**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»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9301 |  | Русанова К.В. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2021

### 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

Найти наиболее эффективный по стоимости перелет из города i в город j. При этом использовать список смежности и алгоритм Дейкстры.

**Описание реализуемых алгоритмов и используемых структур**

Для хранения списка смежности и весов используется массив списков. Для сохранения соответствия между городом и его номером используется массив структур, полями которых являются название города и длина названия. Для оптимального перемещения по списку используется итератор.

### Оценка временной сложности

Табл. 1

|  |  |
| --- | --- |
| **Название метода** | **Сложность** |
| readList() | O(n) |
| Dijkstra() | O(n2) |
| writeList() | O(n) |

### Реализованные unit-тесты

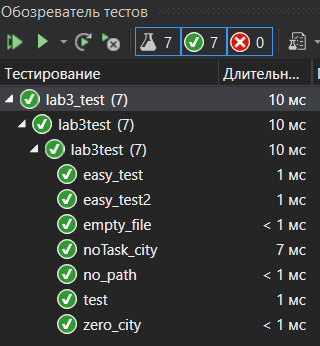


Рис. 1 — Пройденные тесты

* empty\_file — Проверка ошибки при пустом файле
* zero\_city — Проверка ошибки при нулевом графе
* noTask\_city — Проверка ошибки при отсутствии условия
* easy\_test — Построение прямого пути
* easy\_test2 — Проверка нахождения кротчайшего пути
* no\_path — Проверка изолированной вершины
* test — Проверка графа побольше

### Пример работы

На рисунке 2 показан пример входного и итогового файлов.

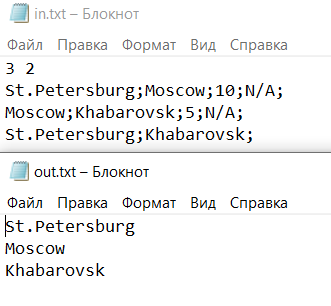


Рис. 2 — Демонстрация работы

Лист. 1 — lab3.cpp

#include <iostream>

#include <fstream>

#include "dualList.h"

#include "task3.h"

using namespace std;

int main()

{

string fileNameIn = "in.txt";

char fileNameOut[8] = "out.txt";

task3 task;

int size;

task.readList(fileNameIn);

task.Dijkstra(size);

task.writeList(fileNameOut);

}

### Листинг

Лист. 1 — task3.h

#pragma once

#include "dualList.h"

class task3

{

private:

int cityCount;

dualList\* adjacencyList;

dualList\* price;

int\* result;

int resultSize;

int from, to;

struct City

{

char\* name;

size\_t nameSize;

};

City\* numberName;

public:

void readList(char\*);

void readList(string);

int\* Dijkstra(int&);

void writeList(char\*);

void writeList(string);

};

Лист. 3 — task3.cpp

#include "task3.h"

#include "Iterator.h"

#include <fstream>

using namespace std;

int\* task3::Dijkstra(int& \_resultSize) {

int\* distance = new int[cityCount];

int\* fromWhere = new int[cityCount];

bool\* visited = new bool[cityCount];

for (size\_t i = 0; i < cityCount; i++) {

distance[i] = INT\_MAX;

visited[i] = false;

}

Iterator\* priceIter = price[from].create\_iterator(0);

Iterator\* adjacIter = adjacencyList[from].create\_iterator(0);

int bufCity,bufPrice;

while (priceIter->has\_next())

{

bufCity = adjacIter->next();

bufPrice = priceIter->next();

distance[bufCity] = bufPrice;

fromWhere[bufCity] = from;

}

distance[from] = 0;

visited[from] = true;

fromWhere[from] = from;

int min,index=0;

for (size\_t i = 0; i < cityCount; i++)

{

min = INT\_MAX;

for (size\_t j = 0; j < cityCount; j++)

{

if ((visited[j]==false)&&(distance[j]<min))

{

min = distance[j];

index = j;

}

}

visited[index] = true;

priceIter = price[index].create\_iterator(0);

adjacIter = adjacencyList[index].create\_iterator(0);

while (priceIter->has\_next())

{

bufCity = adjacIter->next();

bufPrice = priceIter->next();

if ((visited[bufCity]==false)&&(distance[index]!=INT\_MAX)&&((distance[index]+bufPrice)<distance[bufCity]))

{

distance[bufCity] = distance[index] + bufPrice;

fromWhere[bufCity] = index;

}

}

}

if (distance[to] == INT\_MAX)

throw out\_of\_range("Result not found");

else {

int count = 2;

index = to;

while (fromWhere[index]!=from)

{

count++;

index = fromWhere[index];

}

\_resultSize = count;

result = new int[\_resultSize];

result[\_resultSize - 1] = to;

result[0] = from;

int i = \_resultSize -2;

index = to;

while (fromWhere[index] != from)

{

result[i]= fromWhere[index];

i--;

index = fromWhere[index];

}

resultSize = \_resultSize;

return result;

}

}

void task3::readList(char\* fileName) {

fstream file;

file.open(fileName, ios::in);

int countStr;

file >> cityCount >> countStr;

if (file.eof() || cityCount == 0) throw out\_of\_range("Graph is empty");

adjacencyList = new dualList[cityCount];

price = new dualList[cityCount];

numberName = new City[cityCount];

char\* str;

int indexStr = 0;

int indexCity = 0;

int found1 = -1;

int found2 = -1;

int bufPrice;

for (size\_t i = 0; (i < countStr) && (!file.eof()); i++) {

indexStr = 0;

str = new char[100];

file >> str[indexStr];

while (str[indexStr] != ';') //first city

{

indexStr++;

file >> str[indexStr];

}

found1 = -1;

for (size\_t i = 0; (i < indexCity) && (found1 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found1 = i;

else {

found1 = -1;

break;

}

}

}

if (found1 == -1)

{

numberName[indexCity].nameSize = indexStr;

numberName[indexCity].name = new char[indexStr];

for (size\_t i = 0; i < indexStr; i++)

numberName[indexCity].name[i] = str[i];

found1 = indexCity;

indexCity++;

}

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //second city

{

indexStr++;

file >> str[indexStr];

}

found2 = -1;

for (size\_t i = 0; (i < indexCity) && (found2 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found2 = i;

else {

found2 = -1;

break;

}

}

}

if (found2 == -1)

{

numberName[indexCity].nameSize = indexStr;

numberName[indexCity].name = new char[indexStr];

for (size\_t i = 0; i < indexStr; i++)

numberName[indexCity].name[i] = str[i];

found2 = indexCity;

indexCity++;

}

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //first price

{

indexStr++;

file >> str[indexStr];

}

if (str[0] == 'N')

{

}

else {

bufPrice = 0;

for (size\_t i = 0; i < indexStr; i++)

{

switch (str[i]) //convert to int

{

case '0':bufPrice \*= 10; break;

case '1':bufPrice = bufPrice \* 10 + 1; break;

case '2':bufPrice = bufPrice \* 10 + 2; break;

case '3':bufPrice = bufPrice \* 10 + 3; break;

case '4':bufPrice = bufPrice \* 10 + 4; break;

case '5':bufPrice = bufPrice \* 10 + 5; break;

case '6':bufPrice = bufPrice \* 10 + 6; break;

case '7':bufPrice = bufPrice \* 10 + 7; break;

case '8':bufPrice = bufPrice \* 10 + 8; break;

case '9':bufPrice = bufPrice \* 10 + 9; break;

default:

break;

}

}

adjacencyList[found1].push\_back(found2);

price[found1].push\_back(bufPrice);

}

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //second price

{

indexStr++;

file >> str[indexStr];

}

if (str[0] == 'N')

{

}

else {

bufPrice = 0;

for (size\_t i = 0; i < indexStr; i++)

{

switch (str[i]) //convert to int

{

case '0':bufPrice \*= 10; break;

case '1':bufPrice = bufPrice \* 10 + 1; break;

case '2':bufPrice = bufPrice \* 10 + 2; break;

case '3':bufPrice = bufPrice \* 10 + 3; break;

case '4':bufPrice = bufPrice \* 10 + 4; break;

case '5':bufPrice = bufPrice \* 10 + 5; break;

case '6':bufPrice = bufPrice \* 10 + 6; break;

case '7':bufPrice = bufPrice \* 10 + 7; break;

case '8':bufPrice = bufPrice \* 10 + 8; break;

case '9':bufPrice = bufPrice \* 10 + 9; break;

default:

break;

}

}

adjacencyList[found2].push\_back(found1);

price[found2].push\_back(bufPrice);

}

delete str;

}

indexStr = 0;

str = new char[100];

file >> str[indexStr];

while (str[indexStr] != ';') //first city

{

indexStr++;

file >> str[indexStr];

}

found1 = -1;

for (size\_t i = 0; (i < indexCity) && (found1 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found1 = i;

else {

found1 = -1;

break;

}

}

}

if (found1 == -1)

throw out\_of\_range("Missing value");

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //second city

{

indexStr++;

file >> str[indexStr];

}

found2 = -1;

for (size\_t i = 0; (i < indexCity) && (found2 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found2 = i;

else {

found2 = -1;

break;

}

}

}

if (found2 == -1)

throw out\_of\_range("Missing value");

this->from = found1;

this->to = found2;

this->result = NULL;

file.close();

}

void task3::readList(string fileName) {

fstream file;

file.open(fileName, ios::in);

int countStr;

file >> cityCount >> countStr;

if (file.eof() || cityCount == 0) throw out\_of\_range("Graph is empty");

adjacencyList = new dualList[cityCount];

price = new dualList[cityCount];

numberName = new City[cityCount];

char\* str;

int indexStr = 0;

int indexCity = 0;

int found1 = -1;

int found2 = -1;

int bufPrice;

for (size\_t i = 0; (i < countStr) && (!file.eof()); i++) {

indexStr = 0;

str = new char[100];

file >> str[indexStr];

while (str[indexStr] != ';') //first city

{

indexStr++;

file >> str[indexStr];

}

found1 = -1;

for (size\_t i = 0; (i < indexCity) && (found1 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found1 = i;

else {

found1 = -1;

break;

}

}

}

if (found1 == -1)

{

numberName[indexCity].nameSize = indexStr;

numberName[indexCity].name = new char[indexStr];

for (size\_t i = 0; i < indexStr; i++)

numberName[indexCity].name[i] = str[i];

found1 = indexCity;

indexCity++;

}

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //second city

{

indexStr++;

file >> str[indexStr];

}

found2 = -1;

for (size\_t i = 0; (i < indexCity) && (found2 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found2 = i;

else {

found2 = -1;

break;

}

}

}

if (found2 == -1)

{

numberName[indexCity].nameSize = indexStr;

numberName[indexCity].name = new char[indexStr];

for (size\_t i = 0; i < indexStr; i++)

numberName[indexCity].name[i] = str[i];

found2 = indexCity;

indexCity++;

}

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //first price

{

indexStr++;

file >> str[indexStr];

}

if (str[0] == 'N')

{

}

else {

bufPrice = 0;

for (size\_t i = 0; i < indexStr; i++)

{

switch (str[i]) //convert to int

{

case '0':bufPrice \*= 10; break;

case '1':bufPrice = bufPrice \* 10 + 1; break;

case '2':bufPrice = bufPrice \* 10 + 2; break;

case '3':bufPrice = bufPrice \* 10 + 3; break;

case '4':bufPrice = bufPrice \* 10 + 4; break;

case '5':bufPrice = bufPrice \* 10 + 5; break;

case '6':bufPrice = bufPrice \* 10 + 6; break;

case '7':bufPrice = bufPrice \* 10 + 7; break;

case '8':bufPrice = bufPrice \* 10 + 8; break;

case '9':bufPrice = bufPrice \* 10 + 9; break;

default:

break;

}

}

adjacencyList[found1].push\_back(found2);

price[found1].push\_back(bufPrice);

}

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //second price

{

indexStr++;

file >> str[indexStr];

}

if (str[0] == 'N')

{

}

else {

bufPrice = 0;

for (size\_t i = 0; i < indexStr; i++)

{

switch (str[i]) //convert to int

{

case '0':bufPrice \*= 10; break;

case '1':bufPrice = bufPrice \* 10 + 1; break;

case '2':bufPrice = bufPrice \* 10 + 2; break;

case '3':bufPrice = bufPrice \* 10 + 3; break;

case '4':bufPrice = bufPrice \* 10 + 4; break;

case '5':bufPrice = bufPrice \* 10 + 5; break;

case '6':bufPrice = bufPrice \* 10 + 6; break;

case '7':bufPrice = bufPrice \* 10 + 7; break;

case '8':bufPrice = bufPrice \* 10 + 8; break;

case '9':bufPrice = bufPrice \* 10 + 9; break;

default:

break;

}

}

adjacencyList[found2].push\_back(found1);

price[found2].push\_back(bufPrice);

}

delete str;

}

indexStr = 0;

str = new char[100];

file >> str[indexStr];

while (str[indexStr] != ';') //first city

{

indexStr++;

file >> str[indexStr];

}

found1 = -1;

for (size\_t i = 0; (i < indexCity) && (found1 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found1 = i;

else {

found1 = -1;

break;

}

}

}

if (found1 == -1)

throw out\_of\_range("Missing value");

delete str;

str = new char[100];

indexStr = 0;

file >> str[indexStr];

while (str[indexStr] != ';') //second city

{

indexStr++;

file >> str[indexStr];

}

found2 = -1;

for (size\_t i = 0; (i < indexCity) && (found2 == -1); i++) //find city

{

if (indexStr == numberName[i].nameSize) //check size

{

for (size\_t j = 0; j < numberName[i].nameSize; j++) //check name

if (str[j] == numberName[i].name[j])

found2 = i;

else {

found2 = -1;

break;

}

}

}

if (found2 == -1)

throw out\_of\_range("Missing value");

this->from = found1;

this->to = found2;

this->result = NULL;

file.close();

}

void task3::writeList(char\* fileName) {

if (result != NULL) {

fstream file;

file.open(fileName, ios::out);

for (size\_t i = 0; i < resultSize; i++)

{

for (size\_t j = 0; j < numberName[result[i]].nameSize; j++)

{

file << numberName[result[i]].name[j];

}

file << endl;

}

file.close();

}

}

void task3::writeList(string fileName) {

if (result != NULL) {

fstream file;

file.open(fileName, ios::out);

for (size\_t i = 0; i < resultSize; i++)

{

for (size\_t j = 0; j < numberName[result[i]].nameSize; j++)

{

file << numberName[result[i]].name[j];

}

file << endl;

}

file.close();

}

}