Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №10

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: “Поиск расстояний во взвешенном графе”

Выполнили студенты гр. 22ВВП2:

Корнилов В.М.

Самофалова А.В.

Горбунов Д.А.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза,2023

**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++.

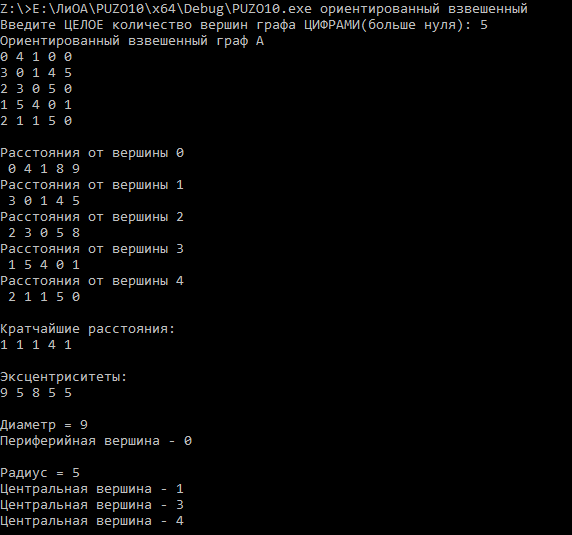
**3.**\* Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа *G*. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

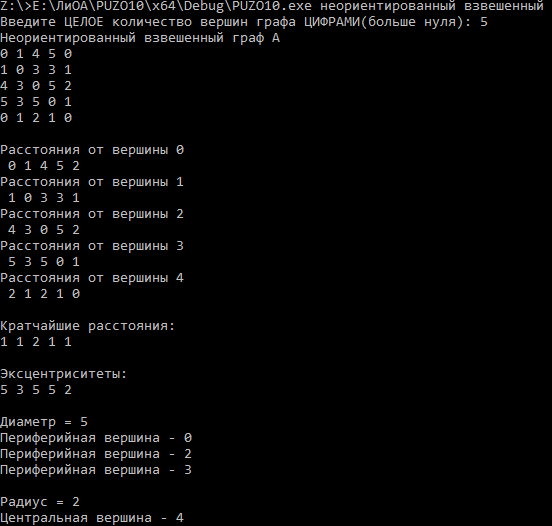
**Задание 2**

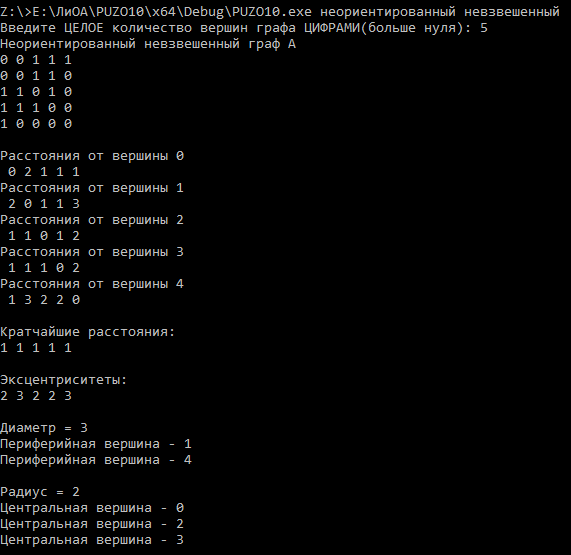
1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

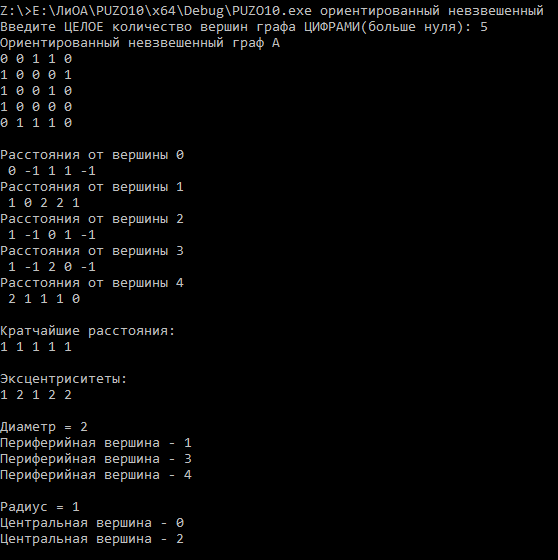
**Задание 3\***

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).







**Вывод:**

Научились искать расстояния во взвешенном графе. Научились передавать параметры в программу через командную строку.

**Листинг**

**ФайлPUZO10.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

#include<queue>

#include<time.h>

#include<iostream>

using namespace std;

int BFSD(int\*\* A, int\* dist, int n, int v, int\* distm, int\* exc) {

queue<int> q;

q.push(v);

dist[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

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

if ((A[v][i] > 0) && (dist[i] ==-1)) {

q.push(i);

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

if ((distm[i] > dist[i]) && (dist[i] != 0))

distm[i] = dist[i];

}

}

}

return 0;

}

int oper(int\*\* A, int\* dist, int n, int v, int\* distm, int\* exc) {

int err, d = -1, d1 = -1, r = 100000, r1 = 100000;

printf("\n");

for (v = 0; v < n; v++) {

printf("Расстояния от вершины %d\n ", v);

BFSD(A, dist, n, v, distm, exc);

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

printf("%d ", dist[i]);

if ((exc[v] < dist[i]) && (dist[i] != 0))

exc[v] = dist[i];

}

printf("\n");

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

dist[i] = -1;

}

}

printf("\nКратчайшие расстояния:\n");

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

printf("%d ", distm[i]);

}

printf("\n");

printf("\nЭксцентриситеты:\n");

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

printf("%d ", exc[i]);

}

printf("\n\n");

//определение диаметра и периферийной вершины

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

if (d < exc[i])

d = exc[i];

}

printf("Диаметр = %d\n", d);

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

if (d == exc[i])

printf("Периферийная вершина - %d\n", i);

}

printf("\n");

//определение радиуса и центральной вершины

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

if ((r > exc[j]) && (exc[j] > 0))

r = exc[j];

}

printf("Радиус = %d\n", r);

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

if (r == exc[j])

printf("Центральная вершина - %d\n", j);

}

return 0;

}

int neorv(int\*\* A, int\* dist, int n, int v, int\* distm, int\* exc) {

srand(time(NULL));

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

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

A[i][j] = rand() % 6;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

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

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

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

}

printf("\n");

}

oper(A, dist, n, v, distm, exc);

return 0;

}

int neorn(int\*\* A, int\* dist, int n, int v, int\* distm, int\* exc) {

srand(time(NULL));

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

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

A[i][j] = rand() % 2;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

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

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

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

}

printf("\n");

}

oper(A, dist, n, v, distm, exc);

return 0;

}

int orv(int\*\* A, int\* dist, int n, int v, int\* distm, int\* exc) {

srand(time(NULL));

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

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

A[i][j] = rand() % 6;

if (i == j) {

A[i][j] = 0;

}

}

}

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

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

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

}

printf("\n");

}

oper(A, dist, n, v, distm, exc);

return 0;

}

int orn(int\*\* A, int\* dist, int n, int v, int\* distm, int\* exc) {

srand(time(NULL));

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

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

A[i][j] = rand() % 2;

if (i == j) {

A[i][j] = 0;

}

}

}

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

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

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

}

printf("\n");

}

oper(A, dist, n, v, distm, exc);

return 0;

}

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

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n = 0;

int\*\* A;

int\* dist;

int\* distm;

int\* exc;

int v = 0, err;

printf("Введите ЦЕЛОЕ количество вершин графа ЦИФРАМИ(больше нуля): ");

err = scanf("%d", &n);

if ((err == 0) || (n <= 0)) {

printf("Ошибка! Введите число заново\n");

return 0;

}

dist = (int\*)calloc(n, sizeof(int));

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

dist[i] = -1;

}

distm = (int\*)calloc(n, sizeof(int));

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

distm[i] = 10000;

}

exc = (int\*)calloc(n, sizeof(int));

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

exc[i] = -1;

}

srand(time(NULL));

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

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

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

}

if (argc > 1) {

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

if (((strcmp(argv[i], "взвешенный") == 0)|| (strcmp(argv[i], "неориентированный") == 0))&& ((strcmp(argv[i+1], "взвешенный") == 0) || (strcmp(argv[i+1], "неориентированный") == 0))) {

printf("Неориентированный взвешенный граф А \n");

neorv(A, dist, n, v, distm, exc);

return 0;

}

if (((strcmp(argv[i], "невзвешенный") == 0) || (strcmp(argv[i], "неориентированный") == 0)) && ((strcmp(argv[i + 1], "невзвешенный") == 0) || (strcmp(argv[i + 1], "неориентированный") == 0))) {

printf("Неориентированный невзвешенный граф А \n");

neorn(A, dist, n, v, distm, exc);

return 0;

}

if (((strcmp(argv[i], "взвешенный") == 0) || (strcmp(argv[i], "ориентированный") == 0)) && ((strcmp(argv[i + 1], "взвешенный") == 0) || (strcmp(argv[i + 1], "ориентированный") == 0))) {

printf("Ориентированный взвешенный граф А \n");

orv(A, dist, n, v, distm, exc);

return 0;

}

if (((strcmp(argv[i], "невзвешенный") == 0) || (strcmp(argv[i], "ориентированный") == 0)) && ((strcmp(argv[i + 1], "невзвешенный") == 0) || (strcmp(argv[i + 1], "ориентированный") == 0))) {

printf("Ориентированный невзвешенный граф А \n");

orn(A, dist, n, v, distm, exc);

return 0;

}

//else {

// printf("Ошибка");

// return 0;

//}

}

}

else

printf("Ошибка");

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

free(A[i]);

free(A);

free(dist);

free(distm);

free(exc);

return 0;

}