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»

Вариант 3

Студент гр. 8301 _____ Готовский К.В.

Преподаватель _____ Тутева А.В.

Санкт-Петербург

2020

Цель работы

Реализовать программу принимающую список рейсов и цены за прямой и обратный и рейс и, в которой пользователь в свою очередь вводит город отправления и назначения и получает самый выгодный рейс или получает информацию о невозможности совершения перелётов методом Флойда-Уоршелла.

Описание реализуемого класса

Класс Matrix, содержит поля:double** matrix (двумерный массив цен на рейсы, схожий с матрицей смежности), int size_of_matrix (размер матрицы смежности), Map<string, int>* map_City_name_to_index (для хранения названия и получения его индекса), Map<int, string>* map_index_to_name_City (хранения индекса и получения его названия города), const int INF (максимальное значение, которого невозможно достигнуть (необходимо для реализации)). Класс содержит следующие методы:

- Констуктор – получает на вход список строк, обрабатывает их и в результате выдает матрицу смежности, где по горизонтали начальный пункт, по вертикали – конечный, и на пересечении строк – цена перелёта из одного пункта в другой.
- Деструктор – вызывает метод clear (на основе обычного удаления двоичного дерева).
- *void print_path (int i, int j, int** p, Map<int, string>* map_index_to_name_City, string&cur)* – функция, рекурсивно записывающая путь в строку.

Оценка временной сложности алгоритмов

- *string Floid_Uorshell(string start_City,string end_City)* – O(N^3)
- *get_list_symbol()* – O(1)
- *print_path (int i, int j, int** p, Map<int, string>* map_index_to_name_City, string&cur)* – O(N^2)

Описание реализованных unit-тестов

Реализованные мною тесты проверяют правильное нахождение выгодного перелёта. Я рассмотрел две ситуации когда перелёт возможен и когда нет.

Пример работы программы

Пример

```
C:\Users\1\source\repos\Lab 3\x64\Debug\Floyd_Uorshell.exe
Flight schedule:
Saint Petersburg; Moscow; 10; 20
Moscow; Khabarovsk; 40; 35
Saint Petersburg; Khabarovsk; 14; N/A
Vladivostok; Khabarovsk; 13; 8
Vladivostok; Saint Petersburg; 20; N/A
Enter the departure city
Khabarovsk
Enter your arrival city
Saint Petersburg
The best route for the price: 28,000000
Route: Khabarovsk -> Vladivostok -> Saint Petersburg
Для продолжения нажмите любую клавишу . . .
```

Листинг

matrix_of_adjacencies.h

```
#pragma once
#include"List.h"
#include"Map.h"
#include<string>
class Matrix {
public:
    Matrix(List<string>* data) {
            map_City_name_to_index=new Map<string, int>();
            map_index_to_name_City=new Map<int, string>();
            int N = data->get_size();
            int index_city = 0;
            for (int i = 0; i < N; i++) {
                string str_cur = data->at(i);
                int cur = str_cur.find(';');//the first occurrence
                int cur1 = str_cur.find(';', cur + 1);//the second occurrence
                string str_name_city1 = str_cur.substr(0, cur);//get first city
                string str_name_city2 = str_cur.substr(cur + 1, cur1 - cur - 1);//get second city
                str_name_city2.erase(0, 1);
                if (!map_City_name_to_index->find_is(str_name_city1)) {
                    map_City_name_to_index->insert(str_name_city1, index_city);
                    map_index_to_name_City->insert(index_city, str_name_city1);
                    index_city++;
                }
                if (!map_City_name_to_index->find_is(str_name_city2)) {
                    map_City_name_to_index->insert(str_name_city2, index_city);
                    map_index_to_name_City->insert(index_city, str_name_city2);
                    index_city++;
                }
            }
            ////////////////////////////////////////////////////////////////////////////make matrix path
            size_of_matrix = index_city;
            matrix = new double* [size_of_matrix];
            for (int i = 0; i < size_of_matrix; i++)
                matrix[i] = new double[size_of_matrix];
            for (int i = 0; i < size_of_matrix; i++)
                for (int j = 0; j < size_of_matrix; j++)
                    matrix[i][j] = INF;
            ////////////////////////////////////////////////////////////////////////////

            ////////////////////////////////////////////////////////////////////////////input matrix pat
            for (int i = 0; i < N; i++) {
                int price_1_to_2 = INF;
```

```
43.                 int price_2_to_1 = INF;
44.                 string str_cur = data->at(i);
45.                 int cur = str_cur.find(';');
46.                 int cur1 = str_cur.find(';', cur + 1);
47.                 int cur2 = str_cur.find(';', cur1 + 1);
48.                 int cur3 = str_cur.find(';', cur2 + 1);
49.                 string str_name_city1 = str_cur.substr(0, cur);
50.                 string str_name_city2 = str_cur.substr(cur + 1, cur1 - cur - 1);
51.                 str_name_city2.erase(0, 1);
52.                 if (str_cur.substr(cur1 + 2, cur2 - 2 - cur1) != "N/A")
53.                     price_1_to_2 = stof(str_cur.substr(cur1 + 2, cur2 - 2 - cur1));
54.                 if (str_cur.substr(cur2 + 2, cur3 - 1) != "N/A")
55.                     price_2_to_1 = stoi(str_cur.substr(cur2 + 2, cur3 - 2 - cur2));
56.
57.                 matrix[map_City_name_to_index->find(str_name_city1)][map_City_name_to_index-
    >find(str_name_city2)] = price_1_to_2;
58.
59.                 matrix[map_City_name_to_index->find(str_name_city2)][map_City_name_to_index-
    >find(str_name_city1)] = price_2_to_1;
60.             }
61.             ////////////////////////////////////////////////////////////////////////////////////////////
62.     }
63.     string Floid_Uorshell(string start_City,string end_City) {
64.         string cur;
65.         while (!map_City_name_to_index->find_is(start_City)) {
66.             cout << "The departure city is missing, enter it again" << endl;
67.             cin >> start_City;
68.         }
69.         while (!map_City_name_to_index->find_is(end_City)) {
70.             cout << "The arrival city is missing, enter it again" << endl;
71.             cin >> end_City;
72.         }
73.         int index_start_vertex = map_City_name_to_index->find(start_City);
74.         int index_end_vertex = map_City_name_to_index->find(end_City);
75.         int** pre = new int* [size_of_matrix];
76.         for (int i = 0; i < size_of_matrix; i++) {
77.             pre[i] = new int[size_of_matrix];
78.             for (int j = 0; j < size_of_matrix; j++)
79.                 pre[i][j] = i;
80.         }
81.         for (int k = 0; k < size_of_matrix; ++k)
82.             for (int i = 0; i < size_of_matrix; ++i)
83.                 for (int j = 0; j < size_of_matrix; ++j) {
84.                     if (matrix[i][k] + matrix[k][j] < matrix[i][j]) {
85.                         matrix[i][j] = matrix[i][k] + matrix[k][j];
86.                         pre[i][j] = pre[k][j];
87.                     }
88.                 }
89.         if (matrix[map_City_name_to_index->find(start_City)][map_City_name_to_index->find(end_City)] != IN
90.             cur = "The best route for the price: " + to_string(matrix[map_City_name_to_index-
    >find(start_City)][map_City_name_to_index->find(end_City)]) + '\n' + "Route: ";
91.             print_path(index_start_vertex, index_end_vertex, pre, map_index_to_name_City, cur);
92.             cur.erase(cur.size() - 3);
93.         }
94.         else {
95.             cur = "This route can't be built, try waiting for the flight schedule for tomorrow!";
96.         }
97.         return cur;
98.     }
99. private:
100.         void print_path(int i, int j, int** p, Map<int, string>* map_index_to_name_City,string&cur) {
101.             if (i != j)
102.                 print_path(i, p[i][j], p, map_index_to_name_City,cur);
103.             cur=cur+map_index_to_name_City->find(j)+" -> ";
104.         }
105.         double** matrix;
106.         int size_of_matrix;
```

```
107.            Map<string, int>* map_City_name_to_index;
108.            Map<int, string>* map_index_to_name_City;
109.            const int INF = 1000000000;
110.        };
```

UnitTestForFloyd_Uorshell.cpp

```
1.  #include "pch.h"
2.  #include "CppUnitTest.h"
3.  #include <fstream>
4.  #include<string>
5.  #include"..//matrix_of_adjacencies.h"
6.  #include"..//Used_function.h"
7.  using namespace Microsoft::VisualStudio::CppUnitTestFramework;
8.  namespace UnitTestForAlgorithmFloydUorshell
9.  {
10.     TEST_CLASS(UnitTestForAlgorithmFloydUorshell)
11.     {
12.     public:
13.
14.         TEST_METHOD(TestExamplePath_is_avaible)
15.         {
16.             ifstream vvod("C:\\Users\\ROG\\source\\repos\\AISDlab3var3()\\UnitTestForFloyd_Uorshell\\input
17.             List<string>* list_fly = new List<string>();
18.             string city_Start = "Vladivostok";
19.             string city_End = "Moscow";
20.             InputDataFromFile(list_fly, vvod);
21.             Matrix* matrix_floid_uorshell = new Matrix(list_fly);
22.             string cur = "The best route for the price: 30.000000\nRoute: Vladivostok -> Saint Petersburg
    > Moscow ";
23.             Assert::AreEqual(matrix_floid_uorshell->Floid_Uorshell(city_Start, city_End), cur);
24.         }
25.         TEST_METHOD(TestExamplePath_is_not_avaible)
26.         {
27.             ifstream vvod("C:\\Users\\ROG\\source\\repos\\AISDlab3var3()\\UnitTestForFloyd_Uorshell\\input
28.             List<string>* list_fly = new List<string>();
29.             string city_Start = "Tambov";
30.             string city_End = "Saint Petersburg";
31.             InputDataFromFile(list_fly, vvod);
32.             Matrix* matrix_floid_uorshell = new Matrix(list_fly);
33.             string cur ="This route can't be built, try waiting for the flight schedule for tomorrow!";
34.             Assert::AreEqual(matrix_floid_uorshell->Floid_Uorshell(city_Start, city_End), cur);
35.         }
36.     };
37. }
```

Вывод

В данной лабораторной работе я ознакомился с алгоритмом Флойда-Уоршелла и смог применить его на примере нахождения выгодного пути из авиарейсов, а также закрепил свои навыки в объектно-ориентированном программировании.