

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра САПР**

**ОТЧЕТ**  
**по лабораторной работе №3**  
**по дисциплине «Алгоритмы и структуры данных»**  
**"Алгоритмы на графах"**  
**Вариант 3**

Студент гр. 9302

\_\_\_\_\_

Кузнецов В.А.

Преподаватель

\_\_\_\_\_

Тутуева А.В.

Санкт-Петербург

2021

### **Цель работы.**

Реализовать нахождение кратчайшего пути по алгоритму Флойда-Уоршелла.

### **Постановка задачи.**

Дан список возможных авиарейсов в текстовом файле в формате:

Город отправления 1;Город прибытия 1;цена прямого перелета 1;цена обратного перелета 1

Город отправления 2;Город прибытия 2;цена перелета 2;цена обратного перелета 1

...

Город отправления N;Город прибытия N;цена перелета N;цена обратного перелета N

В случае, если нет прямого или обратного рейса, его цена будет указана как N/A (not available)

Пример данных:

Санкт-Петербург;Москва;10;20

Москва;Хабаровск;40;35

Санкт-Петербург;Хабаровск;14;N/A

Владивосток;Хабаровск;13;8

Владивосток;Санкт-Петербург;N/A;20

Задание: найти наиболее эффективный по стоимости перелет из города  $i$  в город  $j$ .

Вариант 3:

алгоритм Флойда-Уоршелла и матрицу смежности

### **Описание реализуемого алгоритма, класса и методов.**

Ключевая идея алгоритма — разбиение процесса поиска кратчайших путей на фазы.

Перед  $k$ -ой фазой ( $k = 0 \dots n-1$ ) считается, что в матрице расстояний сохранены длины таких кратчайших путей, которые содержат в качестве внутренних вершин только вершины из множества  $\{0, 1, \dots, k-1\}$  (вершины графа мы нумеруем, начиная с нуля). Вся работа, которую требуется произвести на  $k$ -ой фазе — это перебрать все пары вершин и пересчитать длину кратчайшего пути между ними. В результате после выполнения  $n$ -ой фазы в матрице расстояний будет записана длина кратчайшего пути между  $i$  и  $j$ , либо бесконечность, если пути между этими вершинами не существует.

В данной лабораторной работе вершинами графа будут считаться города, а ребрами между ними — возможность перелета. Весом ребра будет считаться стоимость перелета.

Был реализован класс `WTFAlgorithm` со следующими полями:

1. `string cities[100]` — статический массив уникальных городов, где индекс позиции города в этом массиве ассоциирует город с индексом в матрице смежности;
2. `int INF = 1000000` — формальное представление большого числа (бесконечности);
3. `int count = 0` — количество городов (вершин графа);
4. `int adjMatrix[100][100]` — статический массив, представляющий матрицу смежности;
5. `bool initd = false` — флаг, показывающий, был алгоритм инициализирован или нет;

Методы класса:

1. `int getCityIndex(string city)` — получить позицию (индекс) города `city` в массиве `cities`. Возвращает `-1`, если города нет;
2. `void addPrice(string first, string second, int price)` — добавление цены `price` перелета (веса ребра) в матрицу смежности от города `first` к городу `second`;

3. `void init()` – выполнение алгоритма. Минимальные суммы перелетов находятся в матрице смежности;
4. `int getMinimumPrice(string first, string second)` – возвращает минимальную стоимость перелета от города `first` в `second` или -1, если пути не существует;

### **Оценка временной сложности алгоритма.**

Так как алгоритм подразумевает обход в тройном цикле, временная сложность составляет  $O(n^3)$ .

### **Описание реализованных unit-тестов.**

Файлы для тестирования:

- test1.txt

New-York;London;15;25

New-York;Paris;30;50

New-York;Berlin;70;100

New-York;Moscow;25;30

New-York;Pekin;50;80

New-York;Tokyo;60;60

London;Paris;95;85

London;Berlin;60;5

London;Moscow;10;30

London;Pekin;25;20

London;Tokyo;30;55

Paris;Berlin;10;5

Paris;Moscow;60;15

Paris;Pekin;35;25

Paris;Tokyo;40;50

Berlin;Moscow;60;55

Berlin;Pekin;5;5

Berlin;Tokyo;45;40

Moscow;Pekin;90;80

Moscow;Tokyo;10;15

Pekin;Tokyo;65;65

- test2.txt

A;B;15;19

A;C;14;20

A;D;10;N/A

A;E;11;N/A

A;F;48;10

A;G;12;34

A;H;N/A;54

B;C;N/A;48

B;D;35;N/A

B;E;25;32

B;F;20;9

B;G;71;33

B;H;25;38

C;D;10;9

C;E;N/A;12

C;F;43;28

C;G;50;45

C;H;60;39

D;E;58;36

D;F;21;29

D;G;15;12

D;H;N/A;49

E;F;61;N/A

E;G;85;67

E;H;91;N/A

F;G;35;27

F;H;N/A;28

- test3.txt

A;B;N/A;25

A;D;13;N/A

A;E;19;10

A;F;N/A;8

B;C;12;8

B;F;20;N/A

C;D;N/A;8

C;E;11;N/A

C;F;10;20

D;F;N/A;6

E;F;3;5

Для тестирования алгоритма при каждом входных данных были реализованы методы Testing1, Testing2 и Testing3 соответственно.







Тестирование	Длительн...	Признаки	Сообщ
▲  UnitTest1 (3)	< 1 мс		
▲  UnitTest1 (3)	< 1 мс		
▲  UnitTest1 (3)	< 1 мс		
 Testing1	< 1 мс		
 Testing2	< 1 мс		
 Testing3	< 1 мс		

Рисунок 1 – Результаты тестирования

### Пример работы программы.

В файле prices.txt хранится информация рейсах:

New-York;London;15;25

New-York;Paris;N/A;50

New-York;Berlin;70;N/A

New-York;Moscow;25;30  
New-York;Pekin;50;N/A  
New-York;Tokyo;60;60  
London;Paris;95;85  
London;Berlin;60;5  
London;Moscow;10;N/A  
London;Pekin;25;20  
London;Tokyo;30;N/A  
Paris;Berlin;10;5  
Paris;Moscow;60;15  
Paris;Pekin;35;25  
Paris;Tokyo;40;N/A  
Berlin;Moscow;N/A;55  
Berlin;Pekin;5;5  
Berlin;Tokyo;45;40  
Moscow;Pekin;N/A;80  
Moscow;Tokyo;10;15  
Pekin;Tokyo;65;N/A

Узнаем, дешевле ли прямой полет из Лондона в Париж, а из Парижа в Москву перелета с пересадками и можно ли из Москвы долететь до Пекина.

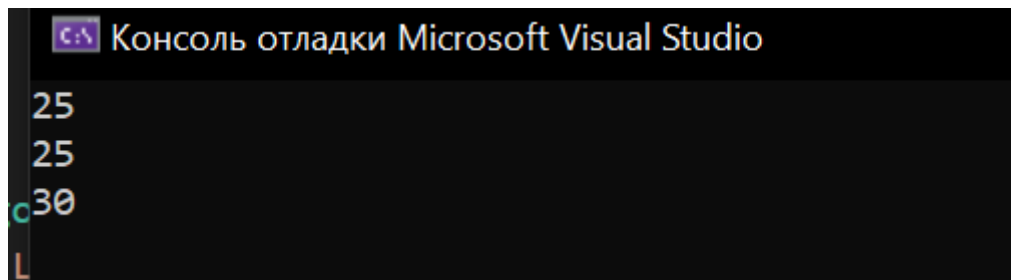
Код:

```
#include <iostream>
#include "WFIAgorithm.h"

using namespace std;

int main() {
    WFIAgorithm* prices = new WFIAgorithm("./prices.txt");
    cout << prices->getMinimumPrice("London", "Paris") << endl;
    cout << prices->getMinimumPrice("Paris", "Moscow") << endl;
    cout << prices->getMinimumPrice("Moscow", "Pekin") << endl;
    return 0;
}
```

Результат:



```
Консоль отладки Microsoft Visual Studio
25
25
30
```

Как видим, во всех трех случаях ответ положительный.



## ЛИСТИНГ

### WFIAAlgorithm.h

```
#pragma once
#include <string>

using namespace std;

class WFIAAlgorithm
{
private:
    string cities[100];
    int INF = 10000000;
    int count = 0;
    int adjMatrix[100][100];
    bool inited = false;
    int getCityIndex(string city);
    void addPrice(string first, string second, int price);
    void init();
public:
    WFIAAlgorithm(string path);
    int getMinimumPrice(string first, string second);
};
```

### WFIAAlgorithm.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include "WFIAAlgorithm.h"

using namespace std;

int WFIAAlgorithm::getCityIndex(string city) {
    for (int i = 0; i < count; i++) {
        if (cities[i] == city) {
            return i;
        }
    }
    return -1;
}

void WFIAAlgorithm::init() {
    for (int k = 0; k < count; ++k) {
        for (int i = 0; i < count; ++i) {
            for (int j = 0; j < count; ++j) {
                adjMatrix[i][j] = min(adjMatrix[i][j], adjMatrix[i][k] +
adjMatrix[k][j]);
            }
        }
    }
    inited = true;
}

WFIAAlgorithm::WFIAAlgorithm(string path) {
    for (int i = 0; i < 100; i++) {
        for (int j = 0; j < 100; j++) {
            adjMatrix[i][j] = i == j ? 0 : INF;
        }
    }
    ifstream file(path);
    if (file.is_open()) {
        std::string firstCity, secondCity, firstPrice, secondPrice;
```

```

        while (getline(file, firstCity, ';'))
        {
            getline(file, secondCity, ';');
            getline(file, firstPrice, ';');
            getline(file, secondPrice, '\n');
            if (firstPrice != "N/A") {
                addPrice(firstCity, secondCity, stoi(firstPrice));
            }
            if (secondPrice != "N/A") {
                addPrice(secondCity, firstCity, stoi(secondPrice));
            }
        }
    }
    file.close();
}

void WFIAAlgorithm::addPrice(string first, string second, int price) {
    int firstPos = getCityIndex(first);
    int secondPos = getCityIndex(second);
    if (firstPos == -1) {
        firstPos = count;
        cities[firstPos] = first;
        count += 1;
    }
    if (secondPos == -1) {
        secondPos = count;
        cities[secondPos] = second;
        count += 1;
    }
    adjMatrix[firstPos][secondPos] = price;
}

int WFIAAlgorithm::getMinimumPrice(string first, string second) {
    if (!init()) {
        init();
    }
    int firstPos = getCityIndex(first);
    int secondPos = getCityIndex(second);
    if (firstPos == -1 || secondPos == -1 || adjMatrix[firstPos][secondPos] == INF) {
        return -1;
    }
    return adjMatrix[firstPos][secondPos];
}

```

## UnitTest1.cpp

```

#include "pch.h"
#include "CppUnitTest.h"
#include "../Lab3_Kuznetsov/WFIAAlgorithm.h"
#include "../Lab3_Kuznetsov/WFIAAlgorithm.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
    TEST_CLASS(UnitTest1)
    {
    public:
        WFIAAlgorithm* test1 = new WFIAAlgorithm("../tests/test1.txt");
        WFIAAlgorithm* test2 = new WFIAAlgorithm("../tests/test2.txt");
        WFIAAlgorithm* test3 = new WFIAAlgorithm("../tests/test3.txt");
        TEST_METHOD(Testing1)
    }
}

```

```

    {
        Assert::AreEqual(40, test1->getMinimumPrice("New-York", "Berlin"));
        Assert::AreEqual(30, test1->getMinimumPrice("Berlin", "New-York"));
        Assert::AreEqual(40, test1->getMinimumPrice("New-York", "Berlin"));
        Assert::AreEqual(35, test1->getMinimumPrice("New-York", "Tokyo"));
        Assert::AreEqual(25, test1->getMinimumPrice("London", "Paris"));
        Assert::AreEqual(25, test1->getMinimumPrice("Paris", "Moscow"));
        Assert::AreEqual(15, test1->getMinimumPrice("Berlin", "Moscow"));
        Assert::AreEqual(30, test1->getMinimumPrice("Moscow", "Pekin"));
    }
    TEST_METHOD(Testing2)
    {
        Assert::AreEqual(15, test2->getMinimumPrice("A", "B"));
        Assert::AreEqual(10, test2->getMinimumPrice("A", "D"));
        Assert::AreEqual(29, test2->getMinimumPrice("D", "A"));
        Assert::AreEqual(31, test2->getMinimumPrice("A", "F"));
        Assert::AreEqual(40, test2->getMinimumPrice("A", "H"));
        Assert::AreEqual(29, test2->getMinimumPrice("B", "D"));
        Assert::AreEqual(25, test2->getMinimumPrice("B", "E"));
        Assert::AreEqual(35, test2->getMinimumPrice("C", "B"));
        Assert::AreEqual(10, test2->getMinimumPrice("C", "D"));
        Assert::AreEqual(60, test2->getMinimumPrice("C", "H"));
        Assert::AreEqual(55, test2->getMinimumPrice("D", "H"));
        Assert::AreEqual(32, test2->getMinimumPrice("E", "B"));
    }
    TEST_METHOD(Testing3)
    {
        Assert::AreEqual(29, test3->getMinimumPrice("A", "B"));
        Assert::AreEqual(26, test3->getMinimumPrice("D", "A"));
        Assert::AreEqual(22, test3->getMinimumPrice("A", "F"));
        Assert::AreEqual(26, test3->getMinimumPrice("B", "D"));
        Assert::AreEqual(19, test3->getMinimumPrice("D", "E"));
        Assert::AreEqual(3, test3->getMinimumPrice("E", "F"));
    }
};
}

```