МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информационные технологии»

Лабораторная работа №3

**«Разработка и исследование алгоритмов на деревьях на основе динамических структур данных»**

по дисциплине

Информационные технологии и программирование

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Халабия Р.Ф.

Москва, 2024 г

1. Цели и задачи

Изучить понятия, формирования, особенности доступа к

данным и работы с памятью в деревьях, разрабатывать программы с использованием алгоритмов рекурсии, алгоритмов обхода деревьев и алгоритмов нахождения кратчайшего пути.

1. Постановка задачи

2.1 Написать программу поиска кратчайшие расстояния от заданной вершины (организуется ввод) до всех остальных для графа согласно своего варианта (см. Рис. 1), используя алгоритмы алгоритмы Дейкстры и Флойда. Составьте таблицу сравнения двух алгоритмов по времени или количеству шагов. Реализация в оконном интерфейсе.

2.2 Написать программу, которая помогает Оле добраться до городов разными способами. Определить наименьшую сумму, которую нужно потратить, чтобы Оля могла навестить каждого из своих друзей и вернуться к себе домой. Предположим Оля живет в городе 1, а друзья живут в других городах (вершины 1,2,3 и тд.). Стоимость билетов из разных городов известна согласно графу варианта задания. Реализация в оконном интерфейсе.

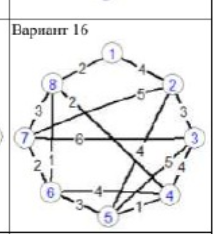


Рисунок 1 – условие индивидуального варианта

1. Листинг программы

#include <iostream>

#include <vector>

#include <set>

#include <algorithm>

**using** **namespace** std;

#define int long long

**const** **int** inf = **1001** \* **1001** \* **1001**;

vector<**int**> dijkstra(**const** vector<vector<pair<**int**, **int**>>>& g, **int** s) {

**int** n = g.size();

vector<**int**> d(n, inf);

d[s] = **0**;

set<pair<**int**, **int**> > q;

q.emplace(**0**, s);

**while** (!q.empty()) {

**int** v = q.begin()->second;

q.erase(q.begin());

**for** (**auto** [u, w] : g[v]) {

**if** (d[u] > d[v] + w) {

q.erase({d[u], u});

d[u] = d[v] + w;

q.emplace(d[u], u);

}

}

}

**return** d;

}

vector<vector<**int**>> to\_adjacency\_matrix(**const** vector<vector<pair<**int**, **int** >>>& g){

vector<vector<**int**>> ret;

ret.resize(g.size());

**for**(**int** i = **0**; i < g.size(); ++i){

ret[i].resize(g.size());

**for**(**int** j = **0**; j < g.size(); ++j){

ret[i][j] = inf;

}

}

**for**(**int** i = **0**; i < g.size(); ++i){

ret[i].resize(g.size());

**for**(**int** j = **0**; j < g[i].size(); ++j){

ret[i][g[i][j].first] = g[i][j].second;

}

}

**return** ret;

}

vector<vector<**int**>> floydWarshall(vector<vector<**int**>> g){

**for**(**int** i = **0**; i < g.size(); ++i){

g[i][i] = **0**;

}

**for**(**int** k=**0**;k<g.size();k++){

**for**(**int** i=**0**;i<g.size();i++){

**for**(**int** j=**0**;j<g.size();j++){

**if**(g[i][k] + g[k][j] < g[i][j]){

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

}

}

}

}

**return** g;

}

pair<**int**, vector<**int**> > TSP(**const** vector<vector<**int**>>& g, **int** start)

{

vector<**int**> vertex;

**for** (**int** i = **0**; i < g.size(); i++)

**if** (i != start)

vertex.push\_back(i);

**int** min\_path = inf;

vector<**int**> min\_path\_vec = {-**1**};

**do** {

**int** current\_pathweight = **0**;

**int** k = start;

**for** (**int** i = **0**; i < vertex.size(); i++) {

current\_pathweight += g[k][vertex[i]];

k = vertex[i];

}

current\_pathweight += g[k][start];

// update minimum

**if** (current\_pathweight < min\_path){

min\_path = current\_pathweight;

min\_path\_vec = vertex;

}

} **while** (

next\_permutation(vertex.begin(), vertex.end()));

**return** {min\_path, min\_path\_vec};

}

**void** task1(){

vector<vector<pair<**int**, **int**>>> g;

**int** n,m,start;

cin >> n >> m >> start;

--start;

g.resize(n);

**for**(**int** i = **0**; i < m; ++i){

**int** s,e,val;

cin >> s >> e >> val;

--s;

--e;

g[s].push\_back({e,val});

g[e].push\_back({s,val});

}

**auto** paths = dijkstra(g,start);

cout << endl<<"Djikstra:" << endl;

**for**(**auto** elem : paths){

cout << elem << " ";

}

cout << endl << endl << "FloydWarshall:" << endl;

**auto** g\_fw = floydWarshall(to\_adjacency\_matrix(g));

**for**(**auto** elem : g\_fw[start]){

cout << elem << " ";

}

}

**void** task2(){

vector<vector<pair<**int**, **int**>>> g;

**int** n,m,start;

cin >> n >> m >> start;

--start;

g.resize(n);

**for**(**int** i = **0**; i < m; ++i){

**int** s,e,val;

cin >> s >> e >> val;

--s;

--e;

g[s].push\_back({e,val});

g[e].push\_back({s,val});

}

**auto** [ans, vec] = TSP(to\_adjacency\_matrix(g),start);

cout << endl << ans << endl;

**for**(**auto** elem : vec){

cout << elem << " ";

}

}

**signed** main()

{

**char** c = '?';

**while** (**1**){

cout << "type 1 to enter task 1, type 2 to enter task 2: ";

cin >> c;

**if** (c == '1'){

task1();

**return** **0**;

}

**if** (c == '2'){

task2();

**return** **0**;

}

}

**return** **0**;

}

1. Схемы алгоритмов

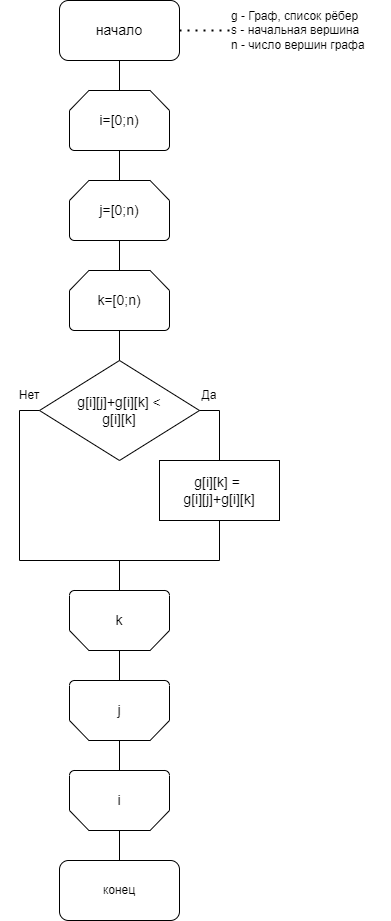


Рисунок 2 – схема алгоритма функции floydWarshall

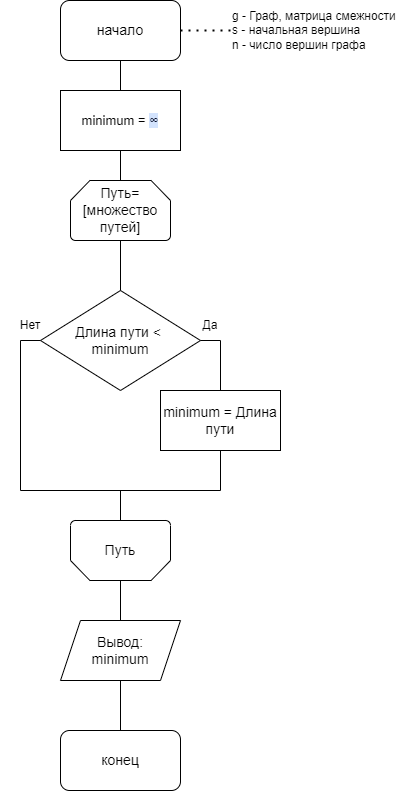


Рисунок 3 – схема алгоритма функции TSP

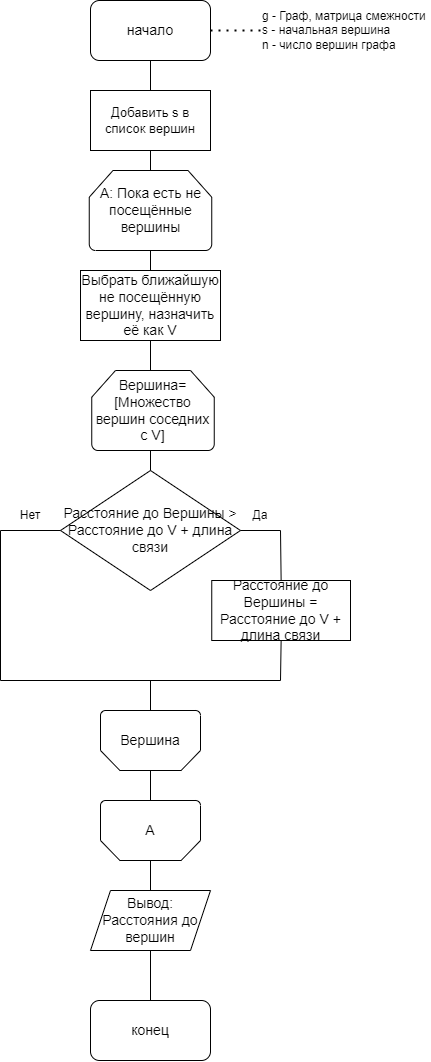


Рисунок 4 – схема алгоритма функции djikstra

1. Контрольный тест. Результаты программы

Результат запуска программы на контрольных данных можно увидеть на рисунках 5-6.

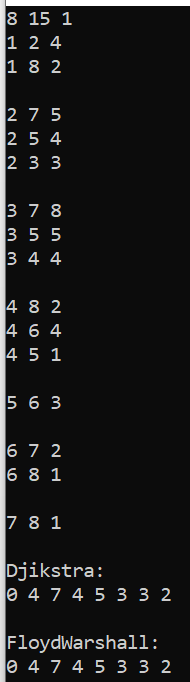


Рисунок 5 – результат запуска программы для первой части задания

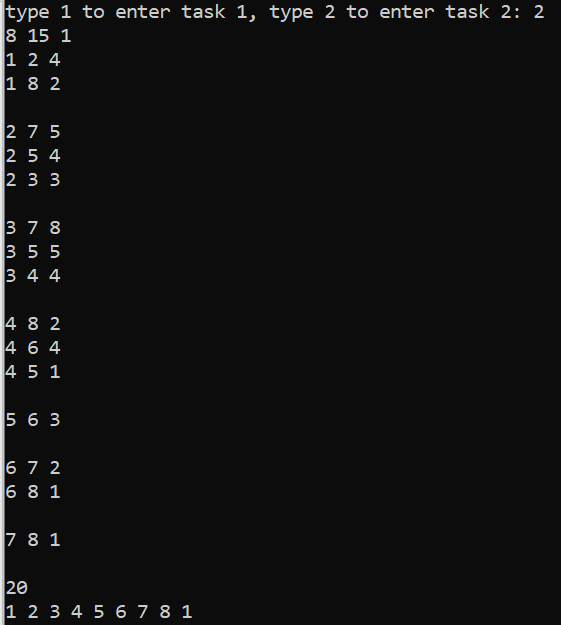


Рисунок 6 – результат запуска программы для второй части задания

1. Выводы по работе

В результате выполнения лабораторной работы мы смогли реализовать 3 алгоритма для работы над графами: алгоритм Дейкстры, алгоритм Флойда, алгоритм Коммивояжёра.