Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Л и ОА в ИЗ»

на тему «Определение характеристик графов»

**Выполнил:**

студент группы 19ВВ3

Субботкин М.В.

**Принял:**

Митрохин М.А.

Пенза 2020

**Цель работы:**

Изучить способы определения характеристик графов.

**Ход работы:**

Эксцентриситет вершины – расстояние до наиболее удаленной вершины графа.

Максимальный эксцентриситет среди эксцентриситетов всех вершин

графа называется диаметром графа.

Вершина v i называется периферийной, если её эксцентриситет равен

диаметру графа e(v i ) = d(G).

Минимальный из эксцентриситетов вершин графа называется его

радиусом и обозначается через r(G).

Вершина v i называется центральной, если её эксцентриситет равен

радиусу графа e(v i ) = r(G).

Множество всех центральных вершин графа называется его центром.

Граф G может иметь единственную центральную вершину или несколько

центральных вершин.

Степенью вершины графа G называется число инцидентных ей ребер.

Степень вершины v i обозначается через deg(v i ).

Вершина v i со степенью 0 называется изолированной, со степенью 1 –

концевой.

Вершина графа, смежная с каждой другой его вершиной, называется

доминирующей.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <random>

#include <time.h>

#include <queue>

#include <locale.h>

using namespace std;

void output(int\*\* Array, int size) {

printf("\n");

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

{

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

{

printf("\t|%d|", Array[i][j]);

}

printf("\n");

}

printf("\n");

}

void GenWeighed(int\*\* Array, int size) {

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

{

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

{

int z = rand() % 100;

if (z > 80) {

Array[i][j] = rand() % size;

}

else {

Array[i][j] = 0;

}

Array[j][i] = Array[i][j];

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, size);

}

void ClearDistanceWeighed(int\* Dist, int size) {

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

Dist[i] = 1000;

}

}

void BFSWeighed(int\*\* mass, int v, int\* DIST, int size) {

std::queue <int> Q;

Q.push(v);

DIST[v] = 0;

while (!Q.empty()) {

v = Q.front();

Q.pop();

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

if (mass!= 0)

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

if (mass[v][i] > 0 && DIST[i] > DIST[v] + mass[v][i]) {

Q.push(i);

DIST[i] = DIST[v] + mass[v][i];

}

}

}

}

}

void BFSWeighed\_2(int\*\* mass, int v, int\* DIST, int size, int count) {

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

if (mass[v][j] != 0) {

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

if ((mass[k][j] != 0) && (DIST[k] == 1000)) {

DIST[k] = DIST[v] + mass[k][j];

}

}

}

}

}

void transfer(int\*\* mass, int\*\* mass\_inc, int size, int count) {

int index;

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

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

mass\_inc[i][j] = 0;

}

}

for (int i = index = 0; i < size; i++) {

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

if (mass[i][j] != 0) {

mass\_inc[i][index] = mass\_inc[j][index] = mass[i][j];

index++;

}

}

}

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

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

printf("\t|%d|", mass\_inc[i][j]);

}

printf("\n");

}

}

void operations(int\*\* mass, int\* eks, int size, int\* DIST, int R, int D, int count) {

printf("Матрица смежности:\n");

GenWeighed(mass, size);

ClearDistanceWeighed(DIST, size);

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

eks[i] = -1;

}

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

BFSWeighed(mass, i, DIST, size);

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

if (eks[i] < DIST[j] && DIST[j] != 1000) {

eks[i] = DIST[j];

}

}

if (eks[i] < R && eks[i] != 0) {

R = eks[i];

}

if (eks[i] > D) {

D = eks[i];

}

ClearDistanceWeighed(DIST, size);

}

printf("Эксцентриситеты -> ");

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

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

}

printf("\nРадиус рафа -> %d", R);

printf("\nДиаметр графа -> %d", D);

printf("\nЦентральные вершины графа -> ");

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

if (eks[i] == R) {

printf("%d ", i + 1);

}

}

printf("\nПереферийные вершины графа -> ");

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

if (eks[i] == D) {

printf("%d ", i + 1);

}

}

printf("\nИзолированные вершины графа -> ");

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

if (eks[i] == 0) {

printf("%d ", i + 1);

}

}

printf("\nКонцевые вершины графа -> ");

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

count = 0;

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

if (mass[i][j] != 0) {

count++;

}

}

if (count == 1) {

printf("%d ", i + 1);

}

}

bool tmp = false;

printf("\nДоминирующие вершины графа -> ");

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

count = 0;

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

if (mass[i][j] != 0) {

count++;

}

}

if (count == size - 1) {

printf("%d ", i + 1);

tmp = true;

}

}

printf("\n");

}

void ioperations(int\*\* mass\_inc, int\* eks, int size, int\* DIST, int R, int D, int count) {

ClearDistanceWeighed(DIST, size);

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

eks[i] = -1;

}

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

DIST[i] = 0;

BFSWeighed\_2(mass\_inc, i, DIST, size, count);

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

if ((eks[i] < DIST[k]) && (DIST[k] != 1000)) {

eks[i] = DIST[k];

}

}

if ((eks[i] < R) && (eks[i] != 0)) {

R = eks[i];

}

if (eks[i] > D) {

D = eks[i];

}

ClearDistanceWeighed(DIST, size);

}

printf("Эксцентриситеты матрицы инцедентности -> ");

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

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

}

printf("\nРадиус матрицы инцедентности -> %d ", R);

printf("\nДиаметр матрицы инцедентности -> %d ", D);

printf("\nЦентральные вершины матрицы инцедентности -> ");

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

if (eks[i] == R) {

printf("%d ", i + 1);

}

}

printf("\nПереферийные вершины матрицы инцедентости -> ");

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

if (eks[i] == D) {

printf("%d ", i + 1);

}

}

printf("\nИзолированные вершины матрицы инцедентности -> ");

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

if (eks[i] == 0) {

printf("%d ", i + 1);

}

}

printf("\nКонцевые вершины матрицы инцедентности -> ");

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

int index = 0;

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

if (mass\_inc[i][j] != 0) {

index++;

}

}

if (index == 1) {

printf("%d ", i + 1);

}

}

printf("\nДоминирующие веришны матрицы инцедентности -> ");

bool tmp = false;

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

int index = 0;

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

if (mass\_inc[i][j] != 0) {

index++;

}

}

if (index == count) {

printf("%d ", i + 1);

tmp = true;

}

}

if (tmp == false) {

printf("Нет доминирующих вершин");

}

}

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

int count = 0;

int N = 0;

int D = -1, R = 10000;

int\*\* mass = NULL;

int\*\* mass\_inc = NULL;

int\* DIST = NULL;

int\* eks = NULL;

printf("Введите количество вершин : ");

scanf("%d", &N);

DIST = (int\*)malloc(N \* sizeof(int));

eks = (int\*)malloc(N \* sizeof(int));

mass = (int\*\*)malloc(N \* sizeof(int\*));

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

{

mass[i] = (int\*)malloc(N \* sizeof(int));

}

//Матрица смежности//

operations(mass, eks, N, DIST, R, D, count);

//Матрица инцидентности//

D = -1, R = 10000; count = 0;

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

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

if (mass[i][j] != 0) {

count++;

}

}

}

mass\_inc = (int\*\*)malloc(N \* sizeof(int\*));

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

{

mass\_inc[i] = (int\*)malloc(count \* sizeof(int));

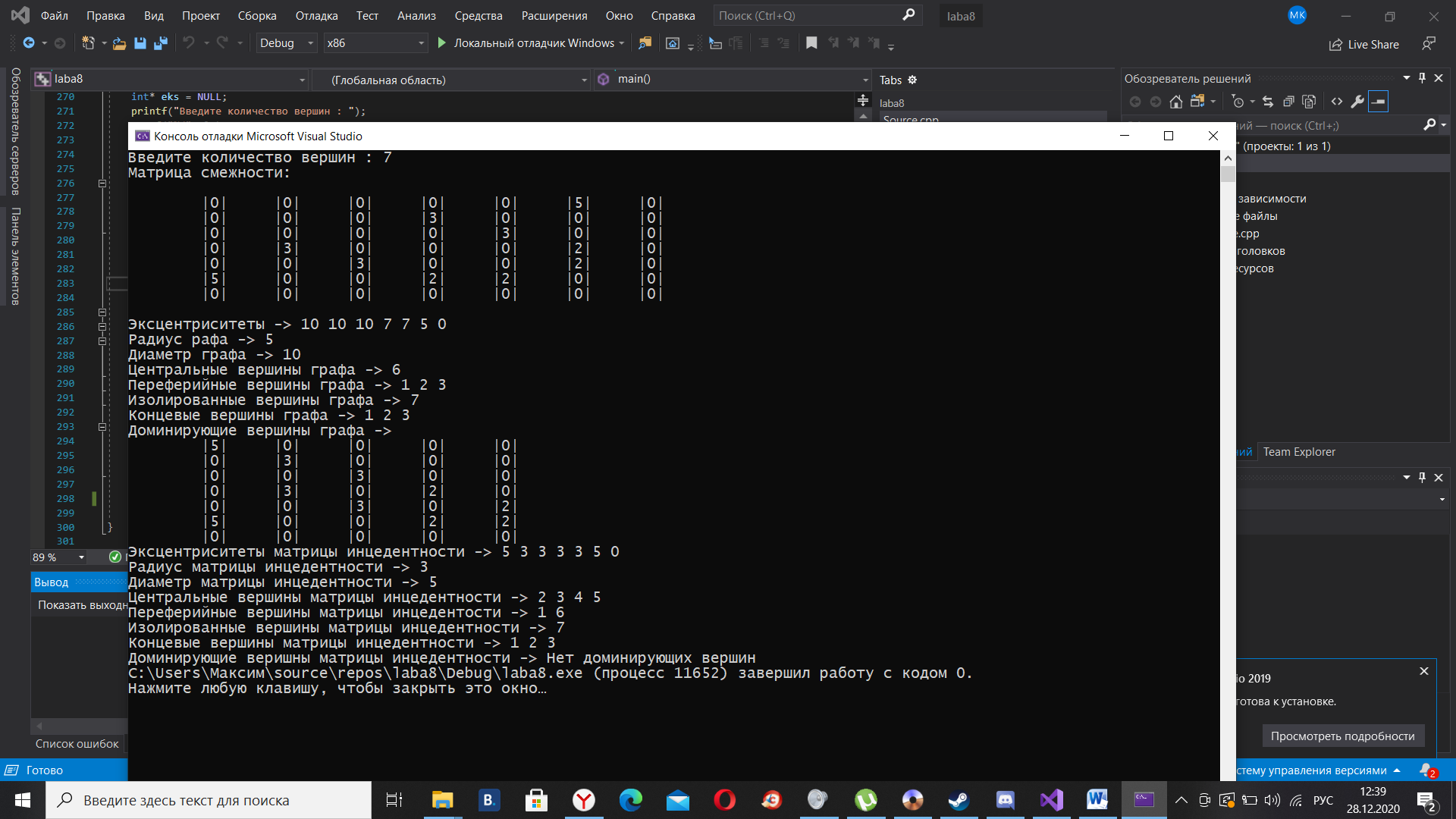
}

transfer(mass, mass\_inc, N, count);

ioperations(mass\_inc, eks, N, DIST, R, D, count);

}

### Результат работы программы:



### Выводы

В ходе выполнения лабораторной работы изучил способы определения характеристик графа.