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

Вариант: 3

Студент гр. 8309 Носов А.С.

Преподаватель Тутуева А.В.

Санкт-Петербург

2020

Оглавление

Постановка задачи. Описание реализуемого класса и методов 2

Оценка временной сложности каждого метода 2

Описание реализованных unit-тестов 3

Листинг 5

Постановка задачи. Описание реализуемого класса и методов

Реализовать алгоритм Флойда-Уоршелла и матрицу смежности.
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.

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;Калининград;40;35
Санкт-Петербург;Калининград;14;N/A
Киев;Москва;13;8
Киев;Санкт-Петербург;N/A;20
Киев;Калининград;13;8
Задание: найти наиболее эффективный по стоимости перелет из города i в город j.

Описание методов:

1. 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.
2. void printCities(): Выводит список городов, а также матрицу смежности с ценой перелета из одного города в другой.
3. void printMatrix(): Выводит матрицу смежности с ценами перелета.
4. int yourFlight(int cityWhence, int cityWhere): Возвращает стоимость перелета из пункта A в пункт B.
5. void clear(): Очищает память, выделенную под массив и список.
6. void createMatrix(): Создает матрицу смежности на размер +1 городов (так как количество городов считается с нуля)
7. nodeList* find(string cityName): Возвращает узел с городом или же пустой, используется при добавлении нового города, а также при заполнении матрицы смежности.
8. void pushBack(string cityName): Добавляет новый узел списка в конец.
9. void originalFloydWarshall(): Алгоритм Флойда-Уоршелла.

Оценка временной сложности каждого метода

Временная сложность оценена с помощью модульных тестов.

10. void assemble(string fLink): O((N*(K+K+Z))+X+(N*(K+K))+(X*X)+(V*V*V)), где
    (N*(K+K+Z)) – создание списка городов
    X – создание матрицы смежности
    (N*(K+K)) – заполнение матрицы смежности

(X*X) – объявление недоступных путей
(V*V*V) – Алгоритм Флойда-Уоршелла

11. void printCities(): O(N+K*K)
12. void printMatrix(): O(N*N)
13. int yourFlight(int cityWhence, int cityWhere): O(1)
14. void clear(): O(N+K)
15. void createMatrix(): O(N)
16. nodeList* find(string cityName): O(N)
17. void pushBack(string cityName): 1) O(1) – создание первого узла; 2) O(N) – создание послед. у
18. void originalFloydWarshall(): O(N*N*N)

Описание реализованных unit-тестов

Модульные тесты нужны для сравнительно эффективного тестирования программы, удобный как для разработчика, так и другого пользователя.

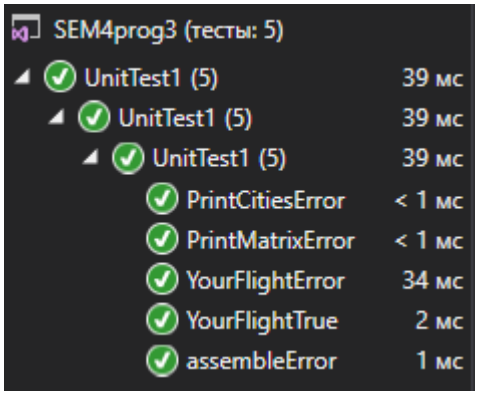
Я использовал предустановленный в Visual Studio шаблон для написания unit-тестов. Созданная мною среда для каждого теста была примерно одна и та же, в основном отличаясь лишь наличием того или иного метода.

Ключевым элементов всех моих тестов было наличие сравнение практического и теоретического результата с помощью Assert::AreEqual().

Проект с юнит-тестами был реализован в одном решении с основной программой

Пример работы

```
C:\Users\Art\Desktop\test.txt
0.Petersburg    1.Moscow        2.Kaliningrad   3.Kiev
0| 0    10      14      18
1| 20   0       21      8
2| 41   21      0       8
3| 33   13      13      0
Enter the number city you want to fly from - 0
enter the city you want to fly to - 2
value - 14
try again? 1 - yes | 0 - no  1
Enter the number city you want to fly from - 0
enter the city you want to fly to - 3
value - 18
try again? 1 - yes | 0 - no  0
all clear
C:\Users\Art\source\repos\SEM4prog3\Debug\SEM4prog3.exe (
Чтобы автоматически закрывать консоль при остановке отлад
 "Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу…
```


SEM4prog3 (тесты: 5)
UnitTest1 (5) 39 мс
UnitTest1 (5) 39 мс
UnitTest1 (5) 39 мс
PrintCitiesError < 1 мс
PrintMatrixError < 1 мс
YourFlightError 34 мс
YourFlightTrue 2 мс
assembleError 1 мс

Листинг

Head.h

```
#pragma once
#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";
}
```

main.cpp

```
#include"Head.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;
}
```

UnitTest1.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "main.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
	TEST_CLASS(UnitTest1)
	{
	public:

		TEST_METHOD(assembleError)
		{
			Flights myFlight;
			try
			{
				myFlight.assemble("nullptr");
			}
			catch (invalid_argument error)
			{
				Assert::AreEqual("file link error", error.what());
			}
		}

		TEST_METHOD(PrintCitiesError)
		{
			Flights myFlight;
			try
			{
				myFlight.printCities();
			}
			catch (out_of_range error)
			{
				Assert::AreEqual("Lists Empty", error.what());
			}

		}
```

```
		TEST_METHOD(PrintMatrixError)
		{
			Flights myFlight;
			try
			{
				myFlight.printMatrix();
			}
			catch (out_of_range error)
			{
				Assert::AreEqual("Matrix Empty", error.what());
			}

		}
		TEST_METHOD(YourFlightError)
		{
			Flights myFlight;
			myFlight.assemble("C:\\Users\\Art\\Desktop\\test.txt");
			try
			{

				myFlight.yourFlight(5, 100);
			}
			catch (out_of_range error)
			{
				Assert::AreEqual("Error - incorrect cities",
error.what());
			}
			myFlight.clear();
		}

		TEST_METHOD(YourFlightTrue)
		{
			Flights myFlight;
			myFlight.assemble("C:\\Users\\Art\\Desktop\\test.txt");
			Assert::AreEqual(34, myFlight.yourFlight(1, 2));
			myFlight.clear();
		}

	};
}
```