МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Структуры и алгоритмы обработки данных

Отчёт по лабораторной работе №2

Выполнил студент группы М3О-219Бк:

Газиева Л.Р.

Проверил:

Ведьманов И.С.

Москва 2021 г.

# Задание

1) Реализовать алгоритмы Дейкстры, Беллмана-Форда, Флойда-Уоршелла, Прима, Крускала тремя способами:

- с использованием матриц смежности,

- с использованием конденсированной матрицы инцидентности (КМИ) через массив структур,

- с использованием КМИ через вектор векторов.

2) Написать отдельную программу, генерирующую случайным образом граф (в файл).

Требования к генерир. Графам:

1. Не должно быть петель,
2. все веса – положительные целые числа,
3. отсутствие двойных рёбер,
4. каждая вершина должна обладать как минимум одним исходящим ребром и как минимум одним входящим ребром.

3) Сравнить время и память, необходимые для функционирования программы (для графов с кол-вом вершин: 32, 256, 2048 (степени двойки)). Результаты представить в виде таблицы.

# Текст программы, генерирующей граф

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

int main() {

unsigned int n, edge = 0;

setlocale(LC\_ALL, "rus");

ofstream f("Graph.txt");

if (!f) {

cout << endl << "Ошибка открытия файла для записи.";

}

cout << "Введите кол-во вершин в графе:" << endl;

cin >> n;

f << n << endl << n << endl;

srand(time(NULL));

int lim = 3;

vector<int> limit(n);

vector<vector<int>> M(n, vector<int>(n));

for (unsigned int i = 1; i <= n; i++) {

limit[i - 1] = i;

}

for (unsigned int i = 0; i < n; i++) { // для каждой вершины вершины

int gr = rand() % 3 + 1; // кол-во исходящих рёбер

edge += gr;

for (int j = 0; j < gr; j++) {

f << i + 1 << " ";

int v = 0, weight = rand() % 1024, k = 0, check = 0;

while (v == 0 || v == i + 1 || M[v - 1][i] != 0 || M[i][v - 1] != 0) {

unsigned int a = rand() % 2;

if ((a == 0 || (n - i - 1 <= limit.size())) && limit.size() > 0) {

k = rand() % limit.size();

v = limit[k];

check = 1;

}

else {

v = rand() % n + 1;

}

}

M[i][v - 1] = weight;

f << v << " " << weight << endl;

if (check == 1) {

limit[k] = limit[limit.size() - 1];

limit.pop\_back();

}

else {

for (unsigned int l = 0; l < limit.size(); l++) {

if (limit[l] == v) {

limit[l] = limit[limit.size() - 1];

limit.pop\_back();

}

}

}

}

}

f.close();

fstream of("Graph.txt");

of.seekg(0, ios\_base::beg);

of << edge;

of.close();

return 0;

}

# Текст основной программы

#include <iostream>

#include <vector>

#include <fstream>

#include <windows.h>

using namespace std;

int flag = 1;

vector<vector<pair<int, int>>> graph;

int\*\* M;

struct stR {

int num1;

int num2;

int weight;

};

stR\* mass;

vector<int> shortest, pred;

void Dijkstra(int s, int num\_vert); // вектор векторов

void Bellman\_Ford(int s, int num\_vert);

void Floyd\_Warshall(int s, int num\_vert);

int Relax(int u, int v);

void Prima(int s, int num\_vert);

void Kruskala(int num\_vert);

void M\_Dijkstra(int s, int num\_vert); // матрица смежности

void M\_Bellman\_Ford(int s, int num\_vert);

void M\_Floyd\_Warshall(int s, int num\_vert);

int M\_Relax(int u, int v);

void M\_Prima(int s, int num\_vert);

void M\_Kruskala(int num\_vert);

void str\_Dijkstra(int s, int num\_vert, int e); // массив структур

void str\_Bellman\_Ford(int s, int num\_vert, int e);

void str\_Floyd\_Warshall(int s, int num\_vert, int e);

int str\_Relax(int u, int v, int e);

void str\_Prima(int s, int num\_vert, int e);

void str\_Kruskala(int num\_vert, int e);

int main() {

int n, e;

setlocale(LC\_ALL, "rus");

ifstream file("C:\\Users\\Lida\\source\\repos\\Struct\_laba\_2\_graph\\Struct\_laba\_2\_graph\\Graph.txt");

if (!file) {

cout << endl << "Ошибка открытия файла для чтения." << endl;

}

file >> e >> n; // кол-во рёбер и вершин в графе

shortest.resize(n);

pred.resize(n);

M = new int\* [n];

for (int i = 0; i < n; i++) {

M[i] = new int[n];

for (int j = 0; j < n; j++) {

M[i][j] = 0;

}

}

mass = new stR[e];

vector<pair<int, int>> v;

pair<int, int> t;

int num\_v, g = 0;

file >> num\_v;

int lss = num\_v;

while (!file.eof()) {

while ((num\_v == lss) && !file.eof()) {

file >> t.first >> t.second;

v.emplace\_back(t);

M[num\_v - 1][t.first - 1] = t.second;

mass[g].num1 = num\_v;

mass[g].num2 = t.first;

mass[g].weight = t.second;

file >> num\_v;

g++;

}

graph.emplace\_back(v);

v.clear();

lss = num\_v;

}

int s = 1;

file.close();

DWORD t1, t2;

int size\_gr = 0;

for (int i = 0; i < n; i++) {

size\_gr += sizeof(pair<int, int>) \* graph[i].size();

}

cout << "------------------------------------------------" << endl << "Количество вершин в графе: " << n << endl;

cout << "------------------------------------------------" << endl;

cout << "Способ хранения графа | Матрица смежности | КМИ через массив структур | КМИ через вектор векторов";

cout << endl << "Объём памяти\t\t|\t" << sizeof(int) \* n \* n << "\t\t|\t" << sizeof(stR) \* e << "\t\t|\t" << size\_gr;

if (flag == 0) {

cout << endl << "------------------------------------------------";

cout << endl << endl << "АЛГОРИТМ ДЕЙКСТРЫ" << endl;

}

t1 = GetTickCount();

M\_Dijkstra(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << "Время выполнения|\t" << (t2 - t1) << " | ";

}

t1 = GetTickCount();

str\_Dijkstra(s, n, e);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1 << " | ";

}

t1 = GetTickCount();

Dijkstra(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1;

cout << endl << "------------------------------------------------";

cout << endl << endl << "АЛГОРИТМ БЕЛЛМАНА-ФОРДА" << endl;

}

t1 = GetTickCount();

M\_Bellman\_Ford(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << "Время выполнения|\t" << (t2 - t1) << " | ";

}

t1 = GetTickCount();

str\_Bellman\_Ford(s, n, e);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1 << " | ";

}

t1 = GetTickCount();

Bellman\_Ford(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1;

cout << endl << "------------------------------------------------";

cout << endl << endl << "АЛГОРИТМ ФЛОЙДА-УОРШЕЛЛА" << endl;

}

t1 = GetTickCount();

M\_Floyd\_Warshall(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << "Время выполнения|\t" << (t2 - t1) << " | ";

}

t1 = GetTickCount();

str\_Floyd\_Warshall(s, n, e);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1 << " | ";

}

t1 = GetTickCount();

Floyd\_Warshall(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1;

cout << endl << "------------------------------------------------";

cout << endl << endl << "АЛГОРИТМ ПРИМА" << endl;

}

t1 = GetTickCount();

M\_Prima(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << "Время выполнения|\t" << (t2 - t1) << " | ";

}

t1 = GetTickCount();

str\_Prima(s, n, e);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1 << " | ";

}

t1 = GetTickCount();

Prima(s, n);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1;

cout << endl << "------------------------------------------------";

cout << endl << endl << "АЛГОРИТМ КРУСКАЛА" << endl;

}

t1 = GetTickCount();

M\_Kruskala(n);

t2 = GetTickCount();

if (flag == 0) {

cout << "Время выполнения|\t" << (t2 - t1) << " | ";

}

t1 = GetTickCount();

str\_Kruskala(n, e);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1 << " | ";

}

t1 = GetTickCount();

Kruskala(n);

t2 = GetTickCount();

if (flag == 0) {

cout << t2 - t1;

}

for (int i = 0; i < n; i++) {

delete[] M[i];

}

delete[] mass;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* АЛГОРИТМЫ С МАТРИЦЕЙ СМЕЖНОСТИ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void M\_Dijkstra(int s, int num\_vert) {

for (int v = 0; v < num\_vert; v++) {

shortest[v] = 100000; // бесконечность

pred[v] = NULL;

}

shortest[s - 1] = 0;

vector<int> Q(num\_vert);

for (int i = 0; i < num\_vert; i++) {

Q[i] = i + 1;

}

while (Q.size() > 0) {

int u = Q[0], ii = 0;

for (unsigned int i = 0; i < Q.size(); i++) {

if (shortest[Q[i] - 1] < shortest[u - 1]) {

u = Q[i];

ii = i;

}

}

Q[ii] = Q[Q.size() - 1];

Q.pop\_back();

for (int j = 0; j < num\_vert; j++) {

if (M[u - 1][j] != 0) {

int v = j + 1; // номер смежной с u вершины

M\_Relax(u, v);

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ ДЕЙКСТРЫ" << endl;

cout << endl << "shortest: ";

for (int i = 0; i < num\_vert; i++) {

cout << shortest[i] << " ";

}

cout << endl << "pred: ";

for (int i = 0; i < num\_vert; i++) {

cout << pred[i] << " ";

}

}

}

void M\_Bellman\_Ford(int s, int num\_vert) {

for (int v = 0; v < num\_vert; v++) {

shortest[v] = 100000; // бесконечность

pred[v] = NULL;

}

shortest[s - 1] = 0; // если пути к v нет, то shortest[v] = беск., pred[v] = NULL

for (int i = 0; i < num\_vert - 1; i++) {

int check = 0;

for (int u = 1; u <= num\_vert; u++) {

for (int j = 0; j < num\_vert; j++) {

if (M[u - 1][j] != 0) {

int check1 = 0, v = j + 1;

check1 += M\_Relax(u, v);

check += check1;

}

}

}

if (check == 0) {

break;

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ БЕЛЛМАНА-ФОРДА" << endl;

cout << endl << "shortest: ";

for (int i = 0; i < num\_vert; i++) {

cout << shortest[i] << " ";

}

cout << endl << "pred: ";

for (int i = 0; i < num\_vert; i++) {

cout << pred[i] << " ";

}

}

}

int M\_Relax(int u, int v) {

int b = M[u - 1][v - 1]; // путь от u до v

if (shortest[u - 1] + b < shortest[v - 1]) {

shortest[v - 1] = shortest[u - 1] + b;

pred[v - 1] = u;

return 1;

}

return 0;

}

void M\_Floyd\_Warshall(int s, int num\_vert) {

int\*\* shortest = new int\* [num\_vert], \*\* pred = new int\* [num\_vert];

for (int i = 0; i < num\_vert; i++) {

shortest[i] = new int[num\_vert];

pred[i] = new int[num\_vert];

}

for (int u = 0; u < num\_vert; u++) {

for (int k = 0; k < num\_vert; k++) {

if (M[u][k] != 0) {

shortest[u][k] = M[u][k];

pred[u][k] = u + 1;

}

else {

shortest[u][k] = 100000;

pred[u][k] = NULL;

}

}

}

for (int k = 0; k < num\_vert; k++) {

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

if (shortest[i][k] + shortest[k][j] < shortest[i][j]) {

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

pred[i][j] = k + 1;

}

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТ ФЛОЙДА\_УОРШЕЛЛА" << endl;

cout << endl << "shortest:" << endl;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

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

}

cout << endl;

}

cout << endl << "pred:" << endl;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

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

}

cout << endl;

}

}

for (int i = 0; i < num\_vert; i++) {

delete[] shortest[i];

delete[] pred[i];

}

delete[] shortest;

delete[] pred;

}

void M\_Prima(int s, int num\_vert) {

vector<int> U(1, s), V, predv;

int weight = 0;

for (int i = 1; i <= num\_vert; i++) {

if (i != s) {

V.push\_back(i);

}

}

for (int i = 0; i < num\_vert - 1; i++) {

int u, w, min\_w = 100000;

for (unsigned int j = 0; j < U.size(); j++) {

int u1 = U[j];

for (unsigned int k = 0; k < V.size(); k++) {

int v1 = V[k];

if ((M[u1 - 1][v1 - 1] != 0) && (M[u1 - 1][v1 - 1] < min\_w)) {

min\_w = M[u1 - 1][v1 - 1];

u = u1;

w = v1;

}

else if ((M[v1 - 1][u1 - 1] != 0) && (M[v1 - 1][u1 - 1] < min\_w)) {

min\_w = M[v1 - 1][u1 - 1];

u = u1;

w = v1;

}

}

}

U.push\_back(w);

predv.push\_back(u);

for (unsigned int i = 0; i < V.size(); i++) {

if (V[i] == w) {

V[i] = V[V.size() - 1];

V.pop\_back();

break;

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ ПРИМА" << endl;

cout << endl << "Рёбра ост. дерева: ";

for (unsigned int i = 0; i < predv.size(); i++) {

cout << predv[i] << "-" << U[i + 1] << " ";

}

cout << endl << endl;

}

}

void M\_Kruskala(int num\_vert) {

vector<int> edge, vert1, vert2;

vector<pair<int, int>> uv;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

if (M[i][j] != 0) {

vert1.push\_back(i + 1);

vert2.push\_back(j + 1);

edge.push\_back(M[i][j]);

}

}

}

for (int i = edge.size() / 2; i >= 1; i /= 2) {

for (int j = i; j < edge.size(); j++) {

for (int k = j; k - i >= 0; k -= i) {

if (edge[k] < edge[k - i]) {

int m = edge[k];

edge[k] = edge[k - i];

edge[k - i] = m;

m = vert1[k];

vert1[k] = vert1[k - i];

vert1[k - i] = m;

m = vert2[k];

vert2[k] = vert2[k - i];

vert2[k - i] = m;

}

}

}

}

vector<int> tree(num\_vert);

for (int i = 0; i < num\_vert; i++) {

tree[i] = i + 1;

}

for (unsigned int i = 0; i < edge.size(); i++) {

int u = vert1[i], v = vert2[i];

if (tree[u - 1] != tree[v - 1]) {

uv.push\_back(make\_pair(u, v));

int h = tree[v - 1];

for (unsigned int j = 0; j < num\_vert; j++) {

if (tree[j] == h) {

tree[j] = tree[u - 1];

}

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ КРУСКАЛА" << endl;

cout << endl << "Рёбра ост. дерева: ";

for (unsigned int i = 0; i < uv.size(); i++) {

cout << uv[i].first << "-" << uv[i].second << " ";

}

cout << endl << endl;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* АЛГОРИТМЫ С МАССИВОМ СТРУКТУР \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void str\_Dijkstra(int s, int num\_vert, int e) {

for (int v = 0; v < num\_vert; v++) {

shortest[v] = 100000; // бесконечность

pred[v] = NULL;

}

shortest[s - 1] = 0;

vector<int> Q(num\_vert);

for (int i = 0; i < num\_vert; i++) {

Q[i] = i + 1;

}

while (Q.size() > 0) {

int u = Q[0], ii = 0;

for (unsigned int i = 0; i < Q.size(); i++) {

if (shortest[Q[i] - 1] < shortest[u - 1]) {

u = Q[i];

ii = i;

}

}

Q[ii] = Q[Q.size() - 1];

Q.pop\_back();

for (unsigned int j = 0; j < e; j++) {

int v;

if (mass[j].num1 == u) {

v = mass[j].num2;

str\_Relax(u, v, e);

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ ДЕЙКСТРЫ" << endl;

cout << endl << "shortest: ";

for (int i = 0; i < num\_vert; i++) {

cout << shortest[i] << " ";

}

cout << endl << "pred: ";

for (int i = 0; i < num\_vert; i++) {

cout << pred[i] << " ";

}

}

}

void str\_Bellman\_Ford(int s, int num\_vert, int e) {

for (int v = 0; v < num\_vert; v++) {

shortest[v] = 100000; // бесконечность

pred[v] = NULL;

}

shortest[s - 1] = 0; // если пути к v нет, то shortest[v] = беск., pred[v] = NULL

for (int i = 0; i < num\_vert - 1; i++) {

int check = 0;

for (int u = 1; u <= num\_vert; u++) {

for (unsigned int j = 0; j < e; j++) {

int v, check1 = 0;

if (mass[j].num1 == u) {

v = mass[j].num2;

check1 += str\_Relax(u, v, e);

}

check += check1;

}

}

if (check == 0) {

break;

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ БЕЛЛМАНА-ФОРДА" << endl;

cout << endl << "shortest: ";

for (int i = 0; i < num\_vert; i++) {

cout << shortest[i] << " ";

}

cout << endl << "pred: ";

for (int i = 0; i < num\_vert; i++) {

cout << pred[i] << " ";

}

}

}

int str\_Relax(int u, int v, int e) {

int b = 0; // путь от u до v

for (unsigned int j = 0; j < e; j++) {

if (mass[j].num1 == u && mass[j].num2 == v) {

b = mass[j].weight;

break;

}

}

if (shortest[u - 1] + b < shortest[v - 1]) {

shortest[v - 1] = shortest[u - 1] + b;

pred[v - 1] = u;

return 1;

}

return 0;

}

void str\_Floyd\_Warshall(int s, int num\_vert, int e) {

int\*\* shortest = new int \*[num\_vert], \*\* pred = new int\* [num\_vert];

for (int i = 0; i < num\_vert; i++) {

shortest[i] = new int [num\_vert];

pred[i] = new int [num\_vert];

}

for (int u = 0; u < num\_vert; u++) {

for (int k = 0; k < num\_vert; k++) {

shortest[u][k] = 100000;

pred[u][k] = NULL;

}

}

for (int j = 0; j < e; j++) {

int u = mass[j].num1, k = mass[j].num2;

shortest[u - 1][k - 1] = mass[j].weight;

pred[u - 1][k - 1] = u;

}

for (int k = 0; k < num\_vert; k++) {

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

if (shortest[i][k] + shortest[k][j] < shortest[i][j]) {

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

pred[i][j] = k + 1;

}

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТ ФЛОЙДА\_УОРШЕЛЛА" << endl;

cout << endl << "shortest:" << endl;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

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

}

cout << endl;

}

cout << endl << "pred:" << endl;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

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

}

cout << endl;

}

}

for (int i = 0; i < num\_vert; i++) {

delete[] shortest[i];

delete[] pred[i];

}

delete[] shortest;

delete[] pred;

}

void str\_Prima(int s, int num\_vert, int e) {

vector<int> U(1, s), V, predv;

int weight = 0;

for (int i = 1; i <= num\_vert; i++) {

if (i != s) {

V.push\_back(i);

}

}

for (int i = 0; i < num\_vert - 1; i++) {

int u, w, min\_w = 100000;

for (unsigned int j = 0; j < U.size(); j++) {

int u1 = U[j];

for (unsigned int k = 0; k < V.size(); k++) {

int v1 = V[k];

for (int m = 0; m < e; m++) {

if (((mass[m].num1 == u1 && mass[m].num2 == v1) || (mass[m].num1 == v1 && mass[m].num2 == u1)) && mass[m].weight < min\_w) {

min\_w = mass[m].weight;

u = u1;

w = v1;

}

}

}

}

U.push\_back(w);

predv.push\_back(u);

for (unsigned int i = 0; i < V.size(); i++) {

if (V[i] == w) {

V[i] = V[V.size() - 1];

V.pop\_back();

break;

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ ПРИМА" << endl;

cout << endl << "Рёбра ост. дерева: ";

for (unsigned int i = 0; i < predv.size(); i++) {

cout << predv[i] << "-" << U[i + 1] << " ";

}

cout << endl << endl;

}

}

void str\_Kruskala(int num\_vert, int e) {

vector<int> edge, vert1, vert2;

vector<pair<int, int>> uv;

for (int i = 0; i < e; i++) {

vert1.push\_back(mass[i].num1);

vert2.push\_back(mass[i].num2);

edge.push\_back(mass[i].weight);

}

for (int i = edge.size() / 2; i >= 1; i /= 2) {

for (int j = i; j < edge.size(); j++) {

for (int k = j; k - i >= 0; k -= i) {

if (edge[k] < edge[k - i]) {

int m = edge[k];

edge[k] = edge[k - i];

edge[k - i] = m;

m = vert1[k];

vert1[k] = vert1[k - i];

vert1[k - i] = m;

m = vert2[k];

vert2[k] = vert2[k - i];

vert2[k - i] = m;

}

}

}

}

vector<int> tree(num\_vert);

for (int i = 0; i < num\_vert; i++) {

tree[i] = i + 1;

}

for (unsigned int i = 0; i < edge.size(); i++) {

int u = vert1[i], v = vert2[i];

if (tree[u - 1] != tree[v - 1]) {

uv.push\_back(make\_pair(u, v));

int h = tree[v - 1];

for (unsigned int j = 0; j < num\_vert; j++) {

if (tree[j] == h) {

tree[j] = tree[u - 1];

}

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ КРУСКАЛА" << endl;

cout << endl << "Рёбра ост. дерева: ";

for (unsigned int i = 0; i < uv.size(); i++) {

cout << uv[i].first << "-" << uv[i].second << " ";

}

cout << endl << endl;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* АЛГОРИТМЫ С ВЕКТОРАМИ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Dijkstra(int s, int num\_vert) {

for (int v = 0; v < num\_vert; v++) {

shortest[v] = 100000; // бесконечность

pred[v] = NULL;

}

shortest[s - 1] = 0;

vector<int> Q(num\_vert);

for (int i = 0; i < num\_vert; i++) {

Q[i] = i + 1;

}

while (Q.size() > 0) {

int u = Q[0], ii = 0;

for (unsigned int i = 0; i < Q.size(); i++) {

if (shortest[Q[i] - 1] < shortest[u - 1]) {

u = Q[i];

ii = i;

}

}

Q[ii] = Q[Q.size() - 1];

Q.pop\_back();

for (unsigned int j = 0; j < graph[u - 1].size(); j++) {

int v = graph[u - 1][j].first; // номер смежной с u вершины

Relax(u, v);

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ ДЕЙКСТРЫ" << endl;

cout << endl << "shortest: ";

for (int i = 0; i < num\_vert; i++) {

cout << shortest[i] << " ";

}

cout << endl << "pred: ";

for (int i = 0; i < num\_vert; i++) {

cout << pred[i] << " ";

}

}

}

void Bellman\_Ford(int s, int num\_vert) {

for (int v = 0; v < num\_vert; v++) {

shortest[v] = 100000; // бесконечность

pred[v] = NULL;

}

shortest[s - 1] = 0; // если пути к v нет, то shortest[v] = беск., pred[v] = NULL

for (int i = 0; i < num\_vert - 1; i++) {

int check = 0;

for (int u = 1; u <= num\_vert; u++) {

for (unsigned int j = 0; j < graph[u - 1].size(); j++) {

int check1 = 0;

int v = graph[u - 1][j].first; // номер смежной с u вершины

check1 += Relax(u, v);

check += check1;

}

}

if (check == 0) {

break;

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ БЕЛЛМАНА-ФОРДА" << endl;

cout << endl << "shortest: ";

for (int i = 0; i < num\_vert; i++) {

cout << shortest[i] << " ";

}

cout << endl << "pred: ";

for (int i = 0; i < num\_vert; i++) {

cout << pred[i] << " ";

}

}

}

int Relax(int u, int v) {

int b = 0; // путь от u до v

for (int j = 0; j < graph[u - 1].size(); j++) {

if (graph[u - 1][j].first == v) {

b = graph[u - 1][j].second;

}

}

if (shortest[u - 1] + b < shortest[v - 1]) {

shortest[v - 1] = shortest[u - 1] + b;

pred[v - 1] = u;

return 1;

}

return 0;

}

void Floyd\_Warshall(int s, int num\_vert) {

int\*\* shortest = new int \*[num\_vert], \*\* pred = new int\* [num\_vert];

for (int i = 0; i < num\_vert; i++) {

shortest[i] = new int [num\_vert];

pred[i] = new int [num\_vert];

}

for (int u = 0; u < num\_vert; u++) {

for (int k = 0; k < num\_vert; k++) {

shortest[u][k] = 100000;

pred[u][k] = NULL;

}

}

for (int u = 0; u < num\_vert; u++) {

for (unsigned int j = 0; j < graph[u].size(); j++) {

int v = graph[u][j].first; // номер смежной с u вершины

shortest[u][v - 1] = graph[u][j].second;

pred[u][v - 1] = u + 1;

}

}

for (int k = 0; k < num\_vert; k++) {

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

if (shortest[i][k] + shortest[k][j] < shortest[i][j]) {

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

pred[i][j] = k + 1;

}

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТ ФЛОЙДА\_УОРШЕЛЛА" << endl;

cout << endl << "shortest:" << endl;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

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

}

cout << endl;

}

cout << endl << "pred:" << endl;

for (int i = 0; i < num\_vert; i++) {

for (int j = 0; j < num\_vert; j++) {

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

}

cout << endl;

}

}

for (int i = 0; i < num\_vert; i++) {

delete[] shortest[i];

delete[] pred[i];

}

delete[] shortest;

delete[] pred;

}

void Prima(int s, int num\_vert) {

vector<int> U(1, s), V, predv;

int weight = 0;

for (int i = 1; i <= num\_vert; i++) {

if (i != s) {

V.push\_back(i);

}

}

for (int i = 0; i < num\_vert - 1; i++) {

int u, w, min\_w = 100000;

for (unsigned int j = 0; j < U.size(); j++) {

int u1 = U[j];

for (unsigned int k = 0; k < V.size(); k++) {

int v1 = V[k];

for (unsigned int m = 0; m < graph[u1 - 1].size(); m++) {

if ((v1 == graph[u1 - 1][m].first) && (graph[u1 - 1][m].second < min\_w)) {

min\_w = graph[u1 - 1][m].second;

u = u1;

w = v1;

}

}

for (int m = 0; m < graph[v1 - 1].size(); m++) {

if ((u1 == graph[v1 - 1][m].first) && (graph[v1 - 1][m].second < min\_w)) {

min\_w = graph[v1 - 1][m].second;

u = u1;

w = v1;

}

}

}

}

U.push\_back(w);

predv.push\_back(u);

for (unsigned int i = 0; i < V.size(); i++) {

if (V[i] == w) {

V[i] = V[V.size() - 1];

V.pop\_back();

break;

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ ПРИМА" << endl;

cout << endl << "Рёбра ост. дерева: ";

for (unsigned int i = 0; i < predv.size(); i++) {

cout << predv[i] << "-" << U[i + 1] << " ";

}

cout << endl << endl;

}

}

void Kruskala(int num\_vert) {

vector<int> edge, vert1, vert2;

vector<pair<int, int>> uv;

for (int i = 0; i < num\_vert; i++) {

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

vert1.push\_back(i + 1);

vert2.push\_back(graph[i][j].first);

edge.push\_back(graph[i][j].second);

}

}

for (int i = edge.size() / 2; i >= 1; i /= 2) {

for (int j = i; j < edge.size(); j++) {

for (int k = j; k - i >= 0; k -= i) {

if (edge[k] < edge[k - i]) {

int m = edge[k];

edge[k] = edge[k - i];

edge[k - i] = m;

m = vert1[k];

vert1[k] = vert1[k - i];

vert1[k - i] = m;

m = vert2[k];

vert2[k] = vert2[k - i];

vert2[k - i] = m;

}

}

}

}

vector<int> tree(num\_vert);

for (int i = 0; i < num\_vert; i++) {

tree[i] = i + 1;

}

for (unsigned int i = 0; i < edge.size(); i++) {

int u = vert1[i], v = vert2[i];

if (tree[u - 1] != tree[v - 1]) {

uv.push\_back(make\_pair(u, v));

int h = tree[v - 1];

for (unsigned int j = 0; j < num\_vert; j++) {

if (tree[j] == h) {

tree[j] = tree[u - 1];

}

}

}

}

if (flag == 1) {

cout << endl << "АЛГОРИТМ КРУСКАЛА" << endl;

cout << endl << "Рёбра ост. дерева: ";

for (unsigned int i = 0; i < uv.size(); i++) {

cout << uv[i].first << "-" << uv[i].second << " ";

}

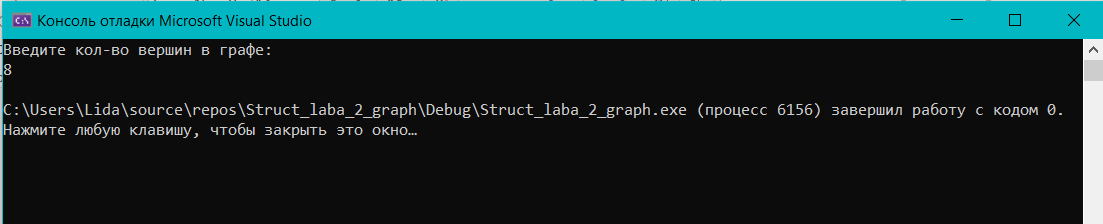
cout << endl << endl;

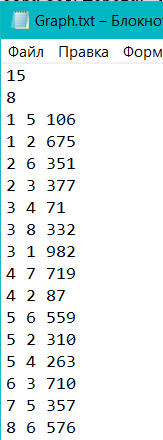
}

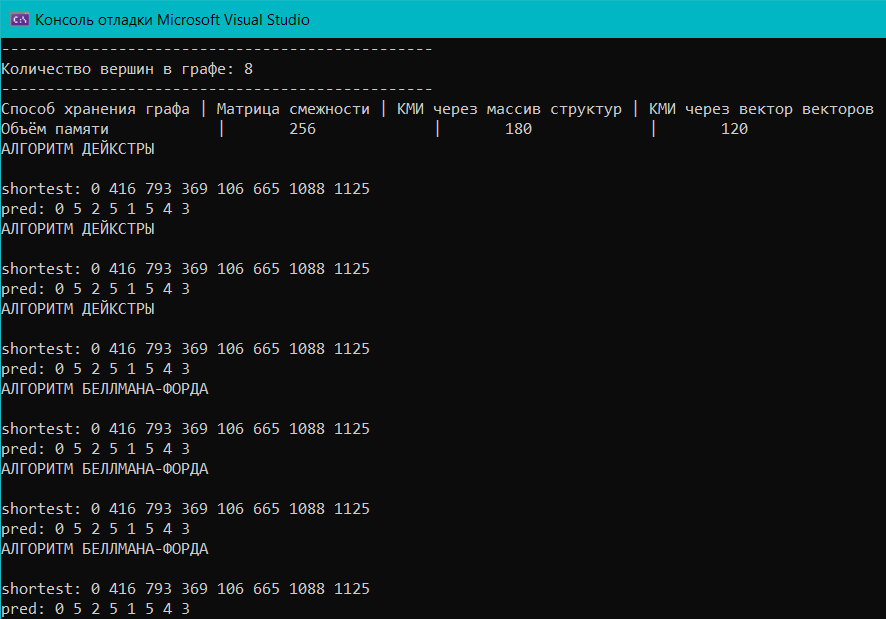
}

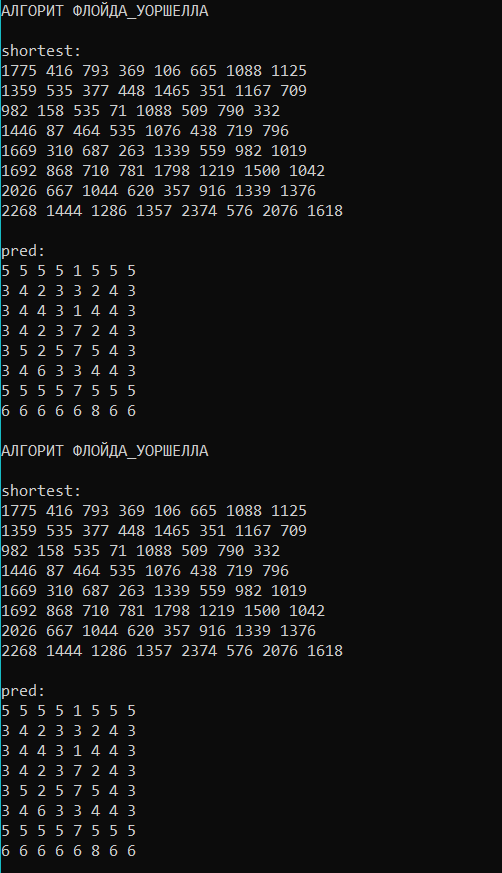
# Результаты

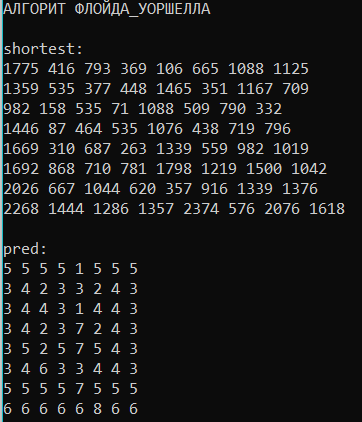
Кол-во вершин: 8

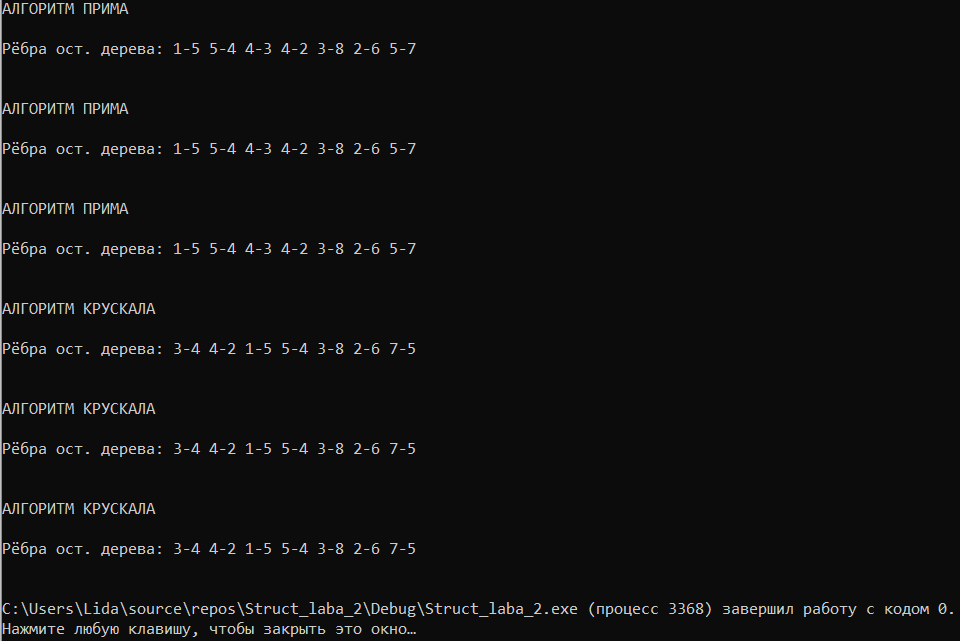




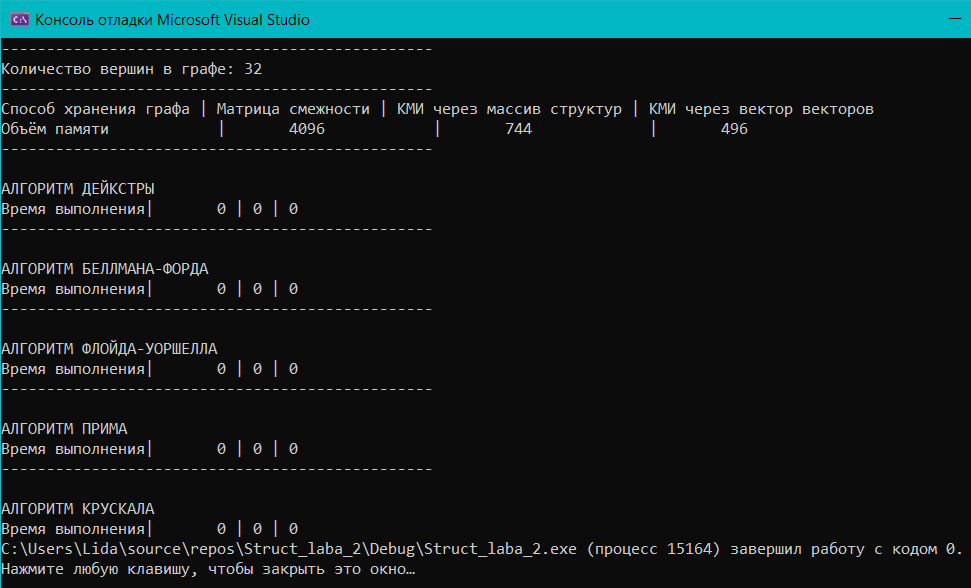




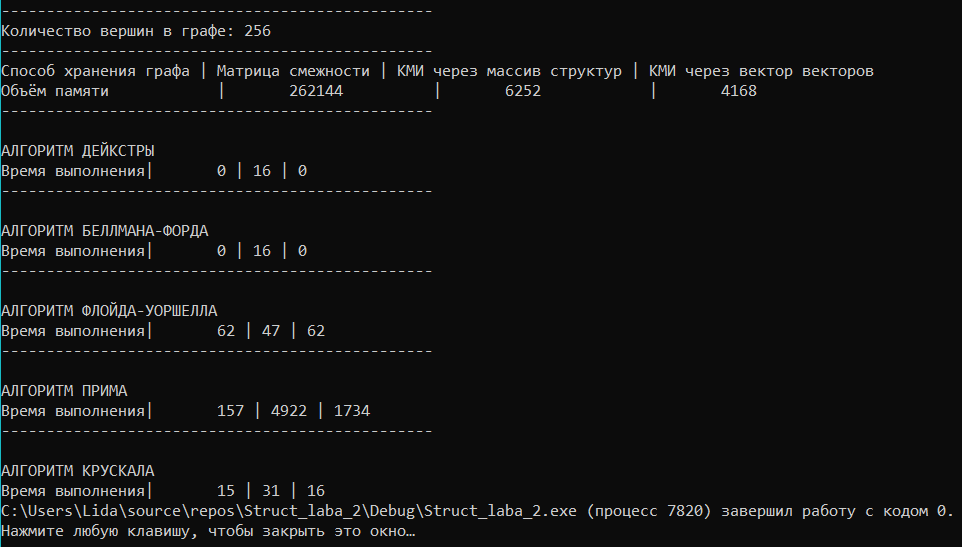




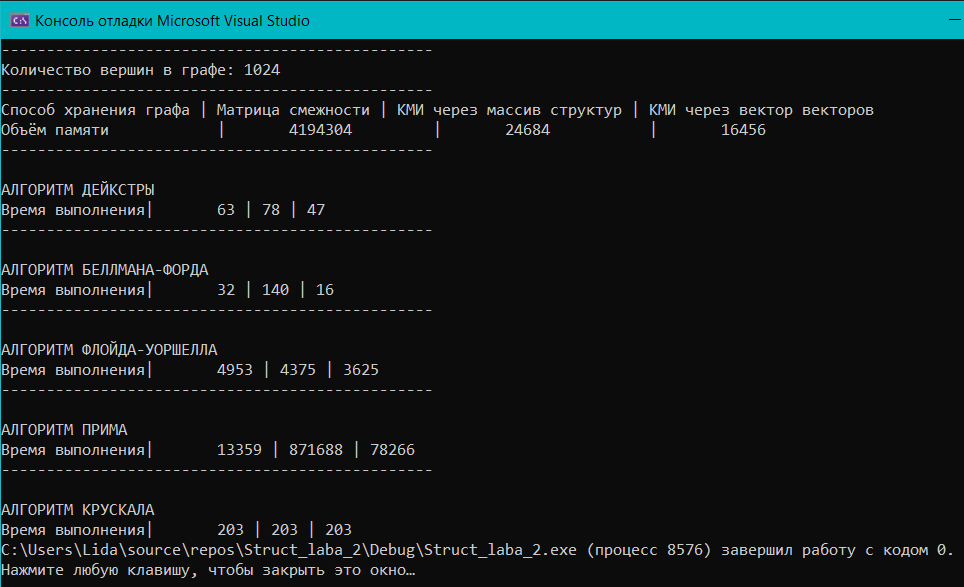
Кол-во вершин: 32



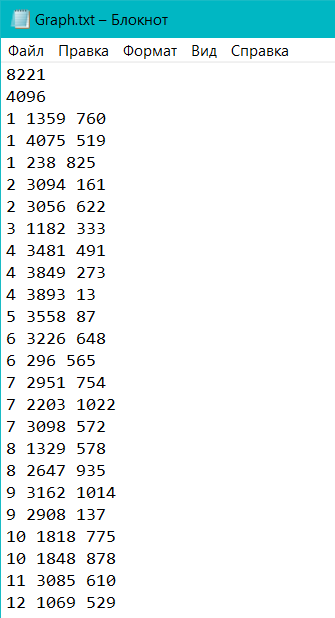
Кол-во вершин: 256

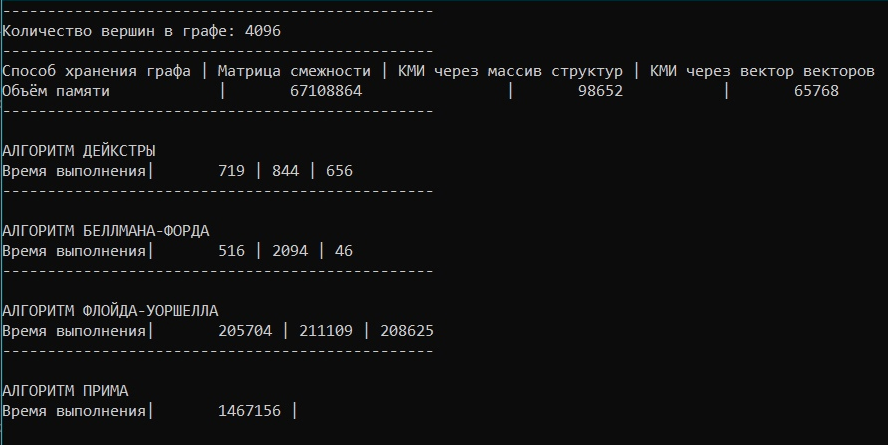


Кол-во вершин: 1024



Кол-во вершин: 4096





(Завершения алгоритма Прима я не дождалась)

|  |  |  |  |
| --- | --- | --- | --- |
|  | АЛГОРИТМ ДЕЙКСТРЫ | | |
| Количество вершин | Матрица смежности | КМИ через массив структур | КМИ через вектор векторов |
| 32 | Объём памяти |  |  |
| 4096 байт | 744 байта | 496 байт |
| Время выполнения |  |  |
| < 1мс | < 1мс | < 1мс |
| 256 | Объём памяти |  |  |
| 262144 байт | 6252 байта | 4168 байт |
| Время выполнения |  |  |
| < 1мс | 16мс | < 1мс |
| 1024 | Объём памяти |  |  |
| 4194304 байт | 24684 байта | 16456 байт |
| Время выполнения |  |  |
| 63мс | 78мс | 47мс |
| 4096 | Объём памяти |  |  |
| 67108864 байт | 98652 байта | 65768 байт |
| Время выполнения |  |  |
| 719мс | 844мс | 656мс |

|  |  |  |  |
| --- | --- | --- | --- |
|  | АЛГОРИТМ БЕЛЛМАНА-ФОРДА | | |
| Количество вершин | Матрица смежности | КМИ через массив структур | КМИ через вектор векторов |
| 32 | Объём памяти |  |  |
| 4096 байт | 744 байта | 496 байт |
| Время выполнения |  |  |
| < 1мс | < 1мс | < 1мс |
| 256 | Объём памяти |  |  |
| 262144 байт | 6252 байта | 4168 байт |
| Время выполнения |  |  |
| < 1мс | 16мс | < 1мс |
| 1024 | Объём памяти |  |  |
| 4194304 байт | 24684 байта | 16456 байт |
| Время выполнения |  |  |
| 32мс | 140мс | 16мс |
| 4096 | Объём памяти |  |  |
| 67108864 байт | 98652 байта | 65768 байт |
| Время выполнения |  |  |
| 516мс | 2094мс | 46мс |

|  |  |  |  |
| --- | --- | --- | --- |
|  | АЛГОРИТМ ФЛОЙДА-УОРШЕЛЛА | | |
| Количество вершин | Матрица смежности | КМИ через массив структур | КМИ через вектор векторов |
| 32 | Объём памяти |  |  |
| 4096 байт | 744 байта | 496 байт |
| Время выполнения |  |  |
| < 1мс | < 1мс | < 1мс |
| 256 | Объём памяти |  |  |
| 262144 байт | 6252 байта | 4168 байт |
| Время выполнения |  |  |
| 62мс | 47мс | 62мс |
| 1024 | Объём памяти |  |  |
| 4194304 байт | 24684 байта | 16456 байт |
| Время выполнения |  |  |
| 4953мс | 4375мс | 3625мс |
| 4096 | Объём памяти |  |  |
| 67108864 байт | 98652 байта | 65768 байт |
| Время выполнения |  |  |
| 205704мс | 211109мс | 208625мс |

|  |  |  |  |
| --- | --- | --- | --- |
|  | АЛГОРИТМ ПРИМА | | |
| Количество вершин | Матрица смежности | КМИ через массив структур | КМИ через вектор векторов |
| 32 | Объём памяти |  |  |
| 4096 байт | 744 байта | 496 байт |
| Время выполнения |  |  |
| < 1мс | < 1мс | < 1мс |
| 256 | Объём памяти |  |  |
| 262144 байт | 6252 байта | 4168 байт |
| Время выполнения |  |  |
| 157мс | 4922мс | 1734мс |
| 1024 | Объём памяти |  |  |
| 4194304 байт | 24684 байта | 16456 байт |
| Время выполнения |  |  |
| 13359мс | 871688мс | 78266мс |

|  |  |  |  |
| --- | --- | --- | --- |
|  | АЛГОРИТМ КРУСКАЛА | | |
| Количество вершин | Матрица смежности | КМИ через массив структур | КМИ через вектор векторов |
| 32 | Объём памяти |  |  |
| 4096 байт | 744 байта | 496 байт |
| Время выполнения |  |  |
| < 1мс | < 1мс | < 1мс |
| 256 | Объём памяти |  |  |
| 262144 байт | 6252 байта | 4168 байт |
| Время выполнения |  |  |
| 15мс | 31мс | 16мс |
| 1024 | Объём памяти |  |  |
| 4194304 байт | 24684 байта | 16456 байт |
| Время выполнения |  |  |
| 203мс | 203мс | 203мс |
| 4096 | Объём памяти |  |  |
| 67108864 байт | 98652 байта | 65768 байт |
| Время выполнения |  |  |
| 3188мс | 3156мс | 3656мс |

# Вывод

Из приведённых выше таблиц видно, что по затратам памяти выгоднее всего представлять граф 3-ьим способом (КМИ через вектор векторов).

Из трёх алгоритмов поиска кратчайшего пути быстрее всего оказался алгоритм Беллмана-Форда с графом представленным 3-ьим способом. Также оказалось, что для графов, генерируемых в данной работе (где число рёбер примерно в 2 раза больше числа вершин), алгоритм Крускала для построения минимального остовного дерева гораздо быстрее алгоритма Прима.