Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

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

на тему «Поиск расстояний в графе»

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:** Составить алгоритм поиска расстояний в графах.

**Лабораторные задания:**

Задание 1

1.Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного 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.\* Реализуйте процедуру поиска расстояний для графа, представленного

списками смежности.

Задание 2\*

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.

2. Реализуйте процедуру поиска расстояний на основе обхода в глубину

для графа, представленного списками смежности.

3. Оцените время работы реализаций алгоритмов поиска расстояний на

основе обхода в глубину и обхода в ширину для графов разных порядков.

**Листинг файла osnova:**

#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

using namespace std;

int main1();

typedef struct queue1 {

int info;

struct queue1\* next;

}node;

void createqueue(node\*\* front1, node\*\* rear1) {

\*front1 = \*rear1 = NULL;

}

void pushqueue(node\*\* front1, node\*\* rear1, int item) {

node\* ptr;

ptr = (node\*)malloc(sizeof(node));

ptr->info = item;

ptr->next = NULL;

if ((\*front1) == NULL)

\*front1 = \*rear1 = ptr;

else {

(\*rear1)->next = ptr;

\*rear1 = ptr;

}

}

int popqueue(node\*\* front1) {

int item;

node\* ptr;

ptr = \*front1;

item = ptr->info;

\*front1 = ptr->next;

return item;

}

int peekqueue(node\* front1) {

return front1->info;

}

bool isEmpty(node\* front1) {

if (front1 == NULL)

return true;

return false;

}

void BFS1(int mas[7][7], int DIST[7], 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 -1

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] == 1 && DIST[j]==-1)

{ // если вершина смежная и не обнаружена

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

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

DIST[j] = DIST[start] +1;

}

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

printf("Через BFS\n");

int graph[7][7] =

{

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

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

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

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

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

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

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

int start;

cout << "Введите начальную вершину \n";

cin >> start;

int DIST[7];

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

DIST[i] = -1;

}

cout << "обход в ширину\n";

BFS1(graph, DIST, start);

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

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

}

main1();

return 0;

}

int DSFD\_Recr(int st, int\*\* b, int num\_of\_elem, int\* visited) {

int i;

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

for (i = 0; i < num\_of\_elem; i++)

{

if (b[st][i] == 1 and visited[i] == -1)

{

visited[i] = visited[st] + 1;

DSFD\_Recr(i, b, num\_of\_elem, visited);

}

}

return 0;

}

int main1() {

printf("\nЧерез DFS\n");

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int\*\* a, max\_size;

printf("Ведите размер графа");

scanf\_s("%d", &max\_size);

a = (int\*\*)malloc(max\_size \* sizeof(int));

for (int i = 0; i < max\_size; i++)

{

a[i] = (int\*)malloc(max\_size \* sizeof(int));

}

for (int i = 0; i < max\_size; i++)

{

for (int j = i + 1; j < max\_size; j++)

{

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

a[j][i] = a[i][j];

}

}

int DIST[10];

for (int i = 0; i < max\_size; i++)

{

DIST[i] = -1;

for (int j = 0; j < max\_size; j++)

{

if (i == j)

a[i][j] = 0;

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

}

printf("\n");

}

printf("\nПоиск расстояний обходомм в глубину \ncтартовая вершина - ");

int start;

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

DIST[start - 1] = 0;

DSFD\_Recr(start - 1, a, max\_size, DIST);

// printf("\nРасстояние из первой:\n");

for (int i = 0; i < max\_size; i++) {

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

DIST[i] = -1;

}

return 0;

}

**Листинг файла bsf\_spiski:**

#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 <vector>

#include <queue> // очередь

#define SIZE 40

using namespace std;

struct queue\_str {

int items[SIZE];

int front;

int rear;

};

struct queue\_str\* createQueue();

void enqueue(struct queue\_str\* q, int);

int dequeue(struct queue\_str\* q);

void display(struct queue\_str\* q);

int isEmpty(struct queue\_str\* q);

void printQueue(struct queue\_str\* q);

void BFS1(int\*\* graph);

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph

{

int numVertices;

int\* visited;

struct node\*\* adjLists; // we need int\*\* to store a two dimensional array. Similary, we need struct node\*\* to store an array of Linked lists

};

struct Graph\* createGraph(int);

void addEdge(struct Graph\*, int, int);

void printGraph(struct Graph\*);

void bfs(struct Graph\* graph, int startVertex, int DIST[5]);

int main()

{

setlocale(LC\_ALL, "Rus");

struct Graph\* graph = createGraph(5);

addEdge(graph, 0, 1);

addEdge(graph, 2, 0);

addEdge(graph, 1, 3);

addEdge(graph, 2, 1);

//addEdge(graph, 1, 4);

addEdge(graph, 3, 4);

int start, end1;

cout << "\nСтартовая вершина "; cin >> start;

cout << "\n поиск на основе списокв, в в ширину ";

int DIST[5];

for (int i = 0; i < 5; i++) {

DIST[i] = -1;

}

bfs(graph, start, DIST);

for (int i = 0; i < 5; i++) {

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

//printf("Посетил %d\n", DIST[i]);

}

printGraph(graph);

return 0;

}

void bfs(struct Graph\* graph, int startVertex, int DIST[5]) {

queue<int> Queue;

int ii = 0;

int search[100];

graph->visited[startVertex] = 1;

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

DIST[startVertex] = 0;

while (!Queue.empty()) {

int currentVertex = Queue.front(); // извлекаем вершину

Queue.pop();

//printf("Посетил %d\n", currentVertex);

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0 && DIST[adjVertex] == -1) {

Queue.push(adjVertex);

DIST[adjVertex] = DIST[currentVertex] + 1;

}

temp = temp->next;

}

}

}

struct node\* createNode(int v)

{

struct node\* newNode = (struct node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (struct node\*\*)malloc(vertices \* sizeof(struct node\*));

graph->visited = (int\*)malloc(vertices \* sizeof(int));

int i;

for (i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph)

{

int v;

for (v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\n Список смежности вершины %d\n ", v);

while (temp)

{

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

**Листинг файла dfs\_spiski:**

#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 <stack>

#include <queue>

using namespace std;

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph

{

int numVertices;

int\* visited;

struct node\*\* adjLists; // we need int\*\* to store a two dimensional array. Similary, we need struct node\*\* to store an array of Linked lists

};

struct Graph\* createGraph(int);

void addEdge(struct Graph\*, int, int);

void printGraph(struct Graph\*);

void DFS(struct Graph\* graph, int vertex, int\* visited);

int main()

{

setlocale(LC\_ALL, "Rus");

int versh, \* a, i, ver, conect;

//printf("Введите количество вершин: ");

//scanf("%d", &versh);

// a = (int\*)malloc(versh \* sizeof(int));

struct Graph\* graph = createGraph(5);

addEdge(graph, 0, 1);

addEdge(graph, 2, 0);

addEdge(graph, 0, 3);

addEdge(graph, 2, 1);

addEdge(graph, 2, 4);

printGraph(graph);

int DIST[5];

for (int i = 0; i < 5; i++) {

DIST[i] = -1;

}

cout << "обход в ширину\n";

DFS(graph, 4, DIST);

for (int i = 0; i < 5; i++) {

printf("рассояние от %d до %d = %d\n", 4, i, DIST[i]);

}

return 0;

}

void DFS(struct Graph\* graph, int vertex, int\* visited) {

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = graph->adjLists[vertex];

graph->visited[vertex] = 1;

//visited[vertex] +=1;

printf(" % d ", vertex);

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (visited[connectedVertex] == -1) {

visited[connectedVertex] = visited[vertex] + 1;

DFS(graph, connectedVertex, visited);

}

temp = temp->next;

}

}

struct node\* createNode(int v)

{

struct node\* newNode = (struct node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

//graph->adjLists = (struct Graph\*)malloc(vertices \* sizeof(struct node\*));

graph->adjLists = (struct node\*\*)malloc(vertices \* sizeof(struct node\*));

graph->visited = (int\*)malloc(vertices \* sizeof(int));

int i;

for (i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph)

{

int v;

for (v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\n Список смежности вершины %d\n ", v);

while (temp)

{

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

**Псевдокод:**

Вход: info, ptr, queue, mas[7][7], DIST[7], start

Вывод: DIST[i]

\*front1 = \*rear1 = NULL

Если (\*front1) = NULL

\*front1 = \*rear1 = ptr

Иначе

\*rear1 = ptr

ptr = \*front1

item = ptr

\*front1 = ptr

return front1

Если front1 = NULL

return false

добавляем в очередь

вершины графа

Для i от 0 До 7

помещаем в очередь первую вершину

исходно все вершины равны -1

пока очередь не пуста

извлекаем вершину

выводим номер вершины

Для от 0 до 7

проверяем для нее все смежные вершины

Если mas[start][j] = 1 И DIST[j]=-1

если вершина смежная и не обнаружена

добавляем ее в очередь

отмечаем вершину как обнаруженную

DIST[j] = DIST[start] +1

Для int i От 0 До 7

DIST[i] = -1

Для i от 0 до 7

Для i от 0 до num\_of\_elem

Если b[st][i] = 1 И visited[i] = -1

visited[i] = visited[st] + 1

Для i От 0 До max\_size

Выделяем память для a[i]

Для i от 0 до max\_size

Для j от i + 1 до max\_size

a[i][j] = rand() % 2

a[j][i] = a[i][j];

DIST[10]

Для I от 0 до max\_size

DIST[i] = -1

Для j от 0 до max\_size

Если i = j

a[i][j] = 0

Для i от 0 до max\_size

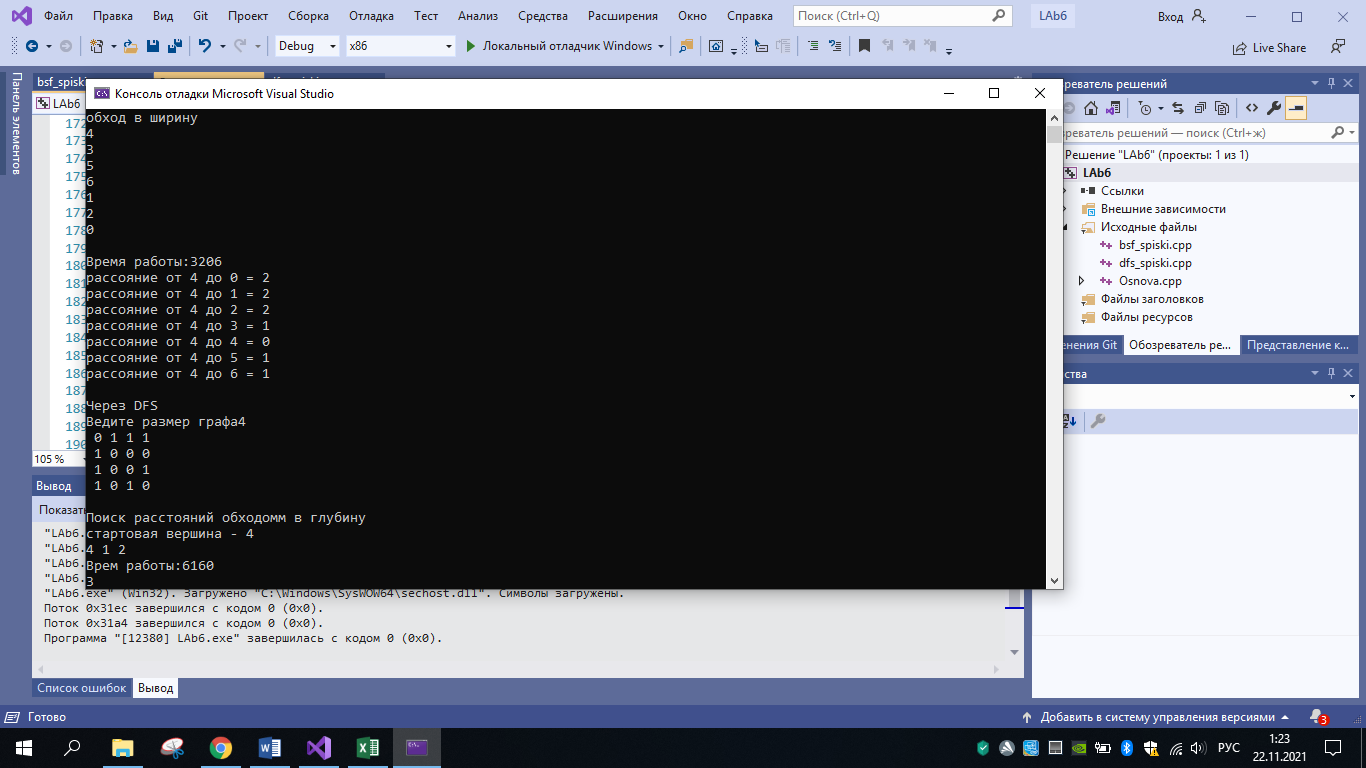
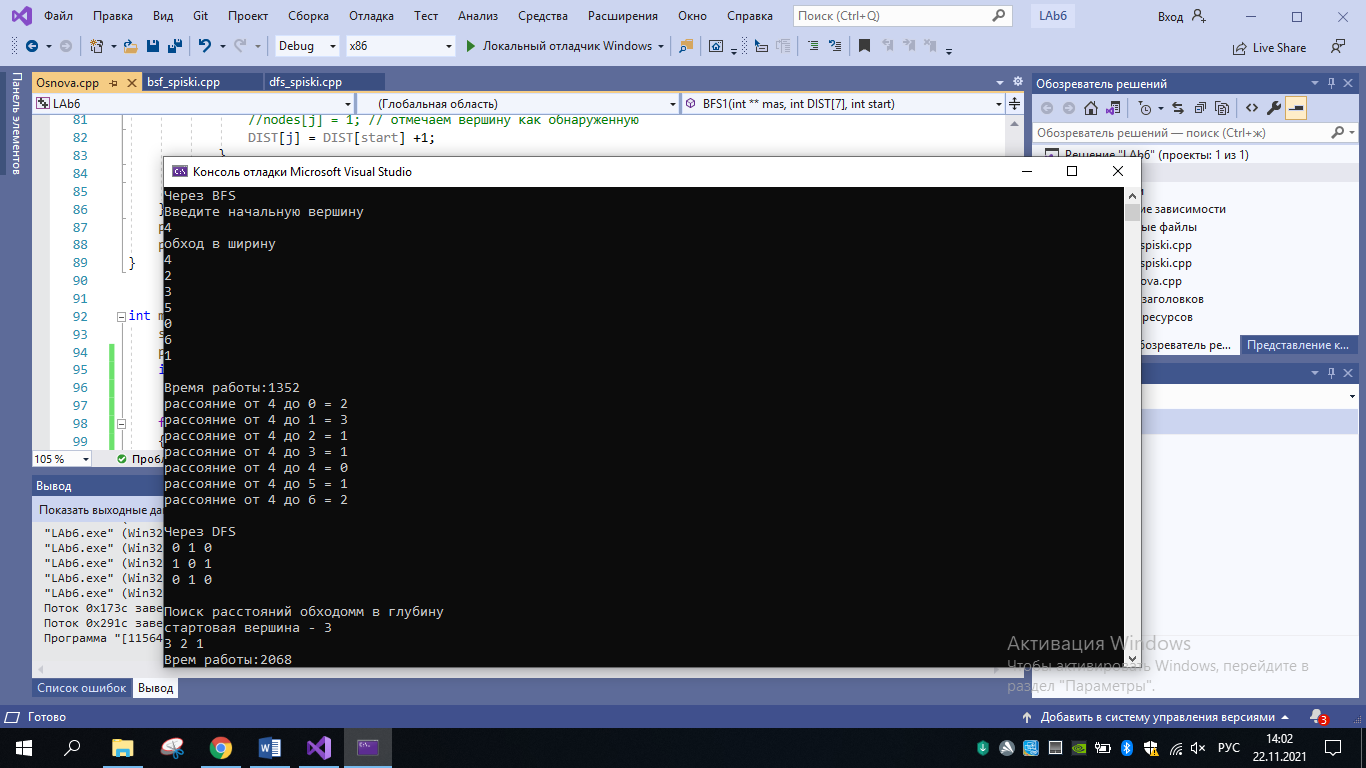
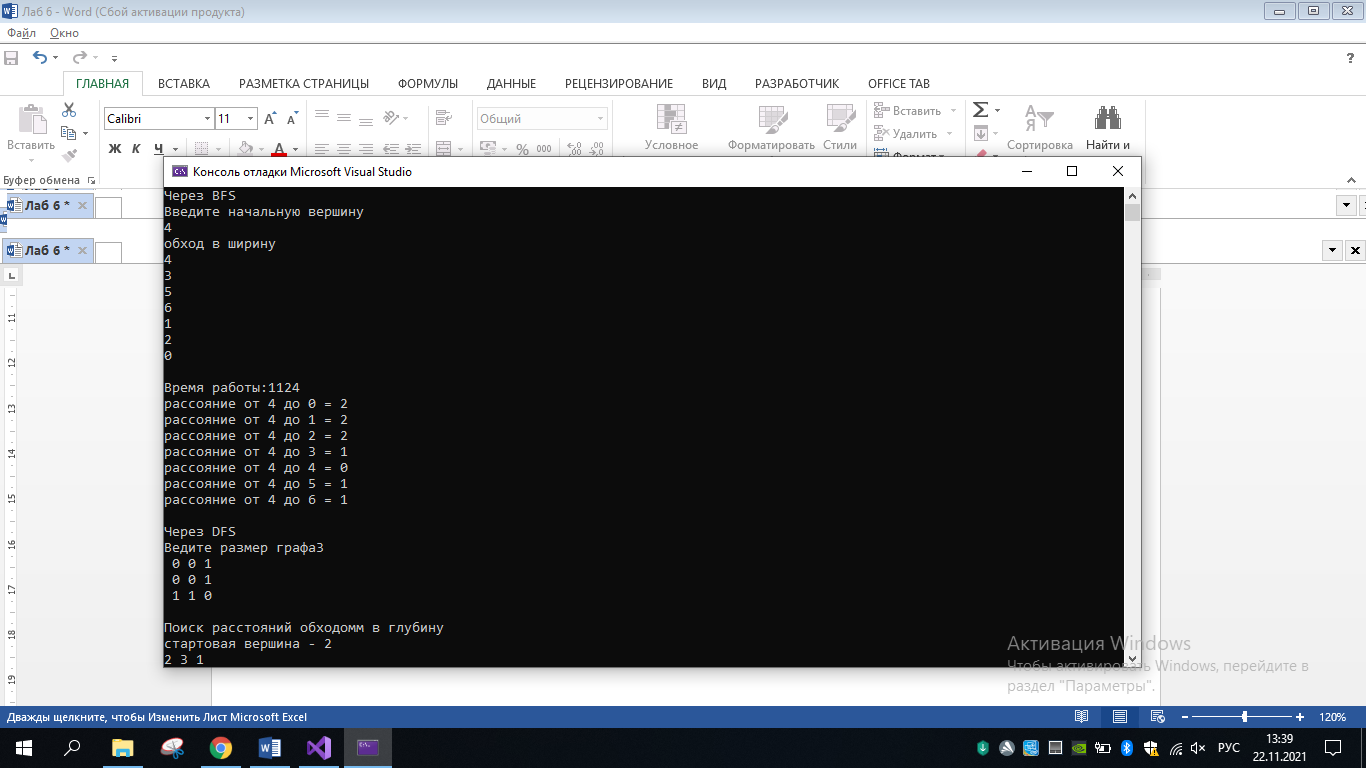
DIST[i] = -1

**Скорость работы bfs и dfs:**

****

Время работы функции bfs примерно в 2 раза больше, чем dfs.

**Результаты работы программы:**



**Вывод:** Составили алгоритм поиска расстояний в графах и сравнили время работы функций bfs с dfs.