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 в графе**»

Выполнил:

студент группы 22ВВП1

Воробьева М.М.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Цель работы:** освоить такую операцию, как поиск расстояний от одной вершины в другую в взвешенных графах, представленных в виде матрицы и списка смежности, на основе алгоритмов обхода графов в ширину, определение основных характеристик графа.

**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.

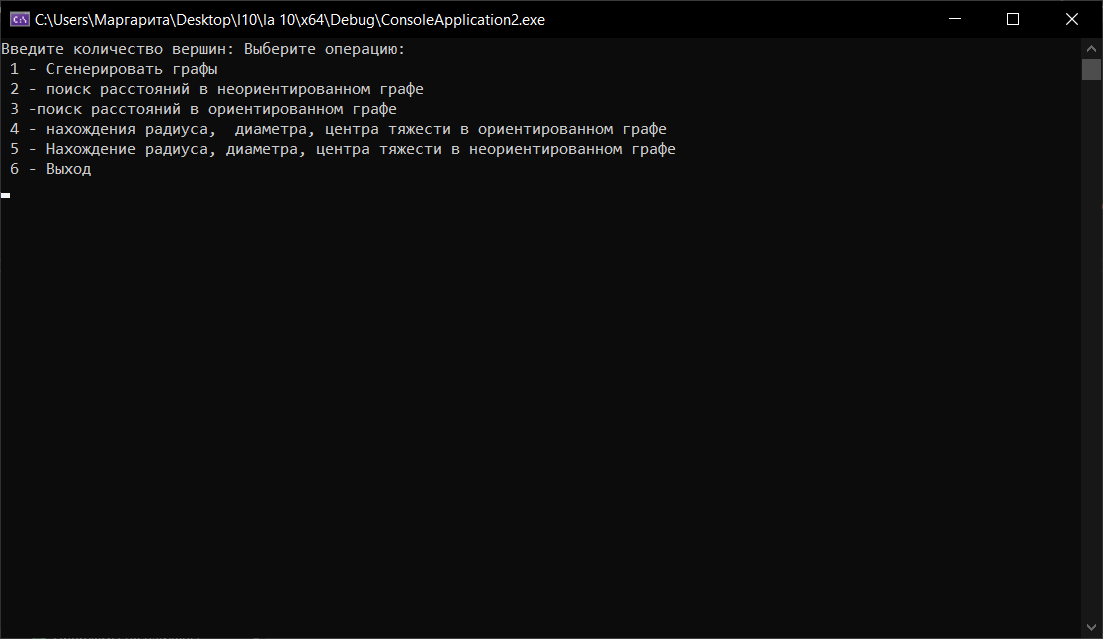
**Задание 2**

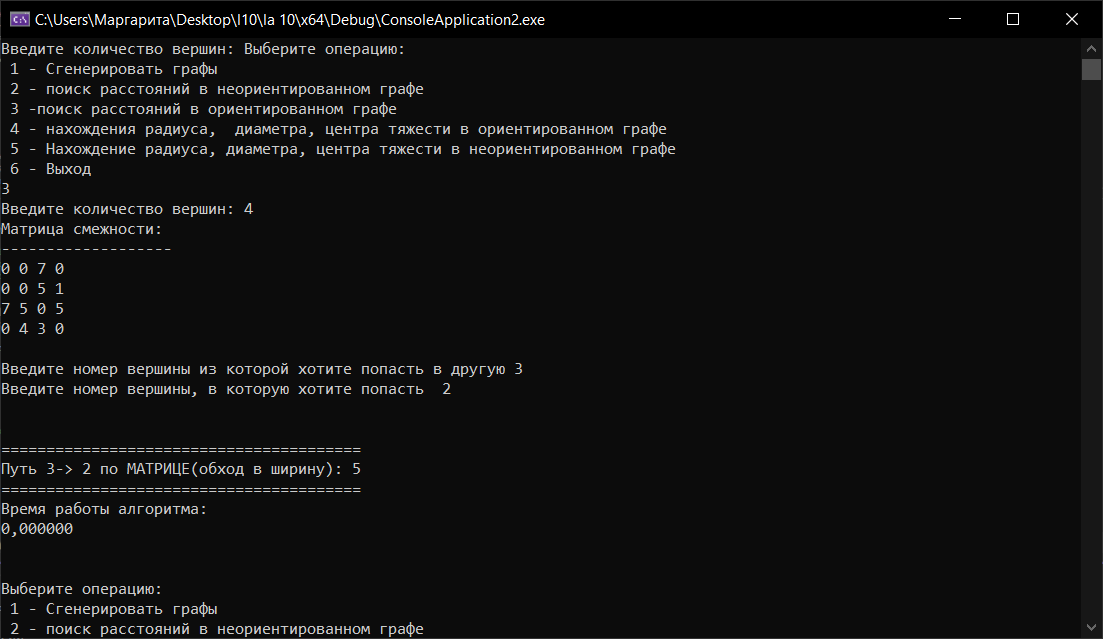
1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

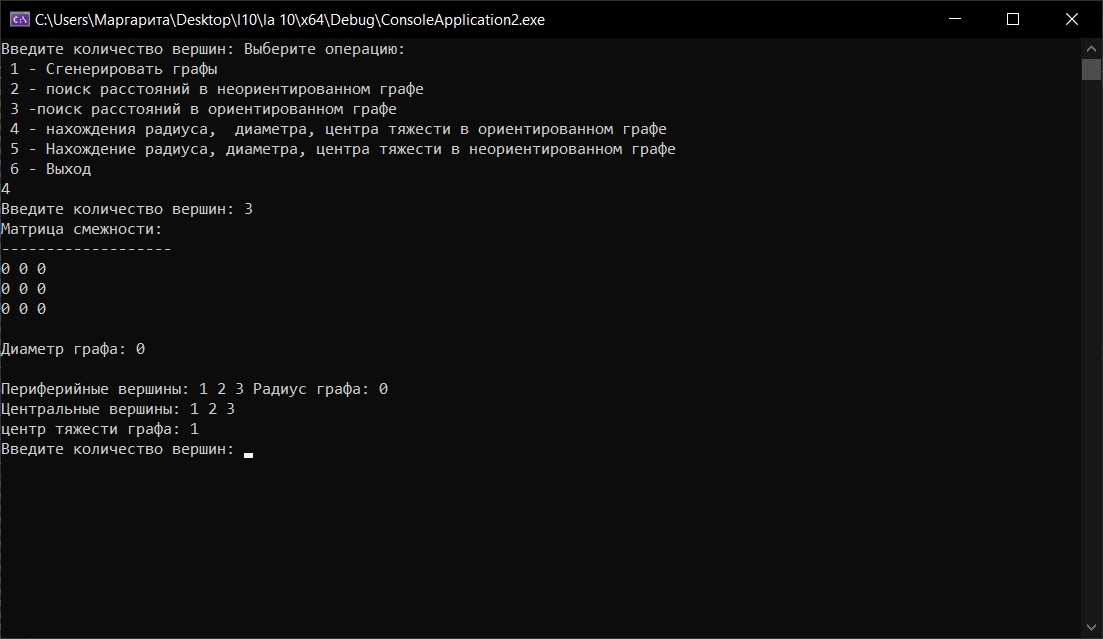
**Задание 3\***

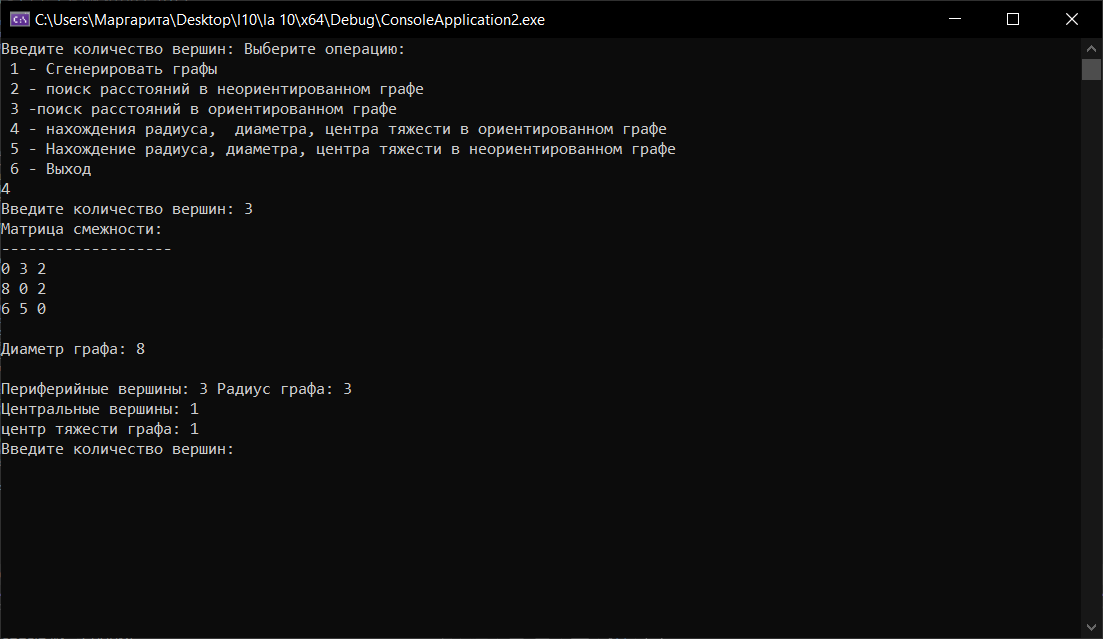
1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже).  В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

**Результат работы программы:**









#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <queue>

#include <iostream>

using namespace std;

queue<int>Q;

void centor\_tiajesti(vector < vector < int>> G) {

vector<int> vis(G.size() + 1);

int s = 1;

for (int p = 1; p <= G.size(); p++) {

vector<int> dist(G.size() + 1, -1);

Q.push(s);

dist[s] = 0;

while (!Q.empty()) {

s = Q.front();

Q.pop();

//printf("%d\n", s);

for (int i = 1; i < G.size(); ++i) {

if (G[s][i] > 0 and dist[i] == -1) {

Q.push(i);

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

}

}

}

int dist\_max = 0;

for (int c : dist) {

if (c != -1) {

dist\_max = max(dist\_max, c);

}

}

vis[p] = dist\_max;

}

int min\_eccentricity = INT\_MAX;

int centroid = -1;

for (int v = 1; v <= G.size(); ++v) {

if (vis[v] < min\_eccentricity) {

min\_eccentricity = vis[v];

centroid = v;

}

}

printf("центр тяжести графа: %d\n", centroid);

}

void BFSD\_rad(vector < vector < int>> G, int num\_vertices) {

vector<int> vis(G.size() + 1, 0);

int rad = INT\_MAX;

int diam = INT\_MIN;

int s = 1;

for (int p = 1; p <= G.size()-1; p++) {

vector<int> dist(G.size() + 1, -1);

Q.push(s);

dist[s] = 0;

while (!Q.empty()) {

s = Q.front();

Q.pop();

//printf("%d\n", s);

for (int i = 1; i < G.size(); i++) {

if (G[s][i] > 0 and dist[i] == -1) {

Q.push(i);

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

}

}

}

int dist\_max = 0;

for (int c : dist) {

if (c != -1) {

dist\_max = max(dist\_max, c);

}

}

vis[p] = dist\_max;

rad = min(rad, dist\_max);

for (int i = 1; i <= G.size(); i++) {

dist[i] = 1000;

}

}

printf("Радиус графа: %d\n", rad);

printf("Центральные вершины: ");

for (int i = 1; i <= G.size()-1; i++) {

if (vis[i] == rad) {

printf("%d ", i);

}

}

printf("\n");

}

void BFSD\_diam(vector < vector < int>> G) {

vector<int> vis(G.size() +1, 0);

int rad = INT\_MAX;

int diam = INT\_MIN;

int s = 1;

for (int p = 1; p <= G.size(); p++) {

vector<int> dist(G.size() + 1, -1);

Q.push(s);

dist[s] = 0;

while (!Q.empty()) {

s = Q.front();

Q.pop();

//printf("%d\n", s);

for (int i = 1; i < G.size(); i++) {

if (G[s][i] > 0 and dist[i] == -1) {

Q.push(i);

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

}

}

}

int dist\_max = 0;

for (int c : dist) {

if (c != -1) {

dist\_max = max(dist\_max, c);

}

}

vis[p] = dist\_max;

diam = max(diam, dist\_max);

//rad = min(rad, dist\_max);

}

printf("Диаметр графа: %d\n", diam);

//printf("Радиус графа: %d\n", rad);

printf("\n");

printf("Периферийные вершины: ");

for (int i = 1; i <= G.size()-1; i++) {

if (vis[i] == diam) {

printf("%d ", i);

}

}

}

void BFSD1\_or(vector < vector < int>> G\_or, int s, vector<int>& dist\_or, int d) {

clock\_t start = clock();

int c = s;

Q.push(s);

dist\_or[s] = 0;

while (!Q.empty()) {

s = Q.front();

Q.pop();

//printf("%d\n", s);

for (int i = 1; i < G\_or.size(); ++i) {

if (G\_or[s][i] > 0 and dist\_or[i] == -1) {

Q.push(i);

dist\_or[i] = dist\_or[s] + G\_or[s][i];

}

}

}

clock\_t end = clock();

if (dist\_or[d] == -1) {

printf("======================\n");

printf("Пути отсутствуют!\n");

printf("======================\n");

}

else {

printf("========================================\n");

printf("Путь %d-> %d по МАТРИЦЕ(обход в ширину): %d\n", c, d, dist\_or[d]);

printf("========================================\n");

}

printf("Время работы алгоритма: \n");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n\n");

}

void BFSD1(vector < vector < int>> G, int s, vector<int>& dist, int d) {

clock\_t start = clock();

int c = s;

Q.push(s);

dist[s] = 0;

for (int i = 1; i <= G.size(); i++) {

while (!Q.empty()) {

s = Q.front();

Q.pop();

//printf("%d\n", s);

for (int i = 1; i < G.size(); ++i) {

if (G[s][i] > 0 and dist[i] == -1) {

Q.push(i);

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

}

}

}

}

clock\_t end = clock();

if (dist[d] == -1) {

printf("======================\n");

printf("Пути отсутствуют!\n");

printf("======================\n");

}

else {

printf("========================================\n");

printf("Путь %d-> %d по МАТРИЦЕ(обход в ширину): %d\n", c, d, dist[d]);

printf("========================================\n");

}

printf("Время работы алгоритма: ");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n\n");

}

int swt = 0;

vector<vector<int>> cmej\_mat\_orient(int num\_vertices, double veroj) {

vector<vector<int>> cmej\_mat\_orient(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

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

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

if (i!=j and static\_cast<double>(rand()) / RAND\_MAX < veroj)

{

int r = rand() % 10 + 1;

int t = rand() % 10 + 1;

cmej\_mat\_orient[i][j] =r;

cmej\_mat\_orient[j][i] = t;

//cont++;

}

}

}

//if (cont == 0) {

//swt = 1;

//}

printf("Матрица смежности:\n");

printf("-------------------\n");

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

for (int j = 1; j <= num\_vertices; ++j) {

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

}

printf("\n");

}

printf("\n");

return cmej\_mat\_orient;

}

vector<vector<int>> cmej\_mat(int num\_vertices, double veroj) {

vector < vector < int>> cmej\_mat(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

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

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

if (i != j and static\_cast<double>(rand()) / RAND\_MAX < veroj) {

int t = rand() % 10 + 1;

cmej\_mat[i][j] = t;

cmej\_mat[j][i] = t;

}

}

}

/\*if (cont == 0) {

swt = 1;

}\*/

printf("Матрица смежности:\n");

printf("-------------------\n");

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

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", cmej\_mat[i][j]);

}

printf("\n");

}

printf("\n");

return cmej\_mat;

}

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "RUS");

srand(static\_cast<unsigned int>(time(0)));

int num\_vertices, operation;

double veroj = 0.7;

int s, d, v, r;

printf("Введите количество вершин: ");

if (argc > 1) {

r = atoi(argv[1]);

printf("%d", r);

num\_vertices = r;

}

while (true) {

printf("Выберите операцию:\n 1 - Сгенерировать графы\n 2 - поиск расстояний в неориентированном графе\n 3 -поиск расстояний в ориентированном графе\n 4 - нахождения радиуса, диаметра, центра тяжести в ориентированном графе\n 5 - Нахождение радиуса, диаметра, центра тяжести в неориентированном графе\n 6 - Выход\n");

if (scanf("%d", &operation) != 1) {

printf("Ошибка: ожидался ввод числа.\n");

while (getchar() != '\n'); // Очистка ввода

continue;

}

if (operation == 6) {

break;

}

switch (operation) {

case 1: {

auto G = cmej\_mat(num\_vertices, veroj);

auto G\_or = cmej\_mat\_orient(num\_vertices, veroj);

break;

}

case 2: {

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

auto G = cmej\_mat(num\_vertices, veroj);

vector<int> dist(G.size(), -1);

printf("Введите номер вершины из которой хотите попасть в другую ");

scanf("%d", &s);

printf("Введите номер вершины, в которую хотите попасть ");

scanf("%d", &d);

printf("\n\n");

if (swt == 1) {

printf("\n Обход произвести нельзя!\n");

}

else {

BFSD1(G, s, dist, d);

break;

}

}

case 3: {

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

auto G\_or = cmej\_mat\_orient(num\_vertices, veroj);

vector<int> dist(G\_or.size(), -1);

printf("Введите номер вершины из которой хотите попасть в другую ");

scanf("%d", &s);

printf("Введите номер вершины, в которую хотите попасть ");

scanf("%d", &d);

printf("\n\n");

if (swt == 1) {

printf("\n Обход произвести нельзя!\n");

}

else {

BFSD1\_or(G\_or, s, dist, d);

break;

}

}

case 4: {

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

auto G\_or = cmej\_mat\_orient(num\_vertices, veroj);

vector<int>dist\_or(G\_or.size(), -1);

BFSD\_diam(G\_or);

BFSD\_rad(G\_or, num\_vertices);

centor\_tiajesti(G\_or);

}

case 5: {

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

auto G = cmej\_mat(num\_vertices, veroj);

vector<int>dist\_or(G.size(), -1);

BFSD\_diam(G);

BFSD\_rad(G, num\_vertices);

centor\_tiajesti(G);

}

return 0;

}

}

}

**Вывод:** в ходе выполнения лабораторной работы была освоена такая операция, как поиск расстояний от одной вершины в другую в взвешенных графах, представленных в виде матрицы и списка смежности, на основе алгоритмов обхода графов в ширину, определение основных характеристик графа.