**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. 8301 |  | Онвука П.А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

## Цель работы

нахождение наиболее эффективный по стоимости перелет из города ***i*** в город ***j*** использованием [алгоритма Флойда-Уоршелла](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A4%D0%BB%D0%BE%D0%B9%D0%B4%D0%B0_%E2%80%94_%D0%A3%D0%BE%D1%80%D1%88%D0%B5%D0%BB%D0%BB%D0%B0) и матрицу смежности

## Описание программы

**Алгоритм Флойда (алгоритм Флойда–Уоршелла)** — алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе. Работает корректно, если в графе нет циклов отрицательной величины, а в случае, когда такой цикл есть, позволяет найти хотя бы один такой цикл.

Обозначим длину кратчайшего пути между вершинами u и v, содержащего, помимо u и v, только вершины из множества {1..i}{1..i} как d(i)uvduv(i), d(0)uv=ωuvduv(0)=ωuv.

На каждом шаге алгоритма, мы будем брать очередную вершину (пусть её номер — i) и для всех пар вершин u и v вычислять d(i)uv=min(d(i−1)uv,d(i−1)ui+d(i−1)iv)duv(i)=min(duv(i−1),dui(i−1)+div(i−1)). То есть, если кратчайший путь из u в v, содержащий только вершины из множества {1..i}{1..i}, проходит через вершину i, то кратчайшим путем из u в v является кратчайший путь из u в i, объединенный с кратчайшим путем из i в v. В противном случае, когда этот путь не содержит вершины i, кратчайший путь из u в v, содержащий только вершины из множества {1..i}{1..i} является кратчайшим путем из u в v, содержащим только вершины из множества {1..i−1}{1..i−1}.

В данной программе используются следующие структуры данных:

2D массив для хранения информации предшественника

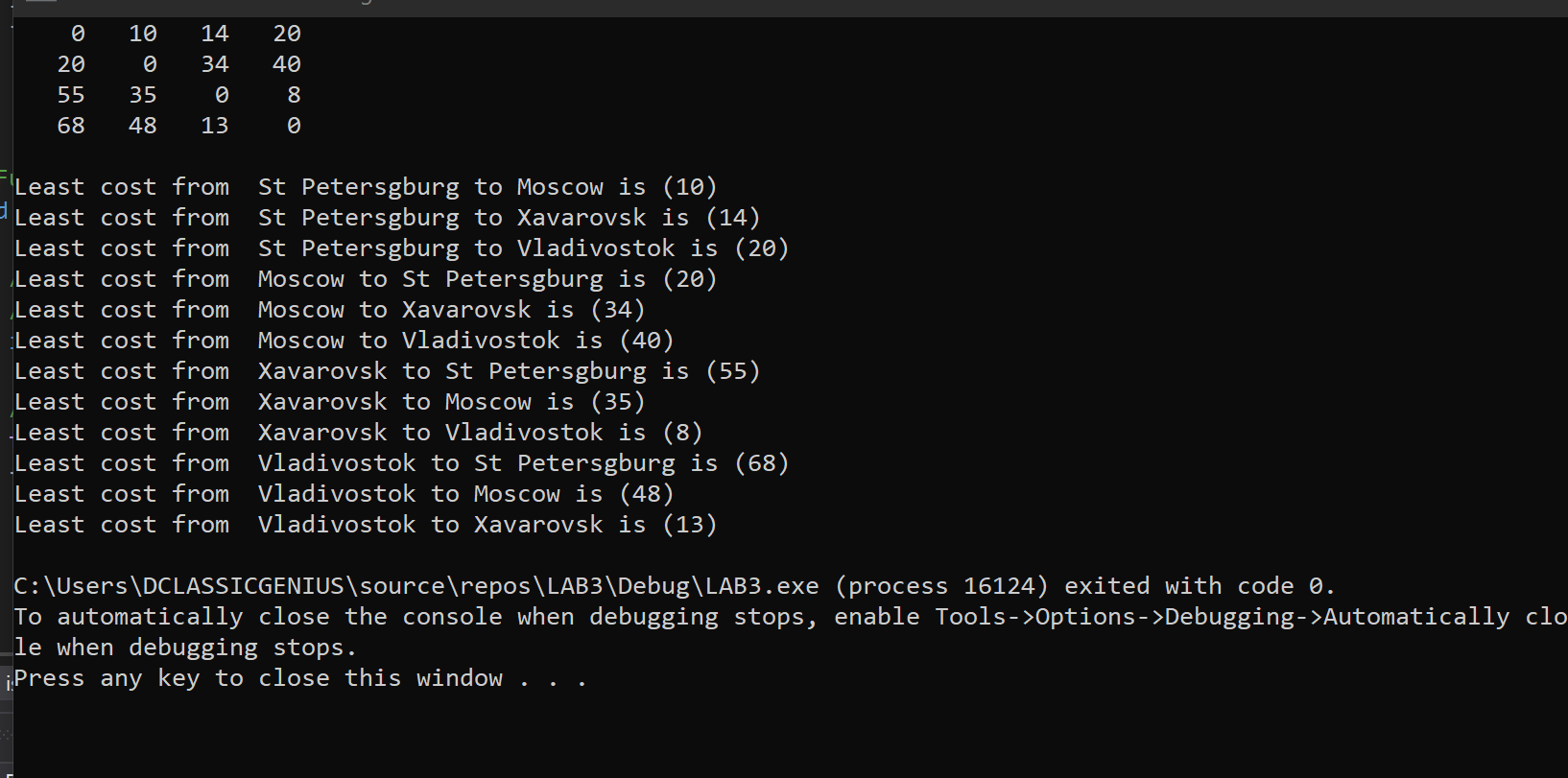
Матрица смежности для представления конечного графа

## Оценка временной сложности методов

1. **printSolution** О(N2)
2. **floydWarshell** O(|V|3)

где N - количество циклов, а V - количество вершин

## Примеры работы



## Листинг

#include <iostream>

#include <climits>

#include <iomanip>

#include <string>

using namespace std;

// Number of vertices in the adjacency Matrix

#define N 4

// Recursive Function to print path of given

// vertex i from source vertex j

void printPath(int path[][N], int i, int j)

{

if (path[i][j] == i)

return;

printPath(path, i, path[i][j]);

cout << path[i][j] << " ";

}

string namecity(int index) {

switch (index) {

case 0: return "St Petersburg";

break;

case 1: return "Moscow";

break;

case 2: return "Xavarovsk";

break;

case 3: return "Vladivostok";

break;

default:

throw invalid\_argument("Invalid city name" );

}

}

// Function to print the least cost

// between all pairs of cities

void printSolution(int cost[N][N], int path[N][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

if (cost[i][j] == INT\_MAX)

cout << setw(5) << "inf";

else

cout << setw(5) << cost[i][j];

}

cout << endl;

}

cout << endl;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

if (j != i )

{

cout << "Least cost from " << namecity(i) <<

" to " << namecity(j) << " is (" << cost[i][j] <<")"<< endl;

}

}

}

}

// Function to run Floyd-Warshell algorithm

void FloydWarshell(int adjacencyMatrix[][N])

{

// cost[] stores shortest cost and path[] stores the shortest route information

int cost[N][N], path[N][N];

// initialize cost[] and path[]

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

// initially cost would be same as weight

// of the edge of the graph

cost[i][j] = adjacencyMatrix[i][j];

if (i == j)

path[i][j] = 0;

else if (cost[i][j] != INT\_MAX)

path[i][j] = i;

}

}

for (int k = 0; k < N; k++)

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

// If vertex k is the least cost from i to j,

// then update the value of cost[i][j], path[i][j]

if (cost[i][k] != INT\_MAX && cost[k][j] != INT\_MAX

&& cost[i][k] + cost[k][j] < cost[i][j])

{

cost[i][j] = cost[i][k] + cost[k][j];

path[i][j] = path[k][j];

}

}

}

}

// Print the least cost between all pairs of cities

printSolution(cost, path);

}

int main()

{

// given adjacency matrix representation of a graph

int graph[N][N] =

{

{ 0, 10, 14, 20 },

{ 20, 0, 40, INT\_MAX },

{ INT\_MAX, 35, 0, 8},

{ INT\_MAX, INT\_MAX, 13, 0 }

};

FloydWarshell(graph);

return 0;

}