**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 |  | Чередилина Е. А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

Цель работы

Целью данной лабораторной работы является реализация программы, принимающей список рейсов и цены за прямой и обратный и рейс и, в которой пользователь в свою может ввести город отправления и назначения, и получить самый выгодный рейс или информацию о невозможности совершения перелётов методом Флойда-Уоршелла.

Описание реализуемых и вспомогательных классов

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:

* *Конструктор* – 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

Я реализовала unit-тесты, проверяющие правильное нахождение выгодного перелёта, а также рассмотрела две ситуации, когда перелёт возможен и когда нет.

Проверила и две исключительные ситуации, когда адрес файл был введен некорректно, либо же отсутствует, и, когда город отправления совпадает с городом прибытия.

Пример работы программы

|  |
| --- |
| Пример работы программы |
|  |

Листинг

|  |
| --- |
| matrix\_of\_adjacencies.h |
| 1. #pragma once 2. #include"List.h" 3. #include"Map.h" 4. #include<string> 5. **class** Matrix { 6. **public**: 7. Matrix(List<string>\* data) { 8. map\_City\_name\_to\_index=**new** Map<string, **int**>(); 9. map\_index\_to\_name\_City=**new** Map<**int**, string>(); 10. **int** N = data->get\_size(); 11. **int** index\_city = 0; 12. **for** (**int** i = 0; i < N; i++) { 13. string str\_cur = data->at(i); 14. **int** cur = str\_cur.find(';');//the first occurrence 15. **int** cur1 = str\_cur.find(';', cur + 1);//the second occurrence 16. string str\_name\_city1 = str\_cur.substr(0, cur);//get first city 17. string str\_name\_city2 = str\_cur.substr(cur + 1, cur1 - cur - 1);//get second city 18. str\_name\_city2.erase(0, 1); 19. **if** (!map\_City\_name\_to\_index->find\_is(str\_name\_city1)) { 20. map\_City\_name\_to\_index->insert(str\_name\_city1, index\_city); 21. map\_index\_to\_name\_City->insert(index\_city, str\_name\_city1); 22. index\_city++; 23. } 24. **if** (!map\_City\_name\_to\_index->find\_is(str\_name\_city2)) { 25. map\_City\_name\_to\_index->insert(str\_name\_city2, index\_city); 26. map\_index\_to\_name\_City->insert(index\_city, str\_name\_city2); 27. index\_city++; 28. } 29. } 30. /////////////////////////////////////////////////////////////////////////////make matrix path 31. size\_of\_matrix = index\_city; 32. matrix = **new** **double**\* [size\_of\_matrix]; 33. **for** (**int** i = 0; i < size\_of\_matrix; i++) 34. matrix[i] = **new** **double**[size\_of\_matrix]; 35. **for** (**int** i = 0; i < size\_of\_matrix; i++) 36. **for** (**int** j = 0; j < size\_of\_matrix; j++) 37. matrix[i][j] = INF; 38. ///////////////////////////////////////////////////////////////////////////// 40. //////////////////////////////////////////////////////////////////////////////input matrix path 41. **for** (**int** i = 0; i < N; i++) { 42. **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)); 57. matrix[map\_City\_name\_to\_index->find(str\_name\_city1)][map\_City\_name\_to\_index->find(str\_name\_city2)] = price\_1\_to\_2; 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)] != INF) { 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"../Input\_data\_from\_file\_function.h" 4. #include"../List.h" 5. #include"../Floyd\_Uorshell\_algorithm.h" 6. #include<string> 7. using namespace Microsoft::VisualStudio::CppUnitTestFramework; 8. namespace UnitTestforFloydUorshellalgorithm 9. { 10. TEST\_CLASS(UnitTestforFloydUorshellalgorithm) 11. { 12. public: 14. TEST\_METHOD(TestExamplePath\_is\_avaible) 15. { 16. List<string>\* list\_fly = new List<string>(); 17. string city\_Start = "Sochi"; 18. string city\_End = "Moscow"; 19. InputDataFromFile(list\_fly, "C:\\Users\\chere\\Desktop\\АиСД-4сем\\LR3\\Floid\_Uorshell\\UnitTest\_for\_Floyd\_Uorshell\_algorithm\\input1.txt"); 20. Matrix\* matrix\_floid\_uorshell = new Matrix(list\_fly); 21. string cur = "The best route for the price: 30.000000\nRoute: Sochi -> Saint Petersburg -> Moscow "; 22. Assert::AreEqual(matrix\_floid\_uorshell->Floid\_Uorshell(city\_Start, city\_End), cur); 23. } 24. TEST\_METHOD(TestExamplePath\_is\_not\_avaible) 25. { 26. List<string>\* list\_fly = new List<string>(); 27. string city\_Start = "Chita"; 28. string city\_End = "Saint Petersburg"; 29. InputDataFromFile(list\_fly, "C:\\Users\\chere\\Desktop\\АиСД-4сем\\LR3\\Floid\_Uorshell\\UnitTest\_for\_Floyd\_Uorshell\_algorithm\\input2.txt"); 30. Matrix\* matrix\_floid\_uorshell = new Matrix(list\_fly); 31. string cur = "This route can't be built, try waiting for the flight schedule for tomorrow!"; 32. Assert::AreEqual(matrix\_floid\_uorshell->Floid\_Uorshell(city\_Start, city\_End), cur); 33. } 34. TEST\_METHOD(TestException\_the\_path\_to\_itself) 35. { 36. try { 37. List<string>\* list\_fly = new List<string>(); 38. string city\_Start = "Chita"; 39. string city\_End = "Chita"; 40. InputDataFromFile(list\_fly, "C:\\Users\\chere\\Desktop\\АиСД-4сем\\LR3\\Floid\_Uorshell\\UnitTest\_for\_Floyd\_Uorshell\_algorithm\\input2.txt"); 41. Matrix\* matrix\_floid\_uorshell = new Matrix(list\_fly); 42. } 43. catch (exception & ex) { 44. string cur = "The path to itself is impossible!"; 45. Assert::AreEqual(ex.what(), cur.c\_str()); 46. } 47. } 48. TEST\_METHOD(TestException\_file\_is\_missing) 49. { 50. try { 51. List<string>\* list\_fly = new List<string>(); 52. InputDataFromFile(list\_fly, "C:\\Users\\chere\\Desktop\\АиСД-4сем\\LR3\\Floid\_Uorshell\\UnitTest\_for\_Floyd\_Uorshell\_algorithm\\inpu.txt"); 53. string city\_Start = "Chita"; 54. string city\_End = "Chita"; 55. Matrix\* matrix\_floid\_uorshell = new Matrix(list\_fly); 56. } 57. catch (exception & ex) { 58. string cur = "File is missing!"; 59. Assert::AreEqual(ex.what(), cur.c\_str()); 60. } 61. } 62. }; 63. } |

|  |
| --- |
| Used\_function.h |
| 1. #pragma once 2. #include<string> 3. #include <fstream> 4. #include"List.h" 5. **void** InputDataFromFile(List<string>\* data, ifstream& file) { 6. **while** (!file.eof()) { 7. string s1; 8. getline(file, s1); 9. data->push\_back(s1); 10. } 11. } |

Работа юнит тестов:

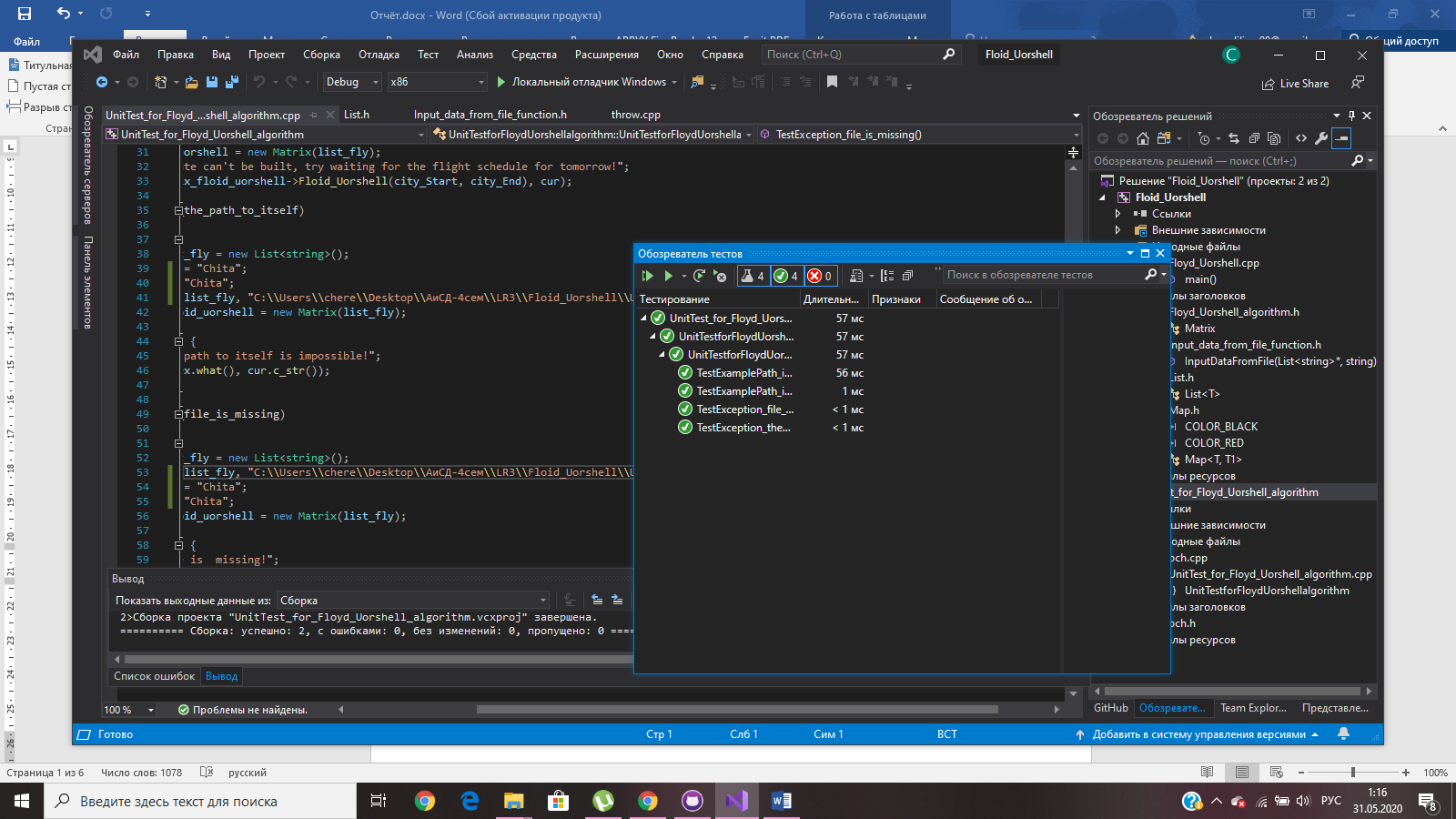


Рисунок 2 – Юнит тесты

Вывод

При выполнении данной лабораторной работы я ознакомилась с алгоритмом Флойда-Уоршелла и смогла применить его на примере нахождения выгодного пути из авиарейсов.