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»
Тема: Алгоритмы на графах

Студент гр. 8309 _____ Хваталов Д.И

Преподаватель _____ Тутуева А.В

Санкт-Петербург
2020

Исходная формулировка задания:

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.

Цель работы:

Реализовать алгоритм Флойда-Уоршелла и матрицу смежности.

Постановка задачи:

Необходимо написать программу способную найти наиболее эффективный по стоимости перелет из города A в город B. Для этого сделаем класс в котором будет храниться список городов и матрицу смежности с ценой перелета из пункта A в пункт B.

Организация данных:

Название	Описание работы метода	Оценка временной сложности
`void assemble(string fLink)`	Метод который формирует список городов путем обхода текстового файла, после чего формируется матрица смежности, куда заносится стоимость перелёта. В конце вызывается метод реализующий алгоритм Флойда – Уоршелла.	O((N*(K+K+Z))+X+(N*(K+K))+X^2+V^3) Создание списка городов Создание матрицы смежности Заполнение матрицы смежности Объявление недоступных путей. Алгоритм Флойда-Уоршелла.
`void printCities()`	Выводит список городов, а также матрицу смежности с ценой перелета из одного города в другой.	O(N+K^2)
`void printMatrix()`	Выводит матрицу смежности с ценами перелета.	O(N^2)
`int yourFlight(int cityWhence, int cityWhere)`	Возвращает стоимость перелета из пункта A в пункт B.	O(1)

`void clear()`	Очищает память, выделенную под массив и список.	O(N+K)
`void createMatrix()`	Создает матрицу смежности на размер +1 городов (так как количество городов считается с нуля)	O(N)
`nodeList* find(string cityName)`	Возвращает узел с городом или же пустой, используется при добавлении нового города, а также при заполнении матрицы смежности.	O(N)
`void pushBack(string cityName)`	Добавляет новый узел списка в конец.	O(1) – первый узел O(N) – Последующие узлы
`void originalFloydWarshall()`	Алгоритм Флойда-Уоршелла	O(N^3)

Название Unit-теста	**Описание работы**
`TEST_METHOD(assembleError)`	Проверяем ошибку при вводе не правильной ссылки на файл.
`TEST_METHOD(PrintCitiesError)`	Проверяем ошибку, при которой пытаемся вывести список городов не сформировав его.
`TEST_METHOD(PrintMatrixError)`	Проверяем ошибку, при которой пытаемся вывести матрицу смежности не сформировав ее.
`TEST_METHOD(YourFlightError)`	Проверяем ошибку, при которой пользователь ввел недоступные города.
`TEST_METHOD(YourFlightTrue)`	Правильная работа программы.

Код программы

```
#include <iostream>
#include <string>
#include <fstream>
#include <stdlib.h>
#include <algorithm>
using namespace std;
#define N_A 1001;


class Flights
{
public:
	Flights();
	~Flights();
	void assemble(string fLink)
	{
		setlocale(0, "");
		fileLink = fLink;
		fstream file(fileLink, ios::in);
		if (!file.is_open()) {
			throw invalid_argument("file link error");
		}

		while (!file.eof()) {
			string city;
			getline(file, city, ';');
			pushBack(city);
			getline(file, city, ';');
			pushBack(city);
			while (file.get() != '\n' && !file.eof());
		}
		file.close();
		file.open(fileLink, ios::in);
		createMatrix();
		while (!file.eof())
		{
			string city;
			getline(file, city, ';');
			nodeList *firstCity = find(city);
			getline(file, city, ';');
			nodeList* secondCity = find(city);
			string next;
			string back;
			getline(file, next, ';');
			if (next != "N/A")
			{
				int temp=0;
				temp = atoi(next.c_str());
				matrix[firstCity->num][secondCity->num] = temp;
			}
			else
			{
				matrix[firstCity->num][secondCity->num] = N_A;
			}
			getline(file, back, '\n');
			if (back != "N/A")
			{
				int temp = 0;
				temp = atoi(back.c_str());
```

```
                        matrix[secondCity->num][firstCity->num] = temp;
                }
                else
                {
                        matrix[secondCity->num][firstCity->num] = N_A;
                }
        }
        for (int i = 0; i < listTail->num + 1; i++)
        {
                for (int j = 0; j < listTail->num + 1; j++)
                {
                        if (i != j && matrix[i][j] == 0)
                        {
                                matrix[i][j] = N_A;
                        }
                }
        }
        originalFloydWarshall();
}


void printCities()
{
        if (listHead == nullptr)
        {
                throw out_of_range("Lists Empty");
        }
        nodeList* bypass = listHead;
        while (bypass != nullptr)
        {
                cout <<bypass->num<<"."<< bypass->cityName << "\t";
                bypass = bypass->next;
        }
        printMatrix();
}

void printMatrix()
{
        if (matrix == nullptr)
        {
                throw out_of_range("Matrix Empty");
        }
        cout << endl;
        for (int i = 0; i < listTail->num + 1; i++)
        {
                cout << i << "| ";
                for (int j = 0; j < listTail->num + 1; j++)
                {
                        if (matrix[i][j] == 1001)
                        {
                                cout << "N/A\t";
                        }
                        else
                        {
                                cout << matrix[i][j] << "\t";
                        }

                }
                cout << endl;
        }
}

int yourFlight(int cityWhence, int cityWhere)
```

```
	{
		if (cityWhence > listTail->num || cityWhere > listTail->num)
		{
			throw out_of_range("Error - incorrect cities");
		}
		return matrix[cityWhence][cityWhere];
	}

	void clear()
	{
		for (int i = 0; i < listTail->num + 1; i++)
		{
			delete[] matrix[i];
		}
		delete[] matrix;
		while (listHead != nullptr)
		{
			nodeList* del = listHead;
			listHead = listHead->next;
			delete del;
		}
		listTail = nullptr;
	}


private:

	struct nodeList
	{
		int num=0;
		string cityName;
		nodeList* next = nullptr;
	};
	nodeList* listHead;
	nodeList* listTail;
	string fileLink;
	int** matrix;

	void createMatrix()
	{
		matrix = new int* [listTail->num+1];
		for (int i = 0; i < listTail->num+1; i++)
		{
			matrix[i] = new int[listTail->num+1]{};
		}
	}

	nodeList* find(string cityName)
	{
		nodeList* bypass = listHead;
		while (bypass != nullptr && bypass->cityName != cityName)
		{
			bypass = bypass->next;
		}
		return bypass;
	}

	void pushBack(string cityName)
	{
		if (listHead == nullptr)
		{
			listHead = new nodeList;
			listHead->cityName = cityName;
```

```
                        listTail = listHead;
                }
                else
                {
                        if (find(cityName) == nullptr)
                        {
                                listTail->next = new nodeList;
                                listTail->next->cityName = cityName;
                                listTail->next->num = listTail->num + 1;
                                listTail = listTail->next;
                        }
                }
        }

        void originalFloydWarshall()
        {
                for (int k = 0; k < listTail->num + 1; k++) {
                        for (int i = 0; i < listTail->num + 1; i++) {
                                for (int j = 0; j < listTail->num + 1; j++) {

                                        matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j]);
                                }
                        }
                }

                return;
        }




};

Flights::Flights()
{
        listHead = nullptr;
        listTail = listHead;
        matrix = nullptr;
}

Flights::~Flights()
{
        clear();
        cout << "all clear";
}
```

Source.cpp

```
#include"Header.h"

int main()
{
        Flights myFlight;
        string link;
        cin >> link;
        myFlight.assemble(link);
        myFlight.printCities();
        bool ret= true;
        while (ret)
        {
```

```
            cout << "Enter the number city you want to fly from - ";
            int cityWhence;
            cin >> cityWhence;
            cout << "enter the city you want to fly to - ";
            int cityWhere;
            cin >> cityWhere;
            cout << "value - " << myFlight.yourFlight(cityWhence, cityWhere) << endl;
            cout << "try again? 1 - yes | 0 - no  ";
            cin >>ret;
        }

        return 0;
}
```

Контрольные примеры

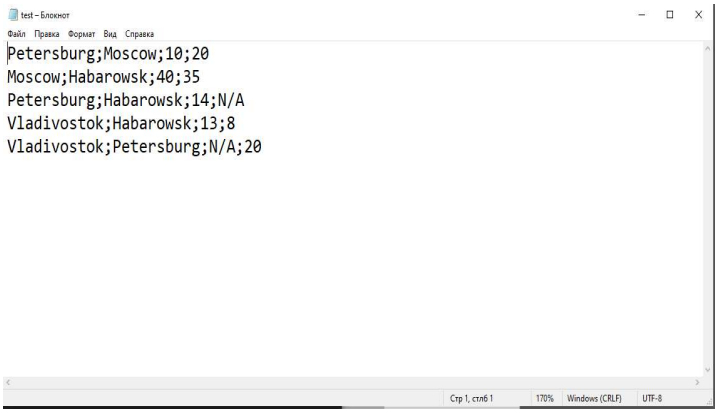

Рисунок 1 Изначальные данные

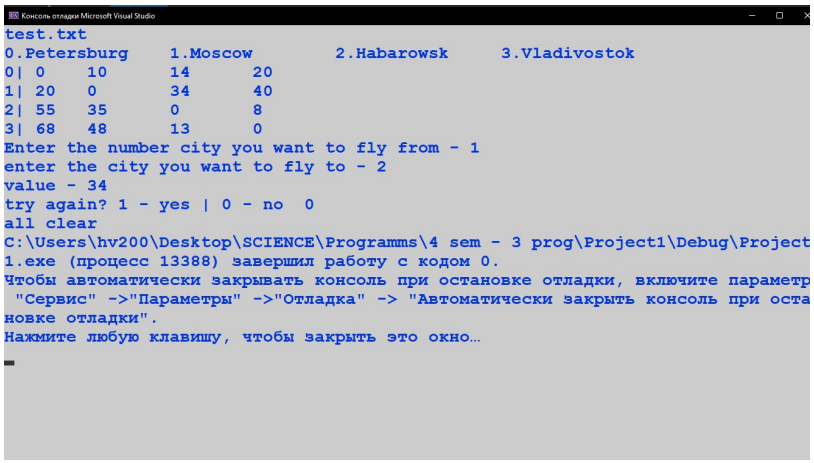

Рисунок 2 Данные после обработки, получилось сэкономить 6 поинтов, на перелете и Москвы, в Хабаровск

Вывод

Научился работать с динамическим двумерным массивом и применять алгоритм Флойда-Уоршелла на графе.