Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №9

по теме «Поиск расстояний во взвешенном графе»

Выполнили:

Студенты группы 21ВВ2

Козлова К.С.

Кривенкова В.С.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2022

**Цель работы:**

Изучить основные алгоритмы поиска расстояний во взвешенном графе.

**Лабораторные работы:**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

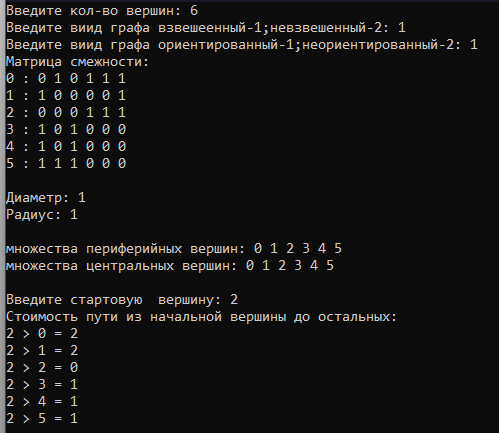
**3.**\* Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа *G*. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

**Задание 2**

1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

**Задание 3\***

1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже).  В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

**Результат работы программы:**

**Листинг:**

#include <iostream>

#include <list>

#include <set>

#include <queue>

using namespace std;

void toString(int\*\* Matrix, int n) {

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

cout << i << " : ";

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

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

}

cout << endl;

}

}

void addEdge(int\*\* Matrix, int i, int j, int w, int u)

{

u == 0 ? Matrix[i][j] = w : u == 1 ? Matrix[j][i] = w : Matrix[i][j] = Matrix[j][i] = w;

}

int main()

{

setlocale(0, "");

srand(time(0));

int n;

printf\_s("Введите кол-во вершин: ");

scanf\_s("%d", &n);

int\*\* Matrix = (int\*\*)malloc(sizeof(int\*) \* n);

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

Matrix[i] = (int\*)malloc(sizeof(int\*) \* n);

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

Matrix[i][j] = 0; // создание матрицы смежности

}

int k, b;

printf\_s("Введите виид графа взвешеенный-1;невзвешенный-2: ");

scanf\_s("%d", &b); // (взвешенный или нет) вводить 1 или 2

printf\_s("Введите виид графа ориентированный-1;неориентированный-2: ");

scanf\_s("%d", &k); // (есть ориентация или нет) вводить 1 или 2

if (b == 1 && k == 1) {

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

{

for (int j = i + 1; j < n; j++)

{

if (0 < (rand() % 3)) {

addEdge(Matrix, i, j, 1, 3);

}

}

}

}

if (b == 1 && k == 2) {

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

{

for (int j = i + 1; j < n; j++)

{

if (0 < (rand() % 3)) {

addEdge(Matrix, i, j, 1, rand() % 3);

}

}

}

}

if (b == 2 && k == 1) {

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

{

for (int j = i + 1; j < n; j++)

{

if (0 < (rand() % 3)) {

addEdge(Matrix, i, j, rand() % 50, 3);

}

}

}

}

if (b == 2 && k == 2) {

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

{

for (int j = i + 1; j < n; j++)

{

if (0 < (rand() % 3)) {

addEdge(Matrix, i, j, rand() % 50, rand() % 3);

}

}

}

}

printf\_s("Матрица смежности: \n");

toString(Matrix, n);

int\* ext = (int\*)malloc(sizeof(int\*) \* n);

set <int> center, pereph;

int rad = INT\_MAX, diam;

int\*\* d = (int\*\*)malloc(sizeof(int\*) \* n);

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

d[i] = (int\*)malloc(sizeof(int\*) \* n);

// Алгоритм Флойда-Уоршелла

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

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

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

d[i][j] = min(Matrix[i][j], Matrix[i][k] + Matrix[k][j]);

}

}

}

// Нахождение эксцентриситета

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

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

ext[i] = max(ext[i], d[i][j]);

}

}

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

if (min(rad, ext[i]) > 0) {

rad = min(rad, ext[i]);

}

diam = max(diam, ext[i]);

}

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

if (ext[i] == rad) {

center.insert(i);

}

}

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

if (ext[i] == diam) {

pereph.insert(i);

}

}

cout << endl << "Диаметр: " << diam << endl << "Радиус: " << rad << endl;

cout << endl << "множества периферийных вершин: ";

for (int i : pereph)

cout << i << " ";

cout << endl << "множества центральных вершин: ";

for (int i : center)

cout << i << " ";

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

free(d[i]);

free(d);

free(ext);

//////////////////////////////////////////////////////////////////////////////

int st;

printf\_s("\n\nВведите стартовую вершину: "); scanf\_s("%d", &st);

int\* dist = (int\*)malloc(sizeof(int\*) \* n);

int\* visit = (int\*)malloc(sizeof(int\*) \* n);

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

visit[i] = false, dist[i] = INT\_MAX;

dist[st] = 0;

visit[st] = true;

queue<int>q;

q.push(st);

while (!q.empty()) {

int t = q.front();

q.pop();

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

if (!visit[i] && Matrix[t][i] && Matrix[t][i] + dist[t] < dist[i])

{ // Ставим условия заходим только если вершина не посещена, с этой вершиной есть ребро, расстояние между вершинами меньше чем уже существует

dist[i] = Matrix[t][i] + dist[t];

q.push(i);

}

}

}

cout << "Стоимость пути из начальной вершины до остальных:\t\n";

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

if (dist[i] != INT\_MAX)

cout << st << " > " << i << " = " << dist[i] << endl;

else

cout << st << " > " << i << " = " << "маршрут недоступен" << endl;

free(dist);

free(visit);

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

free(Matrix[i]);

free(Matrix);

return 0;

}

**Вывод:** в ходе данной лабораторной работы мы научились реализовывать основные алгоритмы поиска расстояний во взвешенном графе.