

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
"Алгоритмы на графах"
Вариант 1

Студент гр. 9302

Точилин А.Е.

Преподаватель

Тутуева А.В.

Санкт-Петербург

2021

Цель работы.

Реализовать алгоритм кратчайшего пути в графе.

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 1:

алгоритм Дейкстры и списки смежности

Описание реализуемого алгоритма и структур данных.

Алгоритм Дейкстры представляет собой n итераций (по количеству вершин – городов), на каждой из которых выбирается непомеченная вершина (город) с наименьшей величиной цены перелета, эта вершина помечается, и затем просматриваются все рёбра (существующие перелеты в другие города),

исходящие из данной вершины, и вдоль каждого ребра делается попытка улучшить значение цены на другом конце ребра (на конечных городах).

Для этого был реализован класс `DijkstraAlgorithm`. Во время инициализации объекта класса происходит чтение файла по пути, переданному в конструкторе, далее файл парсится на отдельные города и цены и данные записываются. Далее инициализируются векторы цен перелетов и меток. Затем выполняется итерация по количеству уникальных городов (вершин). На каждой итерации сначала находится вершина, имеющая наименьшую цену среди непомеченных вершин. Если цена перелета до выбранной вершины оказывается равной бесконечности, то алгоритм останавливается. Иначе вершина помечается как помеченная, и просматриваются все рёбра, исходящие из данной вершины (перелеты в другие города), и вдоль каждого ребра выполняются релаксации. Если релаксация успешна (т.е. цена перелета уменьшается), то эта цена и устанавливается в вектор цен.

После выполнения всех итераций в векторе цен оказываются минимальные цены перелетов от одного города до всех остальных городов.

Класс `EndCity` представляет собой описание перелета от некоторого города до конкретного города. Поля класса:

1. `String city` – название конечного города;
2. `Int price` – цена перелета.

Класс `StartCity` представляет собой отдельный город со всеми существующими перелетами от него до других городов. Поля класса:

1. `String city` – название города;
2. `Vector<EndCity*>` вектор перелетов до конечных городов.

Поля класса `DijkstraAlgorithm`:

1. `vector<StartCity*> cities` – вектор городов. Представляет собой список смежности;
2. `vector<int> prices` – вектор цен перелетов от конкретного города до всех остальных;

3. `vector<bool> marks` – вектор меток посещенных городов;

Методы класса:

1. `void addEdge(string city1, string city2, int price)` – добавление ребра (перелета) из одного города `city1` в `city2` по цене `price` в список смежности;
2. `int getCityPos(string city)` – возвращает позицию города `city` в списке смежности и -1 если его нет;
3. `void clear()` – удаляет информацию в `prices` и `marks`, заполняя элементы в них `INT_MAX` и `false` соответственно;
4. `int getMinPrice(string city1, string city2)` – выполняет алгоритм и возвращает минимальную цену перелета из `city1` в `city2` или -1, если таковой нет.

Оценка временной сложности алгоритма.

Оценка работы алгоритма Дейкстры составляет $O(n^2)$.

Описание реализованных unit-тестов.

Для организации тестов был реализован класс `UnitTest1`, метод `TestAlgorithm` и следующие тестовые данные:

- Файл `test1.txt`

`Perm;St. Petersburg;10;N/A`

`Perm;Moscow;5;N/A`

`Perm;Novosibirsck;9;N/A`

`St. Petersburg;Novosibirsck;N/A;1`

`Moscow;Novosibirsck;3;N/A`

- Файл `test2.txt`

`Perm;St. Petersburg;10;5`

`Perm;Moscow;6;N/A`

`Perm;Novosibirsck;15;N/A`

`St. Petersburg;Novosibirsck;1;N/A`

Moscow;St. Petersburg;2;N/A

- Файл test3.txt

Perm;Moscow;6;10

Perm;Irkutsck;N/A;15

Moscow;St. Petersburg;9;N/A

Moscow;Novosibirsck;10;5

Moscow;Irkutsck;4;N/A

St. Petersburg;Novosibirsck;1;6

St. Petersburg;Irkutsck;7;3

- Test4.txt

Perm;St. Petersburg;10;4

Perm;Moscow;3;N/A

Perm;Novosibirsck;5;7

Perm;Irkutsck;1;N/A

Perm;Rostov;N/A;9

Moscow;St. Petersburg;1;10

Moscow;Rostov;9;N/A

Moscow;Irkutsck;N/A;15

Moscow;Novosibirsck;N/A;5

St. Petersburg;Irkutsck;N/A;13

Rostov;Irkutsck;N/A;15

Rostov;Novosibirsck;15;N/A

Irkutsck;Novosibirsck;8;6





Тестирование	Длительн...	Признаки	Сообщение об ошибке
▲  UnitTest1 (1)	2 мс		
▲  UnitTest1 (1)	2 мс		
▲  UnitTest1 (1)	2 мс		
 TestAlgorithm	2 мс		

Рисунок 1 – Результаты тестирования

Пример работы программы.

В файле aviasales.txt находятся следующие записи о стоимости перелетов:

Moscow;St. Petersburg;2000;2400
Moscow;Kazan;5100;5800
Moscow;Novosibirsk;7400;7800
Moscow;Irkutsk;13700;14000
Moscow;Chita;N/A;15500
Moscow;Vladivostok;16700;N/A
St. Petersburg;Kazan;N/A;5200
St. Petersburg;Novosibirsk;6000;N/A
St. Petersburg;Vladivostok;16900;17600
Kazan;Novosibirsk;1300;1800
Kazan;Irkutsk;4690;4100
Kazan;Chita;5440;5000
Novosibirsk;Irkutsk;3100;4000
Novosibirsk;Chita;3800;4200
Novosibirsk;Vladivostok;4600;5000
Irkutsk;Chita;600;900
Irkutsk;Vladivostok;1000;N/A
Chita;Vladivostok;400;900

Выясним самую дешевую стоимость полетов от Москвы до Иркутска, Читы и Владивостока, и обратно.

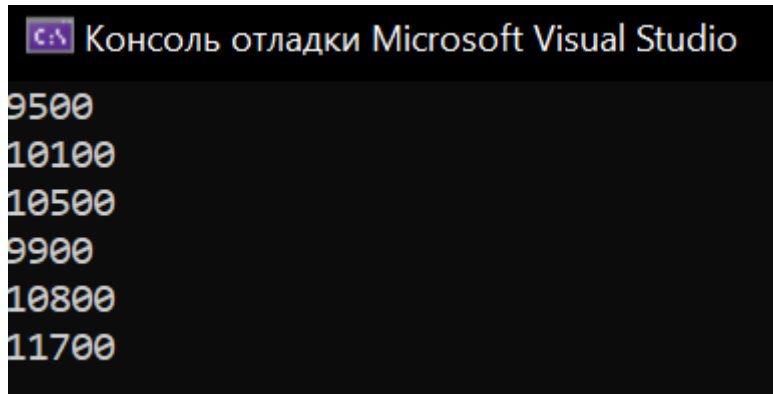
Код приложения:

```
#include <iostream>
#include "Lr3.h"

using namespace std;

int main() {
    DijkstraAlgorithm* aviasales = new DijkstraAlgorithm("./aviasales.txt");
    cout << aviasales->getMinPrice("Moscow", "Irkutsk") << endl;
    cout << aviasales->getMinPrice("Moscow", "Chita") << endl;
    cout << aviasales->getMinPrice("Moscow", "Vladivostok") << endl;
    cout << aviasales->getMinPrice("Irkutsk", "Moscow") << endl;
    cout << aviasales->getMinPrice("Chita", "Moscow") << endl;
    cout << aviasales->getMinPrice("Vladivostok", "Moscow") << endl;
    return 0;
}
```

Демонстрация работы:

A screenshot of the Microsoft Visual Studio debug console. The title bar at the top reads "Консоль отладки Microsoft Visual Studio". The console output shows a list of flight prices in Russian rubles (₽) for various destinations. The prices are: 9500, 10100, 10500, 9900, 10800, and 11700. The text is displayed in a light blue font on a dark background.

```
9500
10100
10500
9900
10800
11700
```

Можем видеть, что цена с пересадками в других городах существенно ниже, чем прямые полеты.

ЛИСТИНГ

Lr3.h

```
#pragma once
#include <string>
#include <vector>

using namespace std;

class EndCity {
public:
    string city;
    int price;
    EndCity(string, int);
};

class StartCity {
public:
    string city;
    vector<EndCity*> endCities;
    StartCity(string);
};

class DijkstraAlgorithm {
    vector<StartCity*> cities;
    vector<int> prices;
    vector<bool> marks;
    void addEdge(string, string, int);
    int getCityPos(string);
    void clear();
public:
    DijkstraAlgorithm(string);
    int getMinPrice(string, string);
};
```

Lr3.cpp

```
#include <iostream>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "Lr3.h"

using namespace std;

StartCity::StartCity(string city) {
    this->city = city;
}

EndCity::EndCity(string city, int price) {
    this->city = city;
    this->price = price;
}

DijkstraAlgorithm::DijkstraAlgorithm(string path) {
    ifstream file(path);
    if (file.is_open()) {
        std::string city1, city2, price1, price2;

        while (getline(file, city1, ';'))
        {
            getline(file, city2, ';');
            getline(file, price1, ';');
        }
    }
}
```



```

        getline(file, price2, '\n');
        if (price1 != "N/A") {
            addEdge(city1, city2, stoi(price1));
        }
        if (price2 != "N/A") {
            addEdge(city2, city1, stoi(price2));
        }
    }
    file.close();
    for (size_t i = 0; i < cities.size(); i++) {
        prices.push_back(INT_MAX);
        marks.push_back(false);
    }
}

void DijkstraAlgorithm::addEdge(string city1, string city2, int price) {
    bool isExists = false;
    int city1Pos = getCityPos(city1);
    int city2Pos = getCityPos(city2);
    if (city1Pos == -1) {
        StartCity* start = new StartCity(city1);
        cities.push_back(start);
        city1Pos = cities.size() - 1;
    }
    if (city2Pos == -1) {
        StartCity* start = new StartCity(city2);
        cities.push_back(start);
    }
    EndCity* end = new EndCity(city2, price);
    cities[city1Pos]->endCities.push_back(end);
}

int DijkstraAlgorithm::getCityPos(string city) {
    if (cities.size() == 0) {
        return -1;
    }
    size_t cityPos = 0;
    while (cities[cityPos]->city != city) {
        cityPos += 1;
        if (cityPos == cities.size()) {
            return -1;
        }
    }
    return cityPos;
}

void DijkstraAlgorithm::clear() {
    for (size_t i = 0; i < cities.size(); i++) {
        prices[i] = INT_MAX;
        marks[i] = false;
    }
}

int DijkstraAlgorithm::getMinPrice(string city1, string city2) {
    int city1Pos = getCityPos(city1);
    int city2Pos = getCityPos(city2);
    if (city1Pos == -1 || city2Pos == -1) {
        return -1;
    }
    if (prices[city1Pos] == 0) {
        return prices[city2Pos];
    }
    clear();
    prices[city1Pos] = 0;
}

```

```

for (size_t i = 0; i < cities.size(); i++) {
    int curr = -1;
    for (size_t j = 0; j < cities.size(); j++) {
        if (!marks[j] && (curr == -1 || prices[j] < prices[curr])) {
            curr = j;
        }
    }
    if (curr == -1) {
        continue;
    }
    if (prices[curr] == INT_MAX) {
        break;
    }
    marks[curr] = true;
    for (size_t j = 0; j < cities[curr]->endCities.size(); j++) {
        size_t endCityPos = 0;
        while (cities[endCityPos]->city != cities[curr]->endCities[j]->city)
        {
            endCityPos += 1;
        }
        if (prices[curr] + cities[curr]->endCities[j]->price <
prices[endCityPos]) {
            prices[endCityPos] = prices[curr] + cities[curr]-
>endCities[j]->price;
        }
    }
    if (prices[city2Pos] == INT_MAX) {
        return -1;
    }
    return prices[city2Pos];
}

```

UnitTest1.cpp

```

#include "pch.h"
#include "CppUnitTest.h"
#include "../Tochilin_lr3/Lr3.h"
#include "../Tochilin_lr3/Lr3.cpp"

```

```
using namespace Microsoft::VisualStudio::CppUnitTestFramework;
```

```
namespace UnitTest1
```

```

{
    TEST_CLASS(UnitTest1)
    {
    public:
        DijkstraAlgorithm* test1 = new
DijkstraAlgorithm("../UnitTest1/tests/test1.txt");
        DijkstraAlgorithm* test2 = new
DijkstraAlgorithm("../UnitTest1/tests/test2.txt");
        DijkstraAlgorithm* test3 = new
DijkstraAlgorithm("../UnitTest1/tests/test3.txt");
        DijkstraAlgorithm* test4 = new
DijkstraAlgorithm("../UnitTest1/tests/test4.txt");
        TEST_METHOD(TestAlgorithm)
        {
            Assert::AreEqual(9, test1->getMinPrice("Perm", "St. Petersburg"));
            Assert::AreEqual(8, test1->getMinPrice("Perm", "Novosibirsk"));
            Assert::AreEqual(-1, test1->getMinPrice("Novosibirsk", "Perm"));
            Assert::AreEqual(9, test2->getMinPrice("Perm", "Novosibirsk"));
            Assert::AreEqual(2, test2->getMinPrice("Moscow", "St. Petersburg"));
            Assert::AreEqual(-1, test2->getMinPrice("Novosibirsk", "St.
Petersburg"));
            Assert::AreEqual(13, test3->getMinPrice("Perm", "St. Petersburg"));

```

```

        Assert::AreEqual(10, test3->getMinPrice("Perm", "Irkutsck"));
        Assert::AreEqual(14, test3->getMinPrice("Perm", "Novosibirsck"));
        Assert::AreEqual(5, test3->getMinPrice("Novosibirsck", "Moscow"));
        Assert::AreEqual(4, test3->getMinPrice("Moscow", "Irkutsck"));
        Assert::AreEqual(7, test3->getMinPrice("St. Petersburg",
"Irkutsck"));

        Assert::AreEqual(5, test4->getMinPrice("Perm", "Novosibirsck"));
        Assert::AreEqual(10, test4->getMinPrice("Moscow", "Novosibirsck"));
        Assert::AreEqual(8, test4->getMinPrice("Irkutsck", "Novosibirsck"));
        Assert::AreEqual(9, test4->getMinPrice("St. Petersburg",
"Novosibirsck"));

        Assert::AreEqual(14, test4->getMinPrice("Rostov", "Novosibirsck"));
        Assert::AreEqual(10, test4->getMinPrice("Rostov", "Irkutsck"));
        Assert::AreEqual(12, test4->getMinPrice("Rostov", "Moscow"));
        Assert::AreEqual(13, test4->getMinPrice("Rostov", "St. Petersburg"));
        Assert::AreEqual(9, test4->getMinPrice("Rostov", "Perm"));
        Assert::AreEqual(8, test4->getMinPrice("Irkutsck", "Novosibirsck"));
        Assert::AreEqual(15, test4->getMinPrice("Irkutsck", "Rostov"));
        Assert::AreEqual(15, test4->getMinPrice("Irkutsck", "Perm"));
    }

};

}

```