with open('database.json', 'w') as file:
            json.dump(d,file)
```

```
    #Deploying Battery Contract
    erc20_address = deploy('contracts/ERC20Token.sol', 'ERC20Token')
    bat_address = deploy_bat_man("contracts/BatteryManagement.sol",
"BatteryManagement", address, erc20_address)

    #print(bat_address)
    #Deploying Currency Token
    set_battery_address(bat_address)
    print("Management contract:", address)
    print("Wallet contract:", wallet_address)
    print("Currency contract:", erc20_address)

def set_fee(new_fee):
    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    man_con = web.eth.contract(address = man_addr['mgmtContract'], abi =
con_abi)
    web.personal.unlockAccount(account_address, account_pass)
    try:
        fullhash = man_con.transact({'from':
account_address}).setFee(web.toWei(new_fee, 'ether'))
        wait_for_transaction_receipt(web,fullhash)
        print("Updated successfully")
    except ValueError:
        print("No permissions to change the service fee")



def main():
    parser = ArgumentParser()
    commands = ['--new', '--setup', '--setfee']
    for i in commands:
            parser.add_argument(i)
    namespace = parser.parse_args(sys.argv[1:])

    if namespace.new:
        new_account(namespace.new)

    elif namespace.setup:
        setup(namespace.setup)

    elif namespace.setfee:
        set_fee(namespace.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:

```
import warnings
warnings.simplefilter("ignore", category=DeprecationWarning)

# install web3 >=4.0.0 by `pip install --upgrade 'web3==4.0.0b11'`
from web3 import Web3
import os
from binascii import *
from web3.utils.transactions import *
from web3.middleware import geth_poa_middleware
from solc import *
from argparse import *
import sys
import json

web3 = Web3()

# configure provider to work with PoA chains
web3.middleware_stack.inject(geth_poa_middleware, layer=0)


account_address = ''
account_pass = ''

try:
	with open('account.json', 'r') as file:
		d = json.load(file)
		account_address = d['account']#.lower()
		account_pass = d['password']
except FileNotFoundError:
	account_address = ''
	account_pass = ''




def new_account(key):
    account_p = key
    account_a = web3.personal.newAccount(account_p)
    d = {'account': account_a, 'password': account_p}
    print(account_a)
    with open('account.json', 'w') as file:
        json.dump(d,file)

def get_one_bat_fee():
    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    man_con = web3.eth.contract(abi = con_abi)
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    man_con = man_con(man_addr['mgmtContract'])
    return man_con.call({'from':man_addr['mgmtContract']}).batteryFee()

def get_reg_fee():
    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    man_con = web3.eth.contract(abi = con_abi)
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    man_con = man_con(man_addr['mgmtContract'])
```

```
    return
man_con.call({'from':man_addr['mgmtContract']}).registrationDeposit()



def vendor_reg(name, money):
    if(web3.eth.getBalance(account_address) >= web3.toWei(money,
'ether')):
        compiled = compile_files(['contracts/ManagementContract.sol'])
        con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
        man_con = web3.eth.contract(abi = con_abi)
        with open('database.json', 'r') as file:
            man_addr = json.load(file)
        man_con = man_con(man_addr['mgmtContract'])
        web3.personal.unlockAccount(account_address, account_pass)
        fee = get_one_bat_fee();
        if(web3.toWei(money, 'ether') >= fee * 1000):
              nameExists = man_con.call({'from':
account_address}).nameExists(name.encode())
              if(not nameExists):
                     addressIsUsed =
man_con.call({'from':account_address}).addressIsUsed()
                     if(not addressIsUsed):
                            full_hash=
man_con.transact({'from':account_address, 'value':web3.toWei(money,
'ether')}).registerVendor(name.encode())
                            wait_for_transaction_receipt(web3, full_hash)
                            print('Success.')
                            id = man_con.call({'from':
account_address}).vendorId(account_address)
                            print("Vendor ID:", id.hex())
                     else:
                            print("Failed. The vendor address already
used.")
              else:
                     print("Failed. The vendor name is not unique.")
       else:
              print("Failed. Payment is low.")
    else:
        print("Failed. No enough funds to deposit.")

def get_vrs(private_key, our_text):
    pk = keys.PrivateKey(decode_hex(private_key))
    keccak = keccak_256()
    keccak.update(our_text.encode())
    _keys = KeyAPI(NativeECCBackend)
    signature = _keys.ecdsa_sign(keccak.digest(), pk)
    return signature.vrs

def create_battery(name, priv_key):
    try:
        f = open('firmware/{}.py'.format(name),'w')
    except FileNotFoundError:
        os.mkdir('firmware')
    f = open('firmware/{}.py'.format(name),'w')
    text = "from rlp.utils import decode_hex\nfrom sha3 import *\nfrom
eth_keys import *\nfrom eth_keys.backends import *\nimport json, argparse,
sys\nfrom datetime import datetime\nprivkey='{}'\ndef charge():\n\tdic = "
+ "{}" + "\n\ttry:\n\t\twith open('{}', 'r') as file:\n\t\t\tdic =
json.load(file)\n\t\t\tdic['n']+=1\n\t\twith open('{}', 'w') as
```

```
file:\n\t\t\tjson.dump(dic,file)\n\texcept FileNotFoundError:\n\t\twith
open('{}','w') as file:\n\t\t\tdic['n'] = 1\n\t\t\tjson.dump(dic,
file)\n\n"
    second_text = "\ndef
main():\n\tparser=argparse.ArgumentParser()\n\tparser.add_argument('--
charge',action='store_true')\n\tparser.add_argument('--get',
action='store_true')\n\tnamespace=parser.parse_args(sys.argv[1:])\n\tif(na
mespace.charge):\n\t\tcharge()\n\telif(namespace.get):\n\t\tget_info()\n\n
if __name__ == '__main__':\n\tmain()\n"
    four = "def get_charge():\n\ttry:\n\t\twith open('{}', 'r') as
file:\n\t\t\tdic = json.load(file)\n\t\t\treturn dic['n']\n\texcept
FileNotFoundError:\n\t\treturn 0"
    third = "\ndef
get_info():\n\tprint(get_charge())\n\tprint(get_time())\n\tget_vrc(str(get
_charge() * (2 ** 32) + get_time()))"
    get_time = "\ndef get_time():\n\treturn(int((datetime.now()-
datetime(1970, 1, 1)).total_seconds()))"
    get_vrc = "\ndef get_vrc(our_text):\n\tkeccak =
keccak_256()\n\tpk=keys.PrivateKey(decode_hex(privkey))\n\tkeccak.update(o
ur_text.encode())\n\t_keys = KeyAPI(NativeECCBackend)\n\tsignature =
_keys.ecdsa_sign(keccak.digest(), pk)\n\tv, r, s =
signature.vrs\n\tprint(v + 27)\n\tprint(hex(r)[2:])\n\tprint(hex(s)[2:])"

    f.write(text.format(priv_key, "{}","{}.json".format(name),
"{}.json".format(name), "{}.json".format(name)) +
four.format("{}.json".format(name)) + get_vrc + get_time + third +
second_text)
    f.close()


def reg_bat(count, value = 0):
    bat_keys = []
    args = []
    for i in range(count):
        priv_key = web3.sha3(os.urandom(20))

        bat_keys.append((priv_key,
web3.eth.account.privateKeyToAccount(priv_key).address))

        create_battery(bat_keys[i][1][2:10], priv_key.hex()[2:])

    for i in range(len(bat_keys)):
        args.append(web3.toBytes(hexstr=bat_keys[i][1]))

    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    man_con = web3.eth.contract(abi = con_abi)
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    man_con = man_con(man_addr['mgmtContract'])
    web3.personal.unlockAccount(account_address, account_pass)
    try:
        fullhash = man_con.transact({'from':account_address, 'value':
web3.toWei(value, 'ether')}).registerBatteries(args)
        receipt = wait_for_transaction_receipt(web3, fullhash)
        result = receipt.status
        for i in range(count):
            print("Created battery with ID:
{}".format(bat_keys[i][1][2:]))
    except ValueError:
```

```
        print("Failed. No enough funds to register object.")

def get_deposit():
        compiled = compile_files(['contracts/ManagementContract.sol'])
        con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
        man_con = web3.eth.contract(abi = con_abi)
        with open('database.json', 'r') as file:
                man_addr = json.load(file)
        web3.personal.unlockAccount(account_address, account_pass)
        man_con = man_con(man_addr['mgmtContract'])
        isUsed = man_con.call({'from':account_address}).addressIsUsed()
        if not isUsed :
                print("Vendor account is not registered.")
        else:
                result =
man_con.call({'from':account_address}).vendorDeposit(account_address)
                print("Deposit: {}".format(web3.fromWei(result, 'ether')))

def change_owner(bat_id, new_owner):
    web3.personal.unlockAccount(account_address, account_pass)
    compiled = compile_files(['contracts/BatteryManagement.sol'])
    con_abi =
compiled['contracts/BatteryManagement.sol:BatteryManagement']['abi']
    bat_con = web3.eth.contract(abi = con_abi)
    with open('database.json', 'r') as file:
        man_addr = json.load(file)
    compiled = compile_files(['contracts/ManagementContract.sol'])
    con_abi =
compiled['contracts/ManagementContract.sol:ManagementContract']['abi']
    man_con = web3.eth.contract(abi = con_abi)
    man_con = man_con(man_addr['mgmtContract'])
    bat_address =
man_con.call({'from':account_address}).batteryManagement()
    bat_con = bat_con(bat_address)
    cur_owner = bat_con.call({'from':account_address}).get_owner(bat_id)
    fullhash =
bat_con.transact({'from':account_address}).transfer(new_owner, bat_id)
    new_owner = bat_con.call({'from':account_address}).get_owner(bat_id)
    if(cur_owner != new_owner):
        print("Success")
    else:
        print("Failed. Not allowed to change ownership.")


def main():
    parser = ArgumentParser()
    commands = ['--new']
    for i in commands:
            parser.add_argument(i)
    parser.add_argument('--regfee', action = 'store_true')
    parser.add_argument('--deposit', action='store_true')
    parser.add_argument('--batfee', action = 'store_true')
    parser.add_argument('--reg', nargs='+', help='<Required> Set flag',
required=False)
    parser.add_argument('--bat', nargs='+', help='<Required> Set flag',
required=False)
    parser.add_argument('--owner', nargs='+', help='<Required> Set flag',
required=False)

    namespace = parser.parse_args(sys.argv[1:])
```

```
    if namespace.new:
        new_account(namespace.new)
    elif namespace.regfee:
        print("Vendor registration fee:
{}".format(web3.fromWei(get_reg_fee(), 'ether')))
    elif namespace.batfee:
        print("Production fee per one battery:
{}".format(web3.fromWei(get_one_bat_fee(), 'ether')))
    elif namespace.reg:
        vendor_reg(namespace.reg[0], (namespace.reg[1]))
    elif namespace.bat:
        if(len(namespace.bat) > 1):
            reg_bat(int(namespace.bat[0]), namespace.bat[1])
        else:
            reg_bat(int(namespace.bat[0]))
    elif namespace.deposit:
        get_deposit()
    elif namespace.owner:
        change_owner(namespace.owner[0], namespace.owner[1])



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

Программы для решения полной финальной задачи нет.