Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 10

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний во взвешенном графе»

Выполнили:

студенты группы 22ВВВ2:

Зубриянова А.А.

Кондратьева В.И.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Поиск расстояний во взвешенном графе.

**Цель работы**

Научиться выполнять поиск расстояний во взвешенном графе.

**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).

**Описание метода решения задачи**

**Задание 1**

1. Сгенерировали (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа G.

2. Реализовали процедуру поиска расстояний на основе обхода в ширину. Использовали рекурсивную функцию BFSD().

3. \*Сгенерировали (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа G. Реализовали процедуру поиска расстояний на основе обхода в ширину. Использовали рекурсивную функцию BFSD().

**Задание 2**

1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определили радиус и диаметр. Для нахождения расстояний использовали обход в ширину.

2. Определили подмножества периферийных и центральных вершин, найдя вершины с максимальным и минимальным эксцентриситетом.

**Задание 3\***

1. Модернизировали программу для получения возможности ее запуска с параметрами командной строки. В качестве параметров указываются наличие ориентации ребер графа (есть ориентация или нет) и тип графа (взвешенный), а также размер графа.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#include <conio.h>

#include <iostream>

#include <queue>

using namespace std;

queue <int> Q;

int size1;

void create\_matrix(int\*\* arr1) {

for (int i = 0; i < size1; i++) {

for (int j = i; j < size1; j++) {

if (i == j) { //главная диагональ == 0

arr1[i][j] = 0;

}

else { //ниже главной диагонали отразить то, что выше

arr1[i][j] = rand() % 5;

arr1[j][i] = arr1[i][j];

}

}

}

}

void print\_matrix(int\*\* arr) {

printf(" ");

for (int i = 0; i < size1; i++) {

printf("%4d ", i);

}

printf("\n");

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

for (int i = 0; i < size1; i++) {

printf("%4d|", i);

for (int j = 0; j < size1; j++) {

printf("%4d ", arr[i][j]);

}

printf("\n");

}

printf("\n");

}

int\* BFSD(int v, int\*\* arr, int\* dist)

{

Q.push(v);

dist[v] = 0;

while (!Q.empty()) {

v = Q.front();

Q.pop();

for (int i = 0; i < size1; i++) {

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

Q.push(i);

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

}

}

}

return dist;

}

void exz\_nolik(int\*\* exz) {

for (int i = 0; i < size1; i++) {

for (int j = i; j < size1; j++) {

exz[i][j] = 0;

}

}

}

void centre\_perif(int maxim, int\*\* exz, int\* mass) {

for (int i = 0; i < size1; i++) {

mass[i] = 0;

}

for (int i = 0; i < size1; i++) {

for (int j = 0; j < size1; j++) {

if (exz[i][j] == maxim) {

mass[i] = 1;

}

}

}

for (int i = 0; i < size1; i++) {

if (mass[i] == 1) {

printf("%5d", i);

}

}

printf("\n");

}

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

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

if (argc > 1)

{

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

{

if ((strcmp(argv[i], "н\_в") == 0) or (strcmp(argv[i], "о\_в") == 0) or (strcmp(argv[i], "size") == 0))

{

if ((strcmp(argv[i], "size") == 0))

{

size1 = atoi(argv[i + 1]);

//printf("%s\n", argv[i+1]);

}

int\* dist1 = (int\*)malloc(sizeof(int) \* size1);

int\* dist2 = (int\*)malloc(sizeof(int) \* size1);

int\*\* arr1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr1[i] = (int\*)malloc(sizeof(int) \* size1);

}

int\*\* arr2 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr2[i] = (int\*)malloc(sizeof(int) \* size1);

}

int max\_exz;

if (strcmp(argv[i], "н\_в") == 0)

{

printf("Задание 1.1\n");

printf("Матрица M1:\n");

create\_matrix(arr1);

print\_matrix(arr1);

int a;

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

printf("Задание 1.2 (поиск расстояний для неориентированного взвешенного граф)\n");

printf("Введите вершину (от 0 до %d) для начала обхода: ", size1 - 1);

scanf("%d", &a);

printf("\n");

for (int i = 0; i < size1; i++) {

dist1[i] = -1;

}

BFSD(a, arr1, dist1);

for (int i = 0; i < size1; i++) {

printf("Расстояние от %d вершины до %d вершины = %d\n", a, i, dist1[i]);

}

printf("\n");

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

printf("Задание 2.1 (радиус и диаметр для неориентированного графа)\n");

int\*\* exz1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

exz1[i] = (int\*)malloc(sizeof(int) \* size1);

}

exz\_nolik(exz1);

for (int k = 0; k < size1; k++) {

for (int i = 0; i < size1; i++) {

dist1[i] = -1;

}

BFSD(k, arr1, dist1);

max\_exz = 0;

for (int i = 0; i < size1; i++) {

if (dist1[i] > max\_exz) {

max\_exz = dist1[i];

}

}

for (int i = 0; i < size1; i++) {

if (dist1[i] != max\_exz) {

dist1[i] = 0;

}

}

for (int e = 0; e < size1; e++) {

exz1[k][e] = dist1[e];

}

}

printf("Матрица для эксцентриситета:\n");

print\_matrix(exz1);

int max1 = 0, min1 = 1000;

for (int i = 0; i < size1; i++) {

for (int j = 0; j < size1; j++) {

if (exz1[i][j] > max1) {

max1 = exz1[i][j];

}

else if ((exz1[i][j] < min1) && (exz1[i][j] != 0)) {

min1 = exz1[i][j];

}

}

}

printf("Радиус неориентированного = %d\n", min1);

printf("Диаметр неориентированного = %d\n", max1);

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

printf("Задание 2.2 (определение подмножества периферийных и центральных вершин)\n");

int\* mass1 = (int\*)malloc(sizeof(int) \* size1);

//printf("Неориентированный граф\n");

printf("Множество центральных вершин:");

centre\_perif(min1, exz1, mass1);

printf("Множество периферийных вершин вершин:");

centre\_perif(max1, exz1, mass1);

for (int i = 0; i < size1; ++i) free(exz1[i]);

free(exz1);

free(mass1);

}

if (strcmp(argv[i], "о\_в") == 0)

{

printf("Задание 1.3 (поиск расстояний для ориентированного взвешенного графа)\n");

printf("Матрица М2:\n");

for (int i = 0; i < size1; i++) { //цикл для задания массива

for (int j = 0; j < size1; j++) {

if (i == j) { //если главная диагональ

arr2[i][j] = 0;

}

else { //если выше главной диагонали

arr2[i][j] = rand() % 5;

}

}

}

printf("\n");

print\_matrix(arr2);

printf("\n");

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

printf("Поиск расстояний для ориентированного взвешенного графа:\n");

int b;

printf("Введите вершину (от 0 до %d) для начала обхода: ", size1 - 1);

scanf("%d", &b);

printf("\n");

for (int i = 0; i < size1; i++) {

dist2[i] = -1;

}

BFSD(b, arr2, dist2);

for (int i = 0; i < size1; i++) {

printf("Расстояние от %d вершины до %d вершины = %d\n", b, i, dist2[i]);

}

int\*\* exz2 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

exz2[i] = (int\*)malloc(sizeof(int) \* size1);

}

exz\_nolik(exz2);

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

printf("Задание 2.1 (радиус и диаметр для ориентированного графа)\n");

for (int k = 0; k < size1; k++) {

for (int i = 0; i < size1; i++) {

dist2[i] = -1;

}

BFSD(k, arr2, dist2);

max\_exz = 0;

for (int i = 0; i < size1; i++) {

if (dist2[i] > max\_exz) {

max\_exz = dist2[i];

}

}

for (int i = 0; i < size1; i++) {

if (dist2[i] != max\_exz) {

dist2[i] = 0;

}

}

for (int e = 0; e < size1; e++) {

exz2[k][e] = dist2[e];

}

}

printf("Матрица для эксцентриситета:\n");

print\_matrix(exz2);

int max2 = 0;

int min2 = 1000;

for (int i = 0; i < size1; i++) {

for (int j = 0; j < size1; j++) {

if (exz2[i][j] > max2) {

max2 = exz2[i][j];

}

else if ((exz2[i][j] < min2) && (exz2[i][j] != 0)) {

min2 = exz2[i][j];

}

}

}

printf("Радиус ориентированного = %d\n", min2);

printf("Диаметр ориентированного = %d\n", max2);

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

printf("Задание 2.2 (определение подмножества периферийных и центральных вершин)\n");

int\* mass2 = (int\*)malloc(sizeof(int) \* size1);

//printf("Ориентированный граф\n");

printf("Множество центральных вершин:");

centre\_perif(min2, exz2, mass2);

printf("Множество периферийных вершин вершин:");

centre\_perif(max2, exz2, mass2);

for (int i = 0; i < size1; ++i) free(exz2[i]);

free(exz2);

free(mass2);

}

for (int i = 0; i < size1; ++i) free(arr1[i]);

free(arr1);

for (int i = 0; i < size1; ++i) free(arr2[i]);

free(arr2);

free(dist1);

free(dist2);

}

}

}

\_getch();

return 0;

}

**Результаты работы программы**

Запуск программы через командную строку:

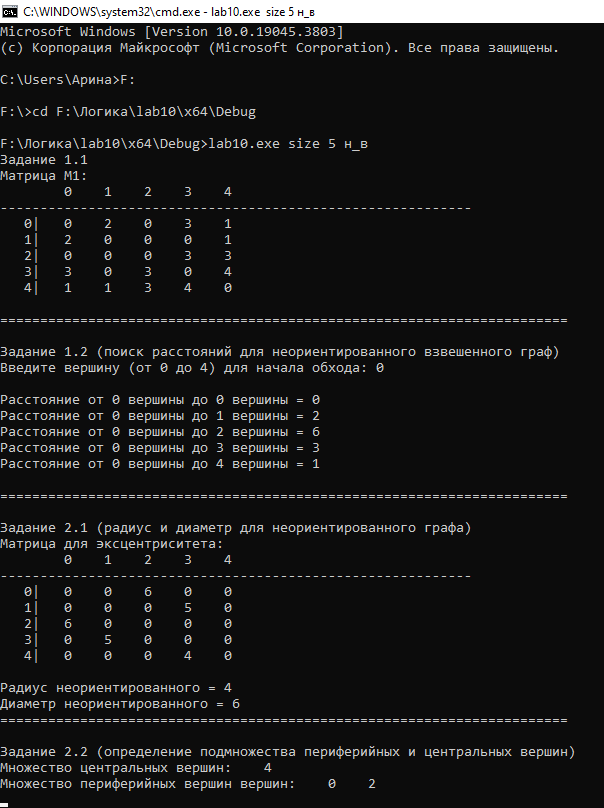


Рисунок 1 – Результат работы программы для неориентированного взвешенного графа (н\_в)

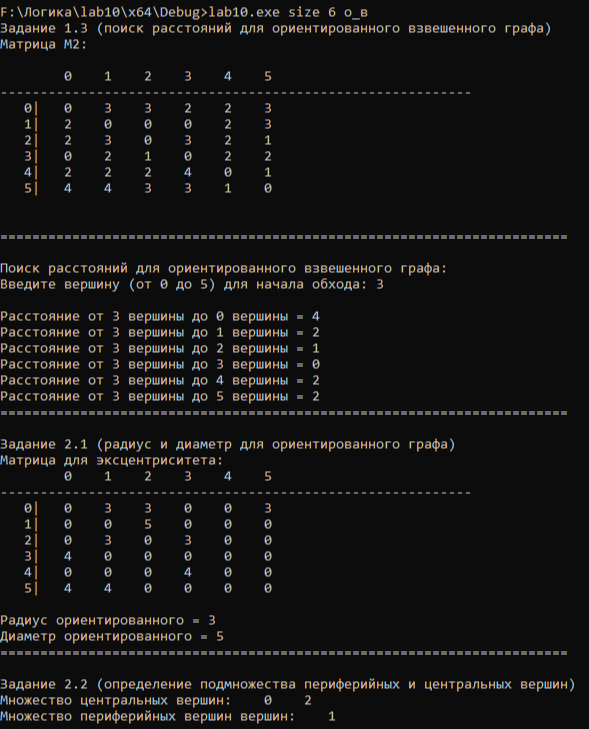


Рисунок 2 – Результат работы программы для ориентированного взвешенного графа (о\_в)

**Вывод**

В ходе выполнения лабораторной работы была разработана программа, выполняющая поиск расстояний во взвешенном графе. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок.

Получили опыт в создании проектов в среде Microsoft Visual Studio, приобрели навыки программирования алгоритмов.