Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №10

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний во взвешенном графе»

Выполнили:

студенты группы 23ВВВ4

Беглов И. В.

Тинчурин Д. А.

Приняли:

Юрова О. В.

Деев М. В.

Пенза 2024

**Название**

Поиск расстояний во взвешенном графе.

**Цель работы**

Реализация алгоритма поиска расстояний во взвешенном графе на основе обходов в глубину и ширину.

**Лабораторное задание**

### 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. В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

**Пояснительный текст к программе**

В алгоритме BFS создается пустая очередь, в которую помещается исходная вершина, из которой начат обход. Расстояние до этой вершины устанавливается равным 0.

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной. Затем в цикле просматривается v-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с v не посещенную вершину , эта вершина помещается в очередь и для нее обновляется вектор расстояния . Расстояние до новой i-й вершины вычисляется как расстояние до текущей v-й вершины плюс вес ребра до новой вершины G(v,i).

Назначение отдельных операторов программы указывается в виде комментариев. Программа завершает свою работу после нажатия на любую клавишу на клавиатуре.

**Результат выполнения программы**

Результат работы программы представлен на рис. 1-3.

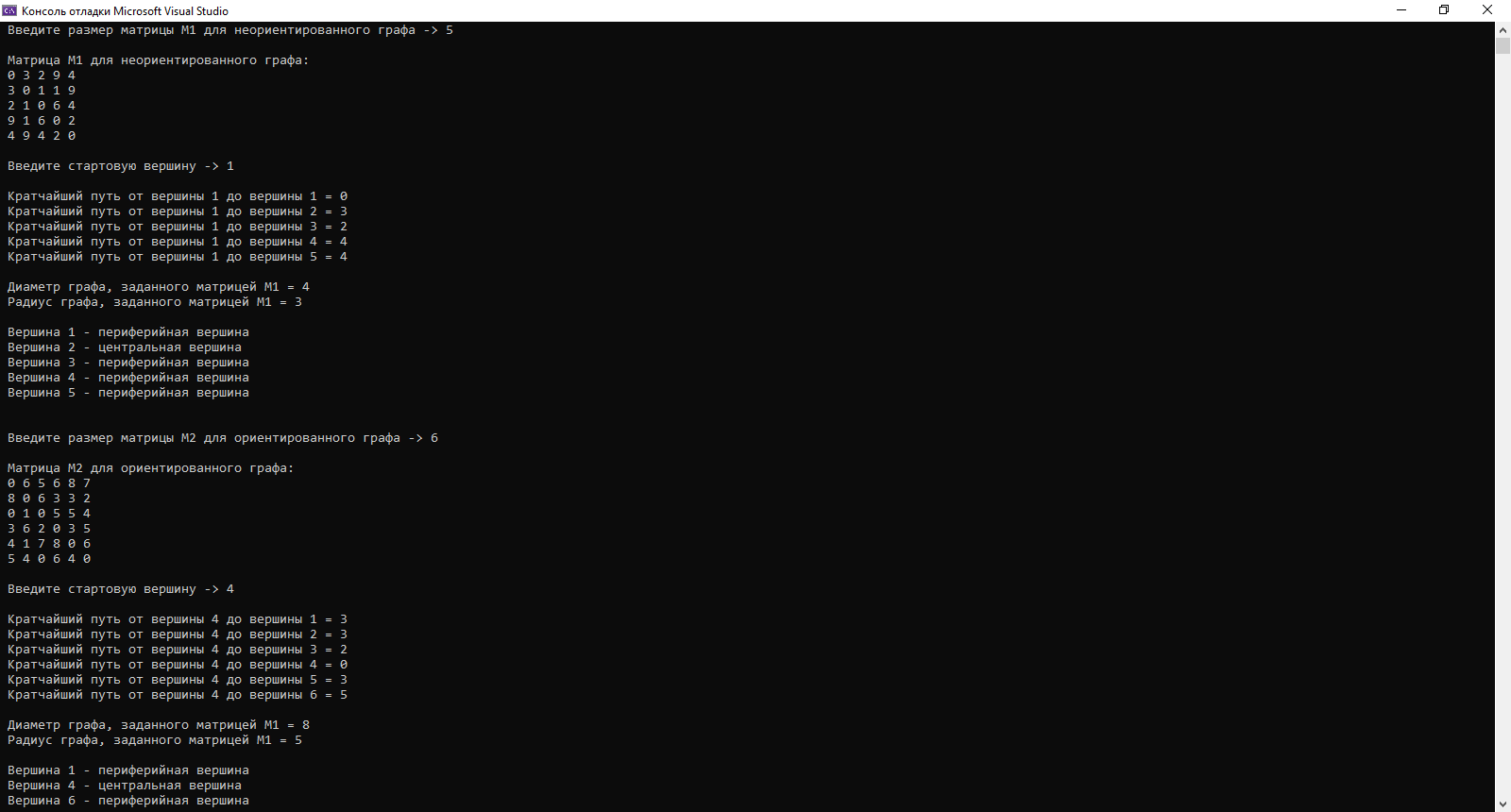


Рисунок 1

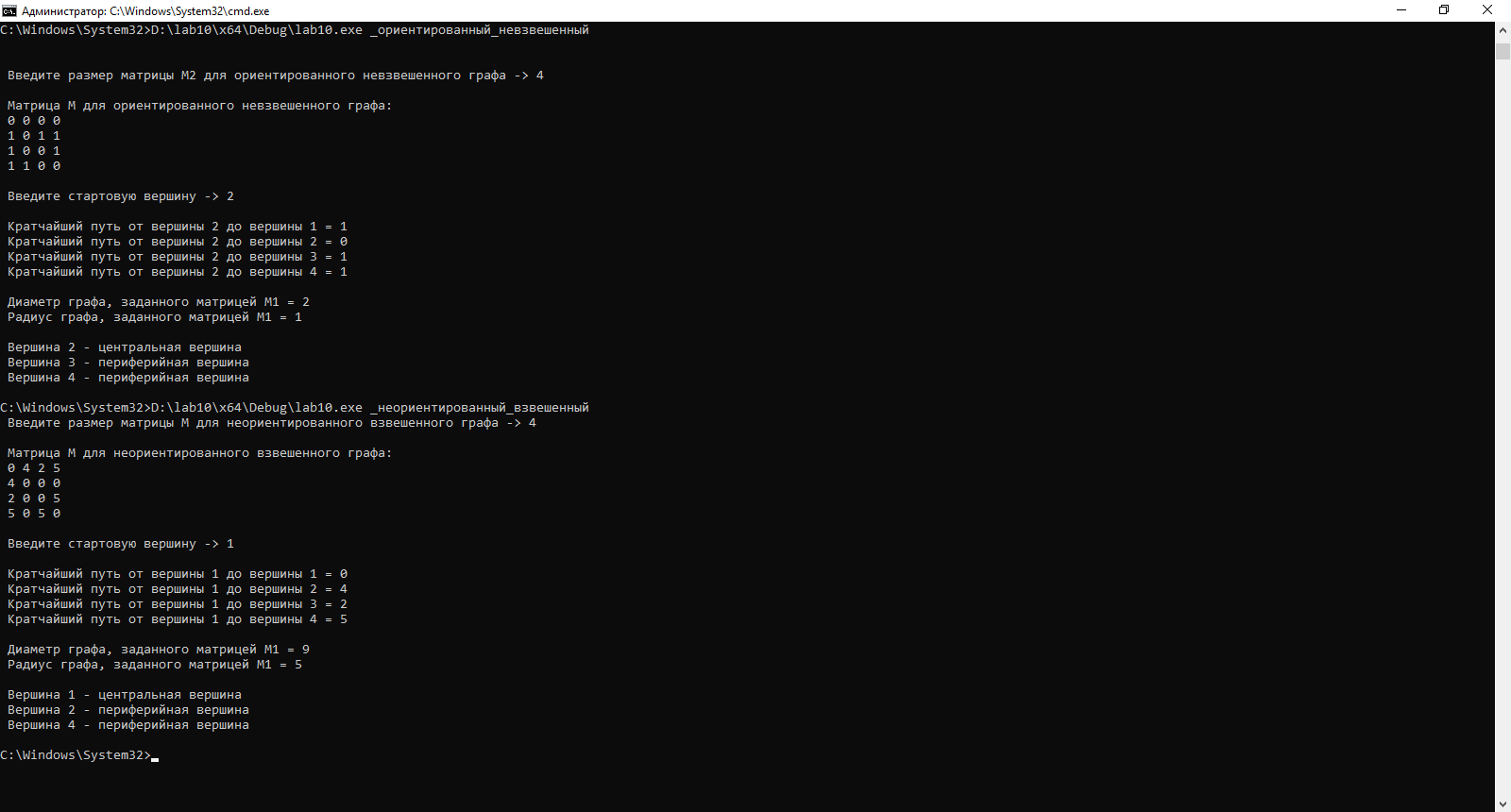


Рисунок 2

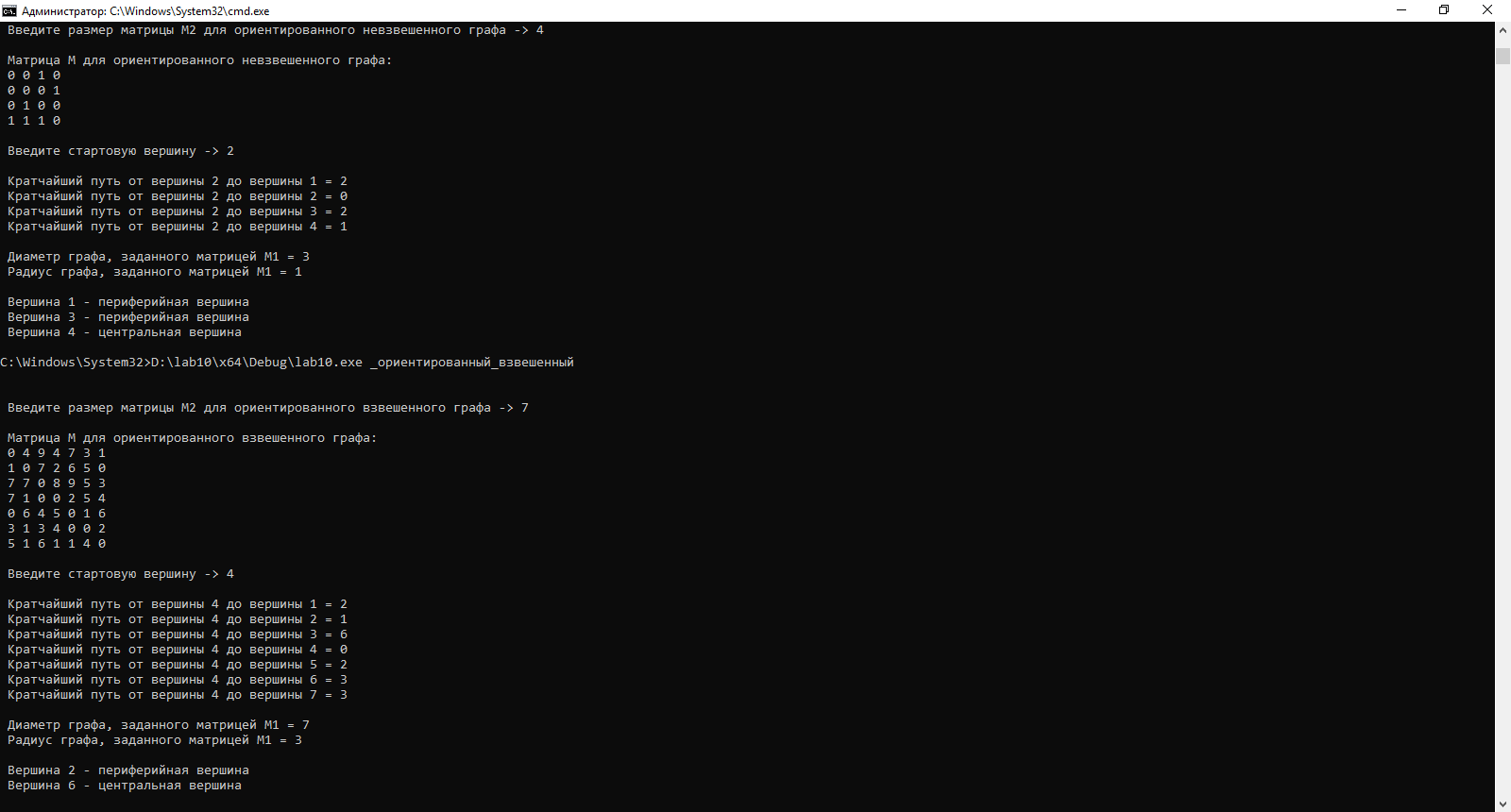


Рисунок 3

**Выводы**

В ходе выполнения лабораторной работы была разработана на языке Си программа, осуществляющая поиск кратчайших расстояний во взвешенном графе на основе обходов в ширину и в глубину. Также программа была модернизирована: осуществлен запуск программы с параметрами командной строки.

**Приложение**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <queue>

#include <iostream>

using namespace std;

void BFSD(int v\_start, int n, int\*\* G, int\* dist) {

int v;

using namespace std;

queue <int> Q;

Q.push(v\_start);

dist[v\_start] = 0;

while (!Q.empty()) {

v = Q.front();

Q.pop();

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

if ((G[v][i] > 0) && (dist[v] + G[v][i] < dist[i])) {

Q.push(i);

dist[i] = dist[v] + G[v][i];

}

}

}

return;

}

void process(int n, int\*\* G, int\* dist){

int v\_start;

printf("\n Введите стартовую вершину -> ");

scanf("%d", &v\_start);

printf("\n");

BFSD(v\_start, n, G, dist);

for (int i = 1; i <= n; i++) { //вывод результата

if (dist[i] == 1000) printf(" Кратчайшего пути от вершины %d до вершины %d не существует\n", v\_start, i);

else printf(" Кратчайший путь от вершины %d до вершины %d = %d\n", v\_start, i, dist[i]);

}

int max\_dist = 0;

int diametr = 0;

int radius = 1000;

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

BFSD(i, n, G, dist);

max\_dist = 0;

for (int i\_4\_diam\_rad = 1; i\_4\_diam\_rad <= n; i\_4\_diam\_rad++) {

if (dist[i\_4\_diam\_rad] > max\_dist) max\_dist = dist[i\_4\_diam\_rad]; //поиск максимального расстояния в массиве

}

if (diametr < max\_dist) diametr = max\_dist; //поиск максимального эксцентрисита - диаметра

if (radius > max\_dist) radius = max\_dist; //поиск минимального эксцентрисита - радиуса

for (int i\_4\_dist = 1; i\_4\_dist <= n; i\_4\_dist++) { //восстановление массива для след вызова BFSD

dist[i\_4\_dist] = 1000;

}

}

printf("\n Диаметр графа, заданного матрицей М1 = %d\n", diametr);

printf(" Радиус графа, заданного матрицей М1 = %d\n\n", radius);

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

BFSD(i, n, G, dist);

max\_dist = 0;

for (int i\_4\_find = 1; i\_4\_find <= n; i\_4\_find++) {

if (dist[i\_4\_find] > max\_dist) max\_dist = dist[i\_4\_find]; //поиск максимального расстояния в массиве - эксцентриситет

}

if (max\_dist == diametr) printf(" Вершина %d - периферийная вершина\n", i);

if (max\_dist == radius) printf(" Вершина %d - центральная вершина\n", i);

for (int i\_4\_dist = 1; i\_4\_dist <= n; i\_4\_dist++) { //восстановление массива для след вызова BFSD

dist[i\_4\_dist] = 1000;

}

}

return;

}

int main() {

int n;

int n\_orient;

setlocale(0, "rus");

printf(" Введите размер матрицы М1 для неориентированного графа -> "); //неориентированный граф

scanf("%d", &n);

printf("\n");

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

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

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

}

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

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

dist[i] = 1000;

}

printf(" Матрица М1 для неориентированного графа:\n");

srand(time(0));

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

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

if (j >= i) {

if (i == j) G[i][j] = 0;

else {

G[i][j] = rand() % 10;

G[j][i] = G[i][j];

}

}

printf(" %d", G[i][j]);

}

printf("\n");

}

process(n, G, dist);

printf("\n\n Введите размер матрицы М2 для ориентированного графа -> "); //ориентированный граф

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

printf("\n");

int\*\* G\_orient = (int\*\*)malloc(n\_orient \* sizeof(int\*));

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

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

}

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

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

dist\_orient[i] = 1000;

}

printf(" Матрица М2 для ориентированного графа:\n");

srand(time(0));

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

for (int j = 1; j <= n\_orient; j++) {

if (i == j) G\_orient[i][j] = 0;

else G\_orient[i][j] = rand() % 10;

printf(" %d", G\_orient[i][j]);

}

printf("\n");

}

process(n\_orient, G\_orient, dist\_orient);

return 0;

}