Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

**по курсу «Логика и основы алгоритмизации в инженерных задачах»**

на тему «Определение характеристик графов»

Выполнили:

студент группы 20ВВ2

Александров В.С.

Кирюткин И.А.

Сафронов Д.В.

Принял:

к.т.н., доцент

Митрохин М.А.

**Пенза 2021**

**Цель работы:** Реализовать программу для нахождения характеристик графа.

**Лабораторные задания:**

Задание 1

Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного взвешенного графа G. Выведите матрицу

на экран.

2. Определите радиус и диаметр графа G, используя матрицу смежности

графа.

3. Определите подмножества периферийных и центральных вершин

графа G, используя матрицу смежности.

4. Найдите изолированные, концевые и доминирующие вершины.

Задание 2\*

1. Постройте для графа G матрицу инцидентности.

2. Определите радиус и диаметр графа G, используя матрицу

инцидентности графа.

3. Определите подмножества периферийных и центральных вершин

графа G, используя матрицу инцидентности.

**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <vector>

#include <queue> // очередь

#include <string>

using namespace std;

void BFS1(int \*\*mas, int \*DIST, int start)

{

queue<int> Queue;

int nodes[7]; // вершины графа

//for (int i = 0; i < 7; i++)

Queue.push(start); // помещаем в очередь первую вершину

DIST[start] = 0; // исходно все вершины равны 1000

while (!Queue.empty())

{ // пока очередь не пуста

start = Queue.front(); // извлекаем вершину

Queue.pop();

cout << start << endl; // выводим номер вершины

for (int j = 0; j < 7; j++)

{ // проверяем для нее все смежные вершины

if (mas[start][j] >0 && DIST[j] > DIST[start] + mas[start][j])

{ // если вершина смежная и не обнаружена

Queue.push(j); // добавляем ее в очередь

//nodes[j] = 1; // отмечаем вершину как обнаруженную

DIST[j] = DIST[start] + mas[start][j];

}

}

}

}

int param() {

int start, size=6, \*\*graph;

printf("Ведите размер графа");

scanf\_s("%d", &size);

graph = (int\*\*)malloc(size \* sizeof(int));

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

{

graph[i] = (int\*)malloc(size \* sizeof(int));

}

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

{

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

{

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

graph[j][i] = graph[i][j];

}

}

/\*int graph[6][6] = { // матрица смежности

{ 0, 2, 3, 0, 0, 1 },

{ 2, 0, 0, 5, 0, 0 },

{ 3, 0, 0, 0, 1, 0 },

{ 0, 5, 0, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0 },

{ 1, 0, 0, 0, 0, 0 } };\*/

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

{

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

{

if (i == j)

graph[i][j] = 0;

printf("%2d", graph[i][j]);

}

printf("\n");

}

int \*max;// [6] = { 0, 0, 0, 0, 0, 0 };

int \*min;// [6] = { 99999, 99999, 99999, 99999, 99999, 99999 };

max = (int\*)malloc(size \* sizeof(int));

min = (int\*)malloc(size \* sizeof(int));

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

{

min[i] = 99999;

max[i] = 0;

}

//int min[6] = { 99999, 99999, 99999, 99999, 99999, 99999 };

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

//cout << "Введите начальную вершину \n";

// cin >> start;

start = j;

int \*DIST;

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

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

DIST[i] = 1000;

}

BFS1(graph, DIST, start);

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

printf("рассояние от %d до %d = %d\n", start, i, DIST[i]);

}

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

{

if (DIST[i] > max[j])

{

max[j] = DIST[i];

}

}

printf("max рассояние %d\n", max[j]);

}

int diam = 0, rad=1000;

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

{

if (max[i] < rad)

{

rad = max[i];

}

if (max[i] > diam)

{

diam = max[i] ;

}

}

printf("\nДиаметр графа = %d \nРадиус графа %d\n", diam, rad);

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

{

if (max[i] == diam)

{

printf("периферийная вершина: = %1d\n", i);

}

if (max[i] == rad)

{

printf("центральная вершина: = %1d\n", i);

}

}

// степень вершины

int \*step;// 6] = { 0, 0, 0, 0, 0, 0 };

step = (int\*)malloc(size \* sizeof(int));

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

{

step[i] = 0;

}

//int step\_dop[6] = { 0, 0, 0, 0, 0, 0 };

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

{

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

{

if (graph[i][j]>0)

{

step[i] ++;;

}

}

printf("степень вершины %d: = %1d\n", i, step[i]);

}

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

{

if (step[i]==0)

{

printf("вершинa %d изолированая\n", i);

}

if (step[i] == 1)

{

printf("вершинa %d концевая\n", i);

}

if (step[i] == size)

{

printf("вершинa %d доминирующая\n", i);

}

}

return 0;

}

void BFS\_inte(int mas[6][5], int DIST[6], int start)

{

queue<int> Queue;

//int nodes[6]; // вершины графа

//for (int i = 0; i < 7; i++)

Queue.push(start); // помещаем в очередь первую вершину

DIST[start] = 0; // исходно все вершины равны 1000

while (!Queue.empty())

{ // пока очередь не пуста

start = Queue.front(); // извлекаем вершину

Queue.pop();

cout << start << endl; // выводим номер вершины

for (int j = 0; j < 5; j++)

{ // проверяем для нее все смежные вершины

if (mas[start][j] > 0) {

for (int k = 0; k < 6; k++) {

if (mas[k][j] > 0 && DIST[k] > DIST[start] + mas[start][j])

{ // если вершина смежная и не обнаружена

Queue.push(k); // добавляем ее в очередь

//nodes[j] = 1; // отмечаем вершину как обнаруженную

DIST[k] = DIST[start] + mas[start][j];

}

}

}

}

}

}

// строки - N

int param2() {

int start, size = 6;

// 0 1 2 3 4

int graph[6][5] = { // матрица инцидентности

{2, 3, 0, 0, 1},//0

{2, 0, 5, 0, 0},//1

{0, 3, 0, 1, 0},//2

{0, 0, 5, 0, 0},//3

{0, 0, 0, 1, 0},//4

{0, 0, 0, 0, 1},//5

};

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

{

for (int j = 0; j < 5; j++)

{

printf("%2d", graph[i][j]);

}

printf("\n");

}

int max[6] = { 0, 0, 0, 0, 0, 0 };

int min[6] = { 99999, 99999, 99999, 99999, 99999, 99999 };

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

// cout << "Введите начальную вершину \n";

// cin >> start;

start = j;

int DIST[6];

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

DIST[i] = 1000;

}

BFS\_inte(graph, DIST, start);

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

printf("рассояние от %d до %d = %d\n", start, i, DIST[i]);

}

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

{

if (DIST[i] > max[j])

{

max[j] = DIST[i];

}

}

printf("max рассояние %d\n", max[j]);

}

int diam = 0, rad = 1000;

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

{

if (max[i] < rad)

{

rad = max[i];

}

if (max[i] > diam)

{

diam = max[i];

}

}

printf("\nДиаметр графа = %d \nРадиус графа %d\n", diam, rad);

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

{

if (max[i] == diam)

{

printf("периферийная вершина: = %1d\n", i);

}

if (max[i] == rad)

{

printf("центральная вершина: = %1d\n", i);

}

}

return 0;

}

int intend() {

return 0;

}

int main() {

setlocale(LC\_ALL, "Rus");

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

param();

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

param2();

}

**Псевдокод программы:**

Вход: G – неориентированный граф, start, max\_size, max, min

Вывод: граф G, start, i, Dist, max, diam, rad, i

Алгоритм Dopy:

Для j От 0 До max\_size

start = j

Для i От 0 До max\_size

DIST[i] = 1000

помещаем в очередь первую вершину

исходно все вершины равны 1000

пока очередь не пуста

извлекаем вершину

выводим номер вершины

Для j От 0 До max\_size

проверяем для нее все смежные вершины

Если mas[start][j] > 0

Для k От 0 До max\_size

Если mas[k][j] > 0 && DIST[k] > DIST[start] + mas[start][j]

если вершина смежная и не обнаружена

добавляем ее в очередь

отмечаем вершину как обнаруженную

DIST[k] = DIST[start] + mas[start][j]

Для i От 0 До max\_size

Если (DIST[i] > max[j])

max[j] = DIST[i]

diam = 0, rad = 1000;

Для i От 0 До max\_size

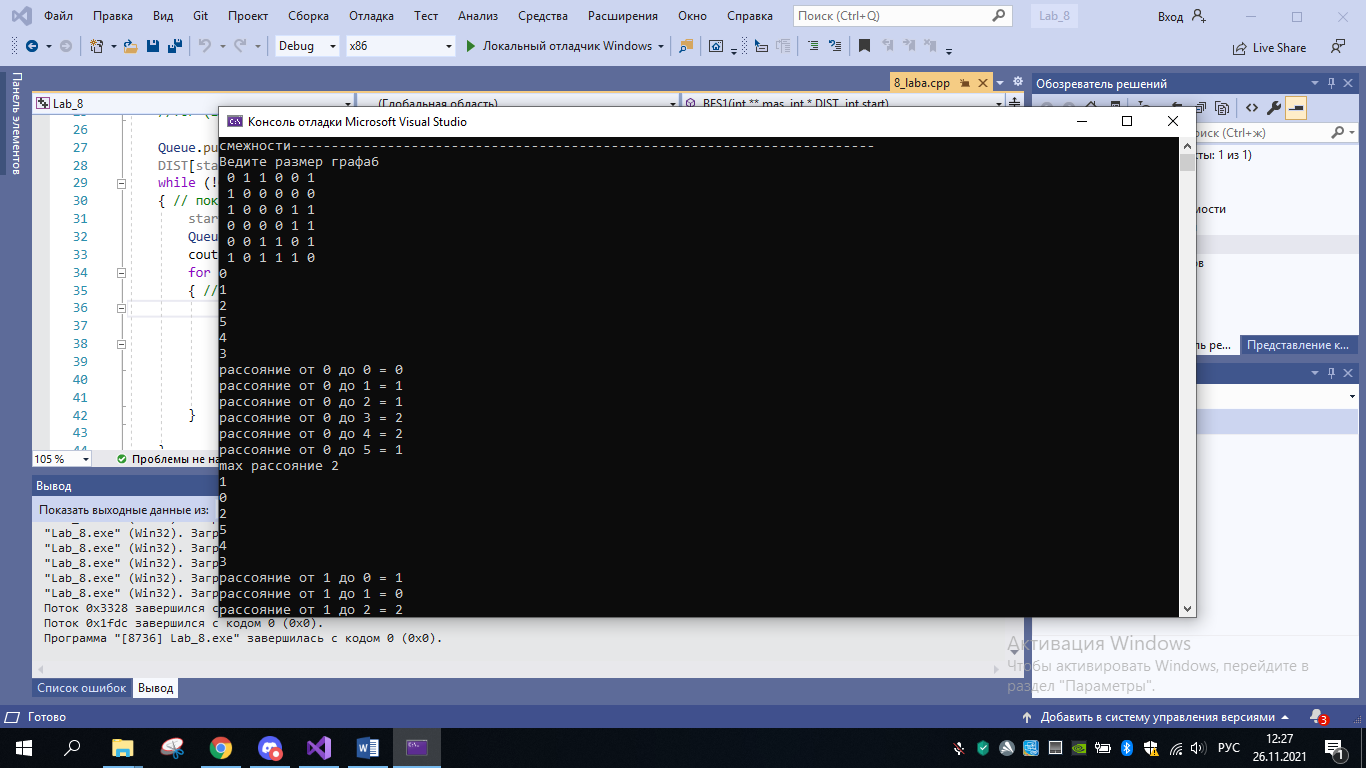
Если (max[i] < rad)

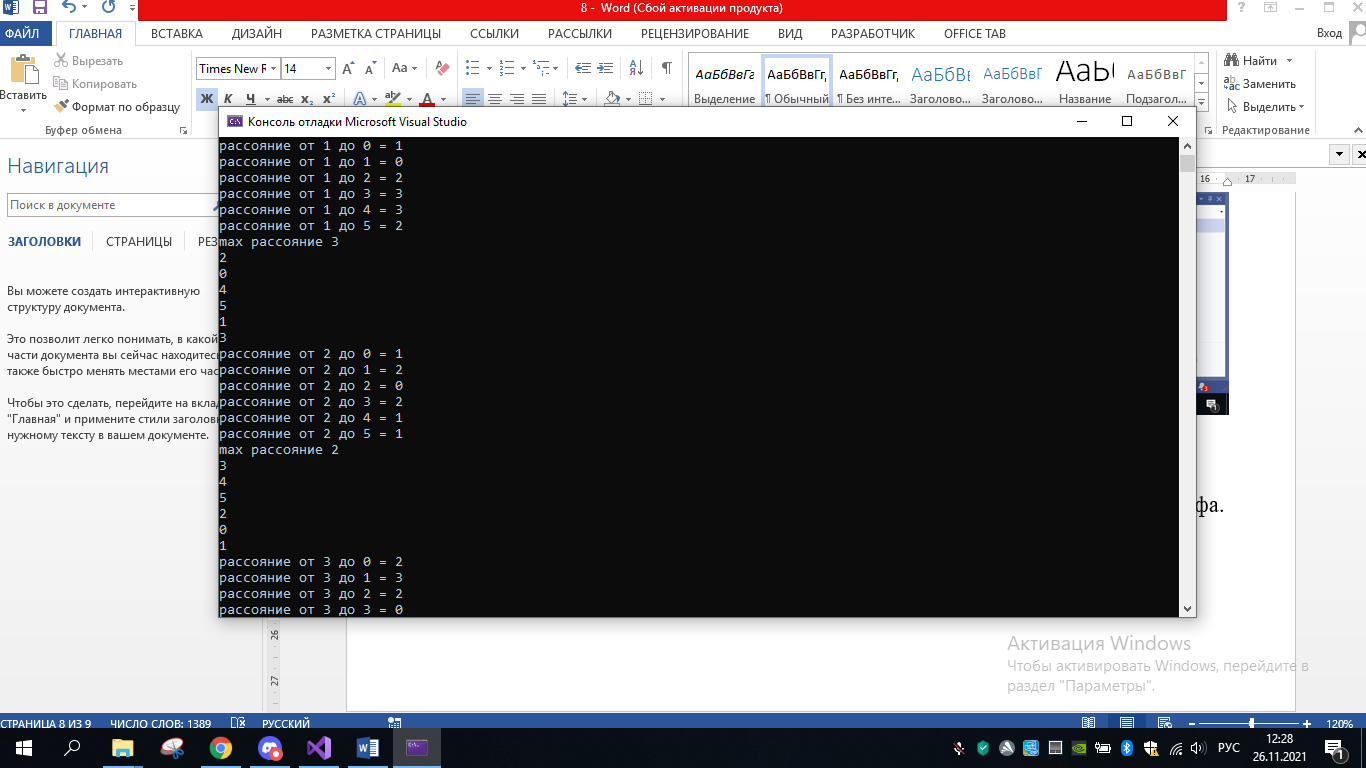
rad = max[i]

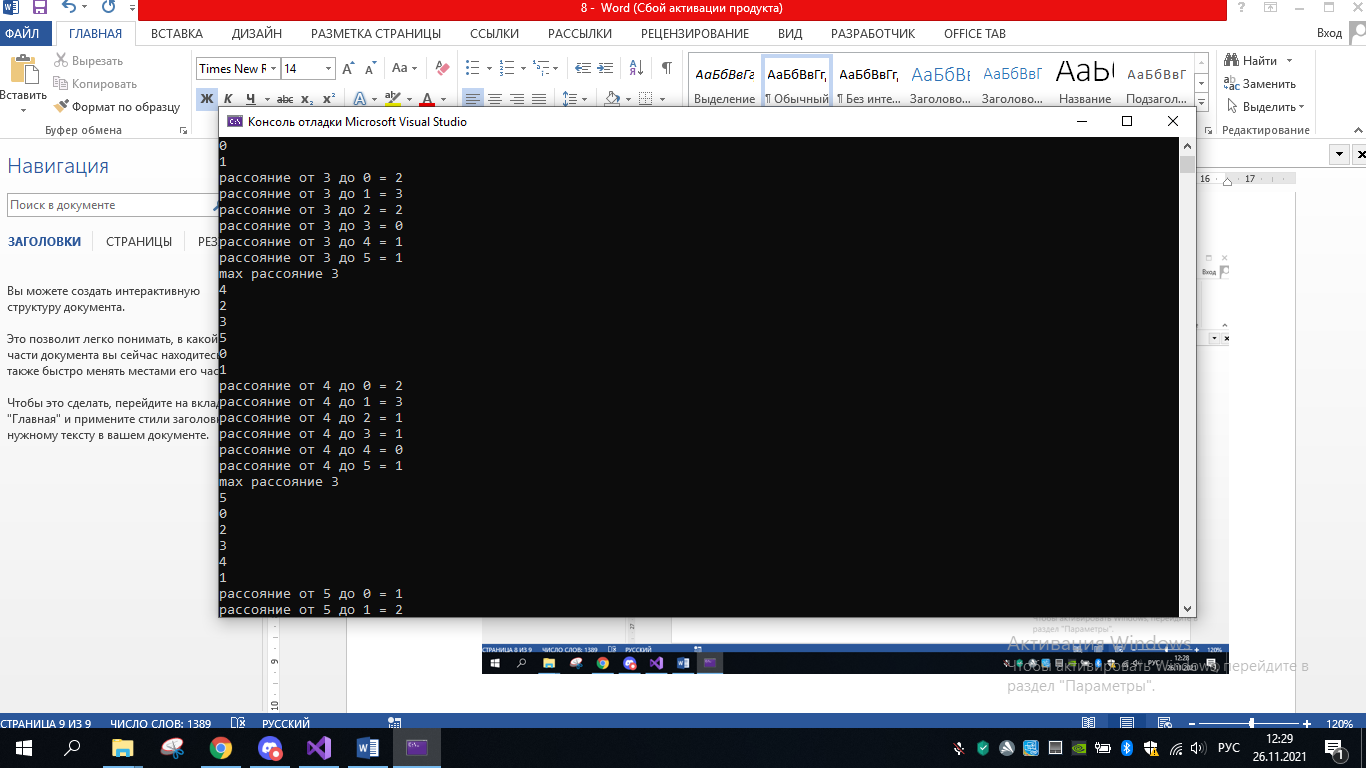
Если (max[i] > diam)

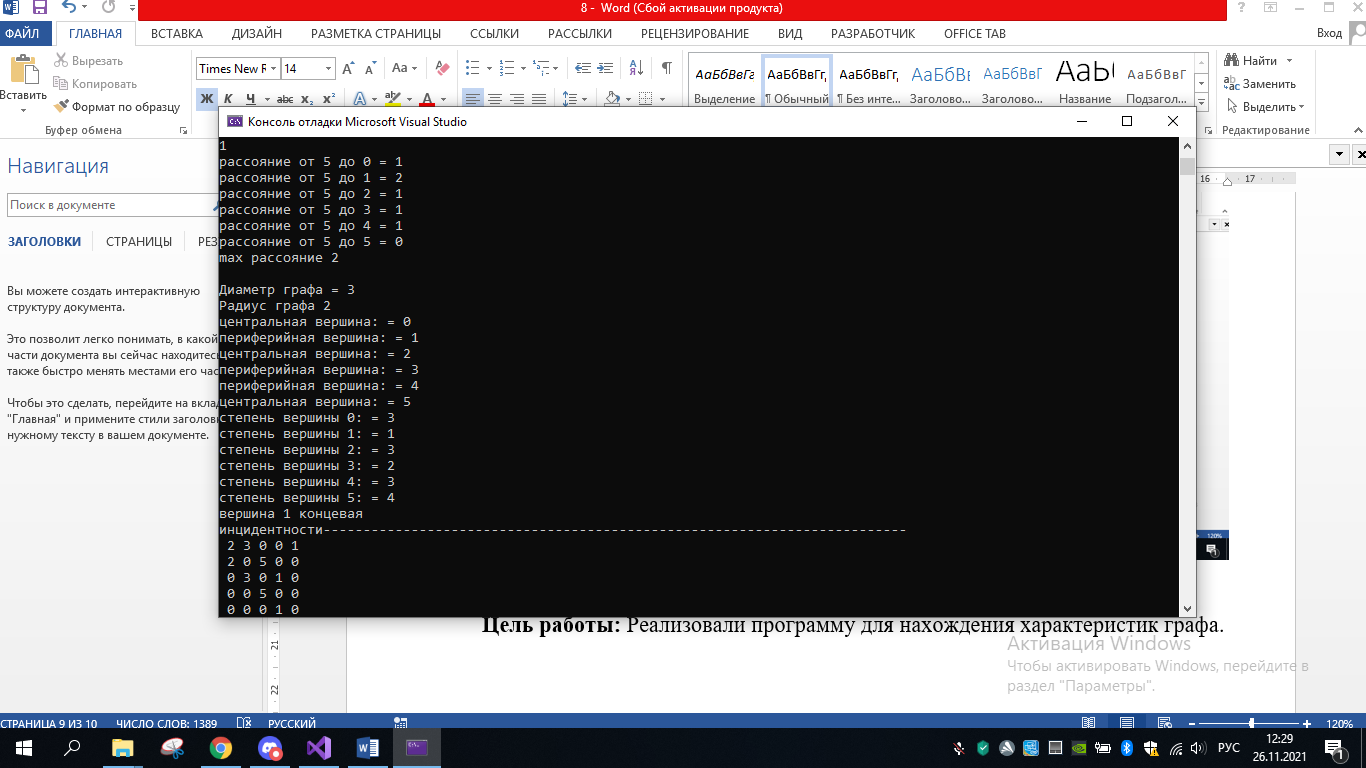
diam = max[i]

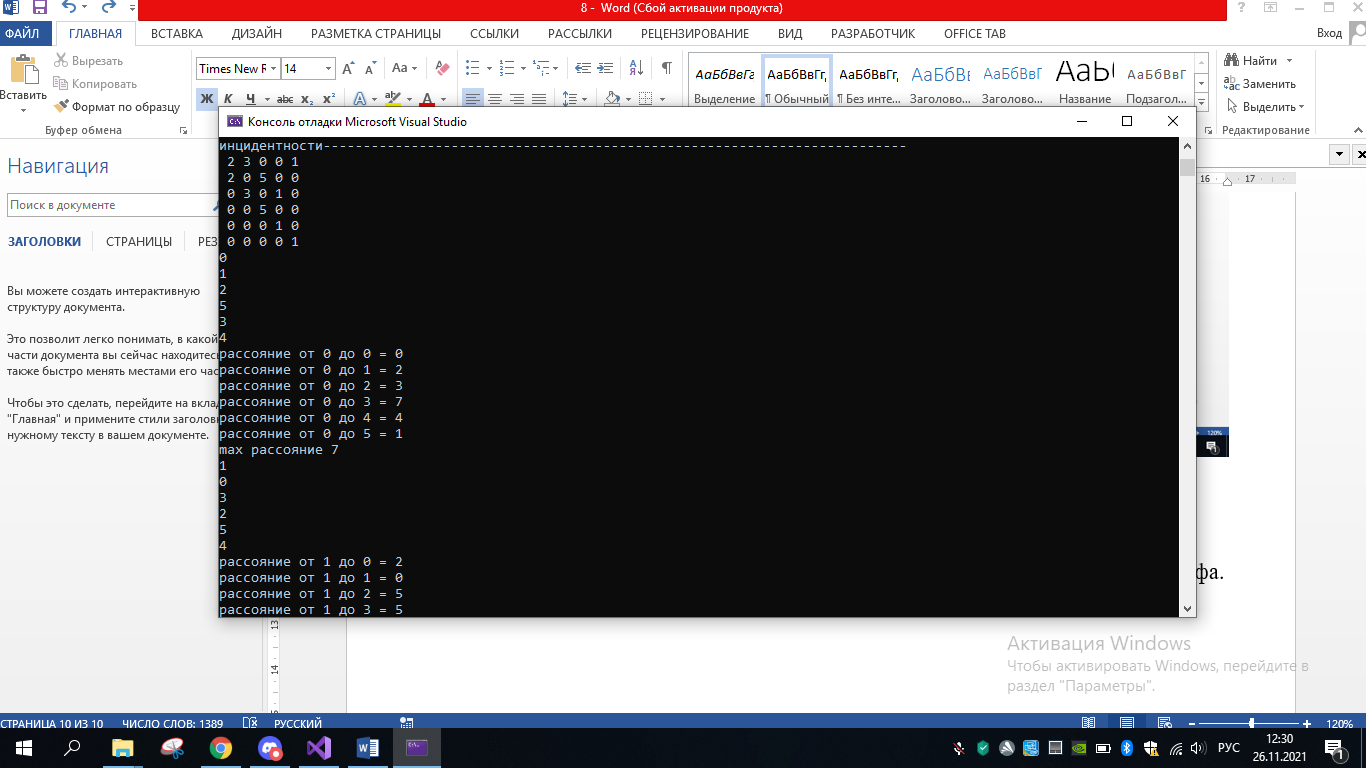
**Результаты работы программы:**

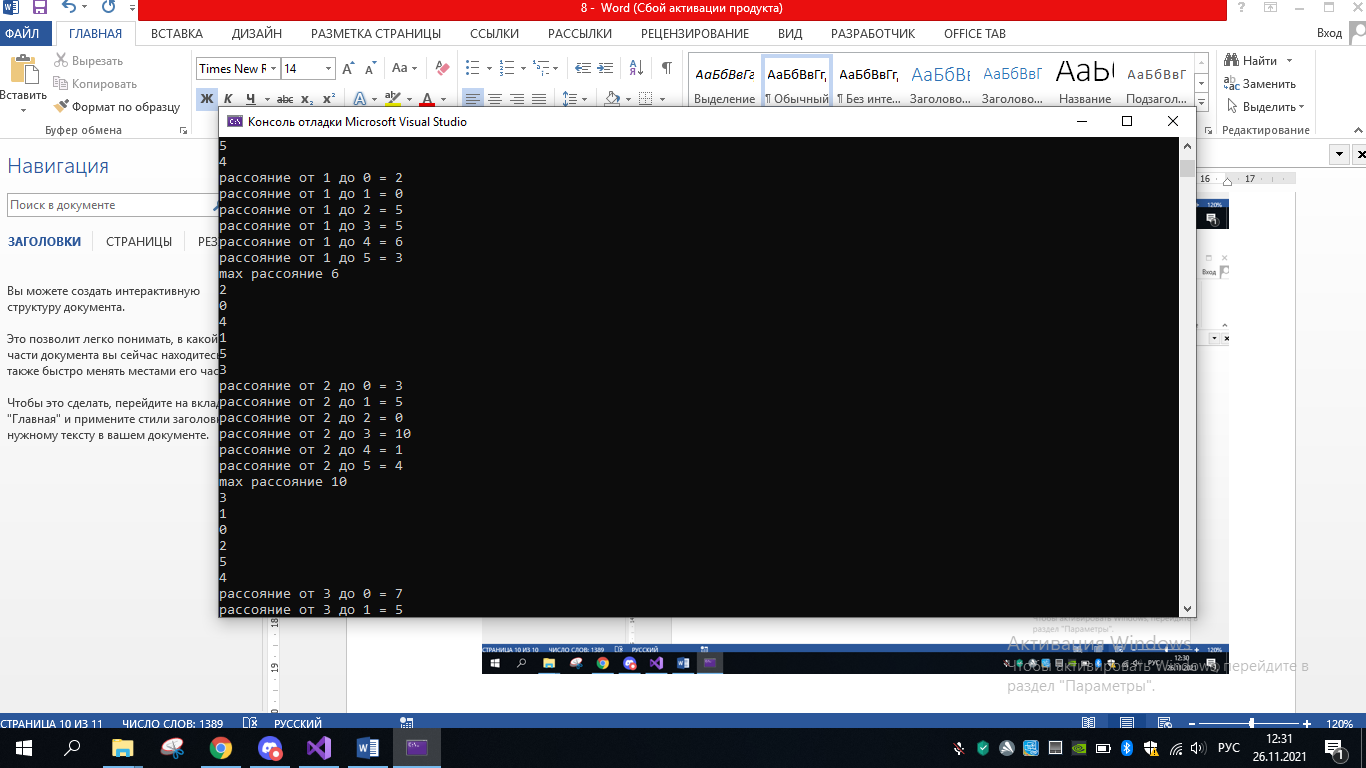


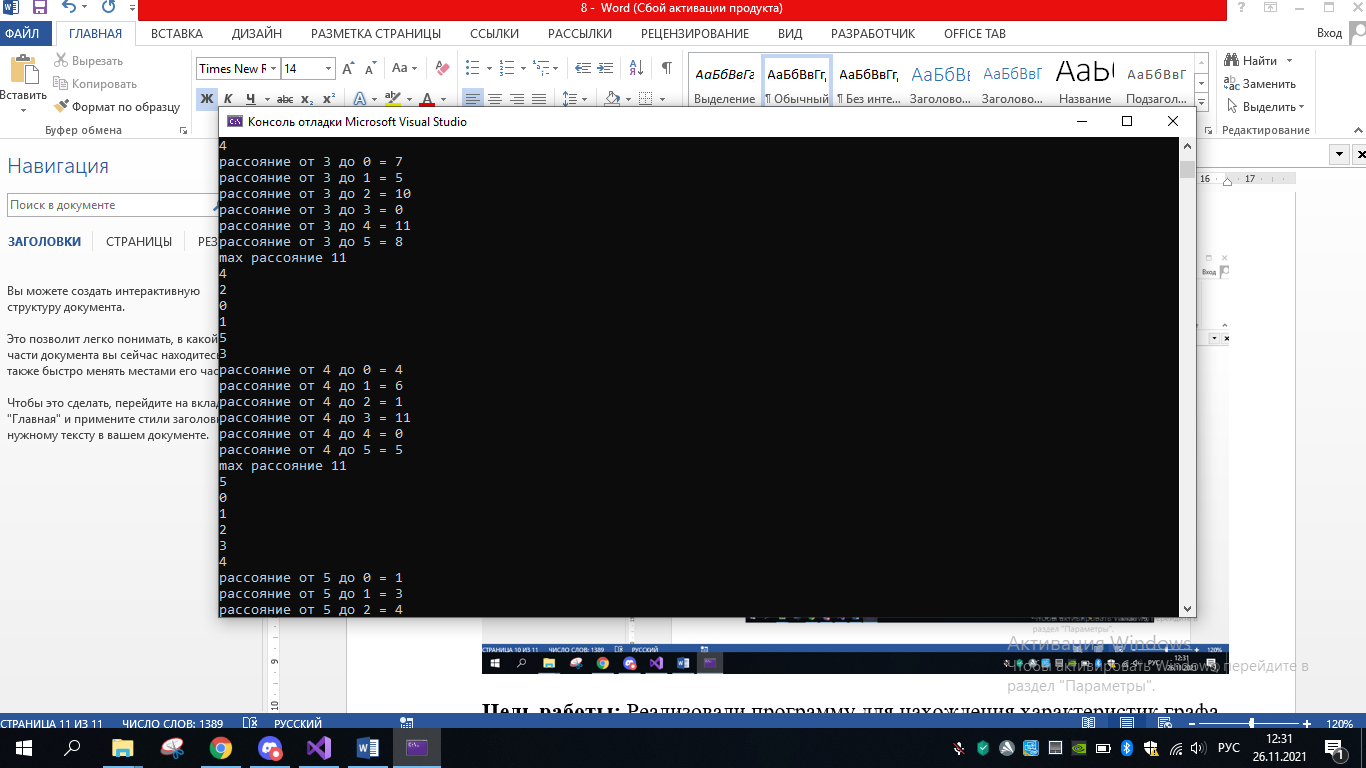


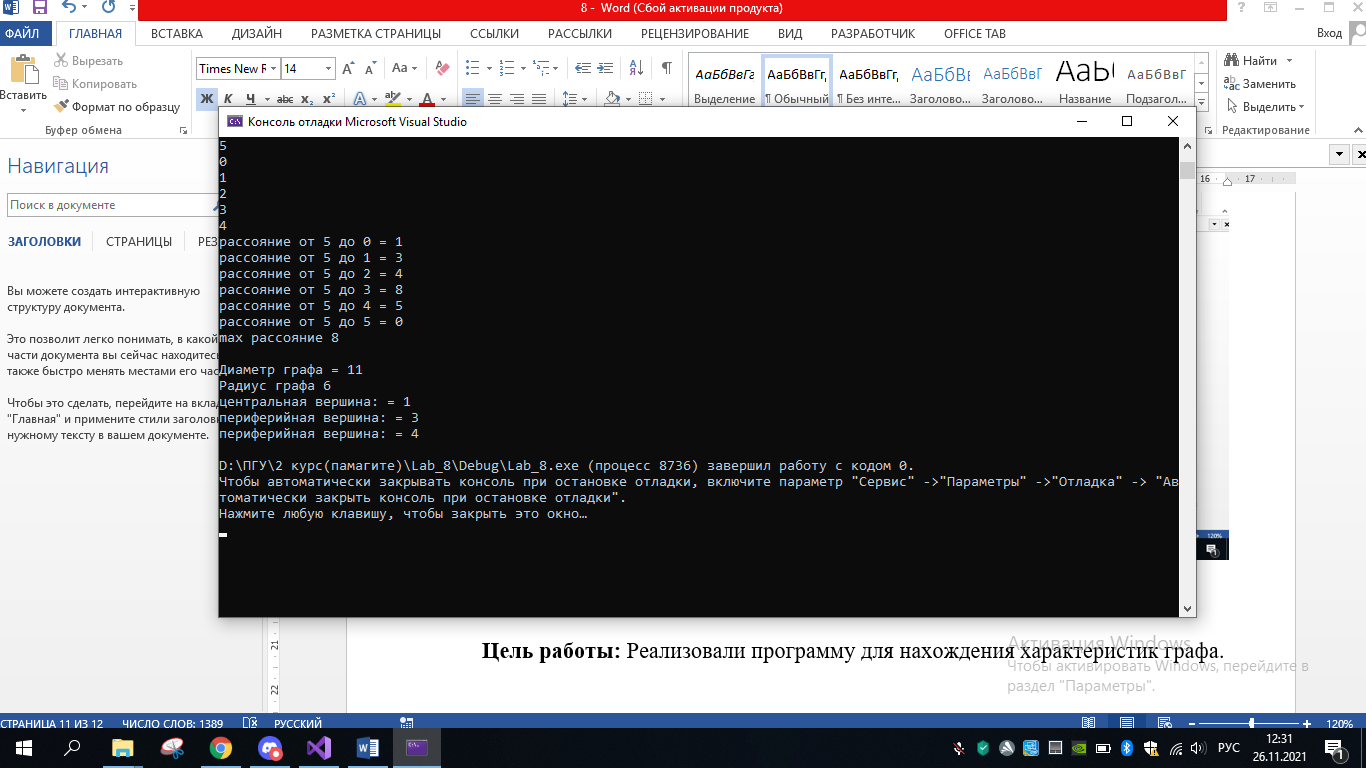












**Цель работы:** Реализовали программу для нахождения характеристик графа.