|  |
| --- |
| 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» |
|  |
|  |
| Выполнила студентка группы 19ВВ2:  Отставнов А.М.  Принял:  Митрохин М. А  Юрова О.В. |
| Пенза  2020 |

### **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:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <locale.h>

#include <iostream>

#include <vector>

#include <time.h>

#include <stack>

#include <string>

#include <limits>

#include <queue>

using namespace std;

void BFS(int st, int N);

void BFS\_LIST(int st);

void DFS(int st, int N);

void DFS\_LIST(int st);

void T\_I\_M\_E(int start1, int start2, int start3, int start4, int end1, int end2, int end3, int end4, int N);

int\*\* graph, \* dist, \* dist3;

int N, i, j;

using Edge = int; // ребро - целое число, указывающее, куда ведёт

using Vertex = vector<Edge>; // вершина - набор рёбер

using Graph = vector<Vertex>; // граф - набор вершин

Graph g1;

vector<int> was; // посещённые вершины (для списка смежности)

using node2 = int;

using dist2 = vector<node2>; // посещённые вершины (нерекурсивный обход для матрицы)

dist2 dist\_list;

using namespace std;

queue <int> Q;

int st;

void main() {

setlocale(LC\_ALL, "russian");

printf("\n Введите количество вершин - ");

scanf("%d", &N);

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

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

{

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

}

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

dist3 = (int\*)malloc(N \* sizeof(int));

srand(time(NULL));

int R = 0;

for (int i = 0; i < N; i++) // генерация матрицы graph

{

graph[i][i] = 0;

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

{

R = rand() % 10;

if (R < 6)

graph[i][j] = 0;

else

graph[i][j] = 1;

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

}

}

if (N < 10) {

printf("\n Матрица смежности\n\n ");

for (j = 0; j < N; j++)

{

printf("%4d ", j + 1); //горизонт

}

printf("\n\n");

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

{

printf(" %d ", i + 1); // вертик

for (j = 0; j < N; j++)

{

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

}

printf("\n\n");

}

}

g1.resize(N); // задали размер графа

for (int i = 0; i < N; i++) // заполнение сгенерированнами матрицами

{

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

{

if (graph[i][j] == 1)

g1[i].push\_back(j);

}

}

printf("\n\n");

if (N < 10) {

for (int i = 0; i < g1.size(); i++) // вывод списка

{

printf(" Вершина %d смежна с ", i + 1);

for (int j = 0; j < g1[i].size(); j++)

{

printf("%d ", g1[i][j] + 1);

}

printf("\n");

}

}

for (int i = 0; i < N; i++) // исходно все вершины равны -1

dist[i] = -1;

printf(" \n");

printf("\n Выберите вершину - ");

scanf("%d", &st);

st--;

printf("\n Поиск расстояний на основе обхода в ширину МАТРИЦЫ смежности\n ");

for (int i = 0; i < N; i++) // исходно все вершины равны -1

dist[i] = -1;

clock\_t start1 = clock();

BFS(st, N);

clock\_t end1 = clock();

printf("\n\n Поиск расстояний на основе обхода в ширину СПИСКА смежности\n ");

was.resize(N);

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

was[i] = -1; // исходно все вершины равны -1

}

clock\_t start2 = clock();

BFS\_LIST(st);

clock\_t end2 = clock();

printf("\n\n Поиск расстояний на основе обхода в глубину МАТРИЦЫ смежности\n ");

for (int i = 0; i < N; i++) // исходно все вершины равны -1

dist[i] = -1;

dist[st] = 0;

clock\_t start3 = clock();

DFS(st, N);

clock\_t end3 = clock();

printf("\n\n Поиск расстояний на основе обхода в глубину СПИСКА смежности\n ");

for (int i = 0; i < N; i++) // исходно все вершины равны -1

dist[i] = -1;

dist[st] = 0;

clock\_t start4 = clock();

DFS\_LIST(st);

clock\_t end4 = clock();

T\_I\_M\_E(start1, start2, start3, start4, end1, end2, end3, end4, N);

\_getch();

}

void BFS(int st, int N)

{

int r, i = 0;

Q.push(st);

dist[st] = 0;

while (!Q.empty())

{

st = Q.front();

Q.pop();

printf("%d[%d] ", st + 1, dist[st]);

for (r = 0; r < N; r++)

if ((graph[st][r] != 0) && (dist[r] == -1)) // ребро существует и вершина не посещена

{

Q.push(r);

dist[r] = dist[st] + 1;

}

}

}

void BFS\_LIST(int st) {

queue<int> q;

q.push(st); //в качестве начальной вершины используем введённую

was[st] = 0;

while (!q.empty()) {

int st = q.front(); //извлекаем из очереди текущую вершину

q.pop();

printf("%d[%d] ", st + 1, was[st]);

for (int ii : g1[st]) { //добавляем всех непосещённых соседей.

if (was[ii] == -1) {

q.push(ii);

was[ii] = was[st] + 1;

}

}

}

}

void DFS(int st, int N)

{

int r;

printf("%d[%d] ", st + 1, dist[st]);

for (r = 0; r < N; r++)

{

if ((graph[st][r] != 0) && (dist[r] == -1)) // ребро существует и вершина не посещена

{

dist[r] = dist[st] + 1;

DFS(r, N);

}

}

}

void DFS\_LIST(int st) {

printf("%d[%d] ", st + 1, dist[st]);

for (int i : g1[st]) {

if (dist[i] == -1)

{

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

DFS\_LIST(i); // запускаем новый обход от каждого соседа

}

}

}

void T\_I\_M\_E(int start1, int start2, int start3, int start4, int end1, int end2, int end3, int end4, int N)

{

printf("\n\n Размер исходной матрицы - %d X %d\n", N, N);

printf("\n BFS %.2f ms", difftime(end1, start1));

printf("\n BFS LIST %.2f ms", difftime(end2, start2));

printf("\n DFS %.2f ms", difftime(end3, start3));

printf("\n DFS LIST %.2f ms", difftime(end4, start4));

}

**Результат работы программы:**

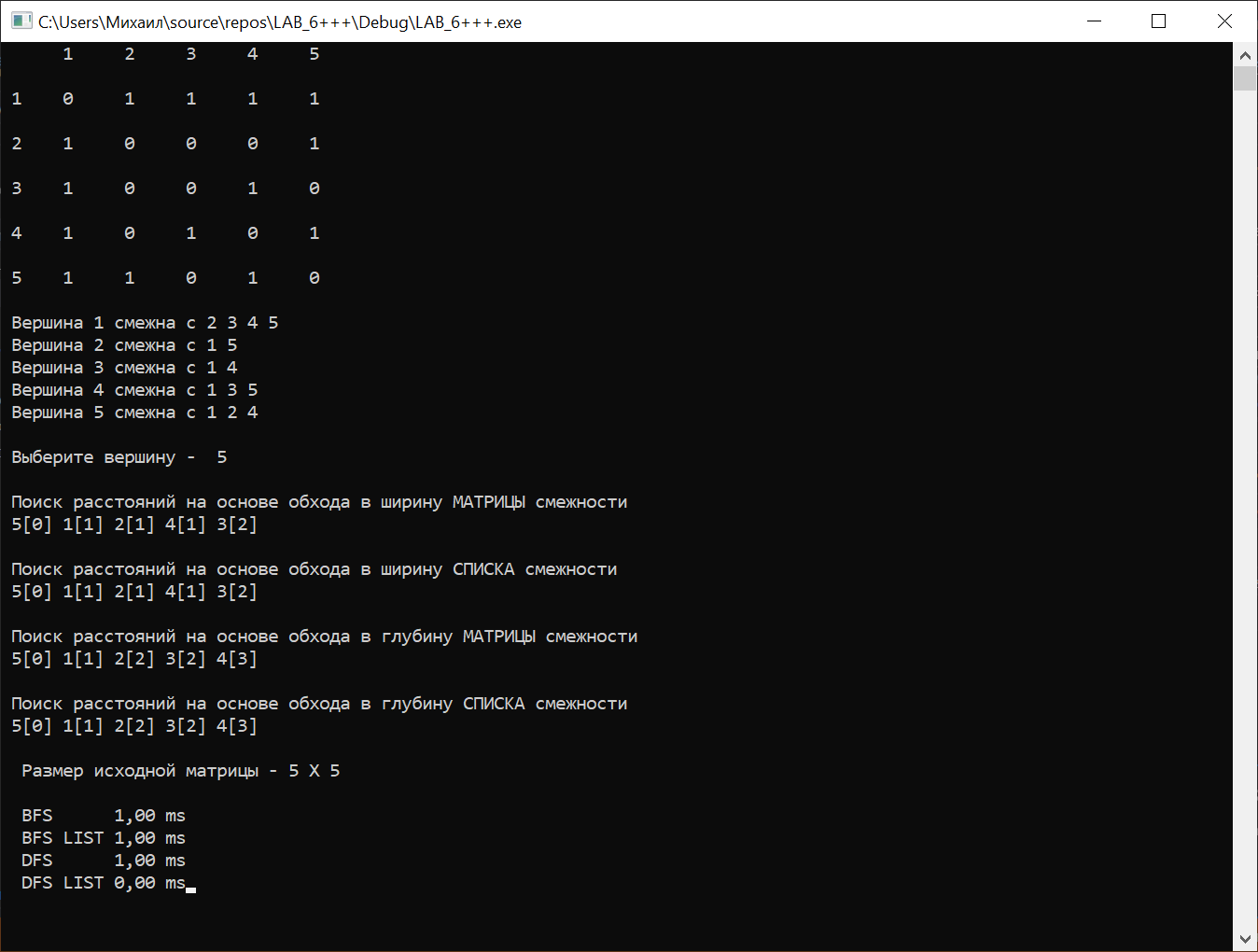
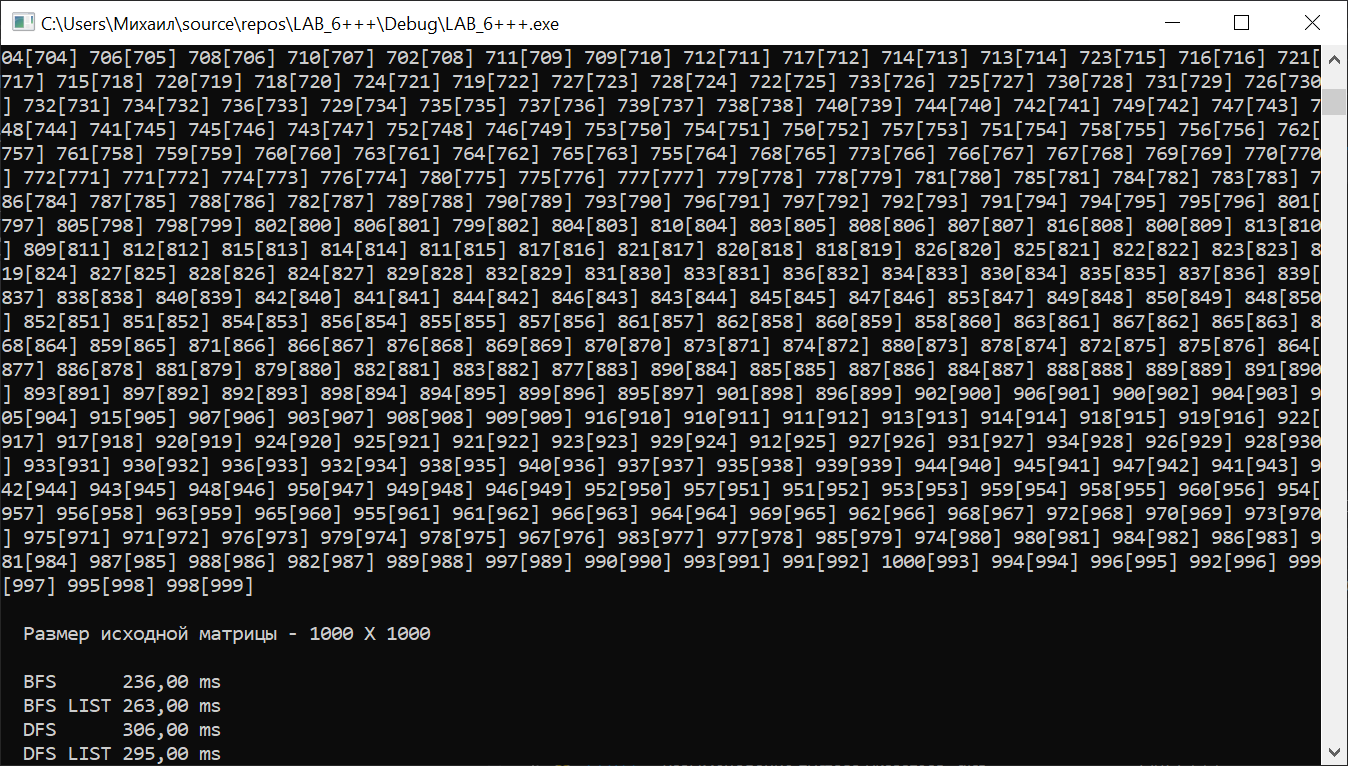
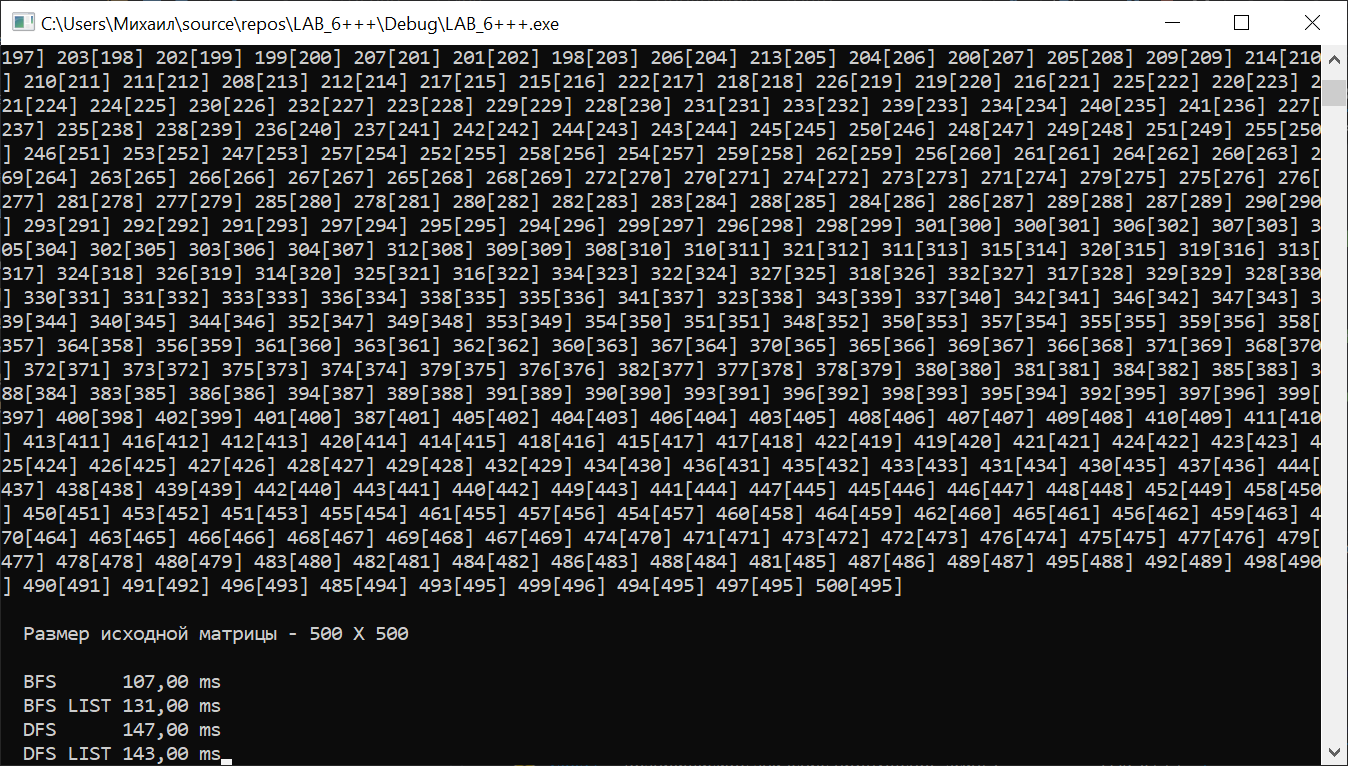
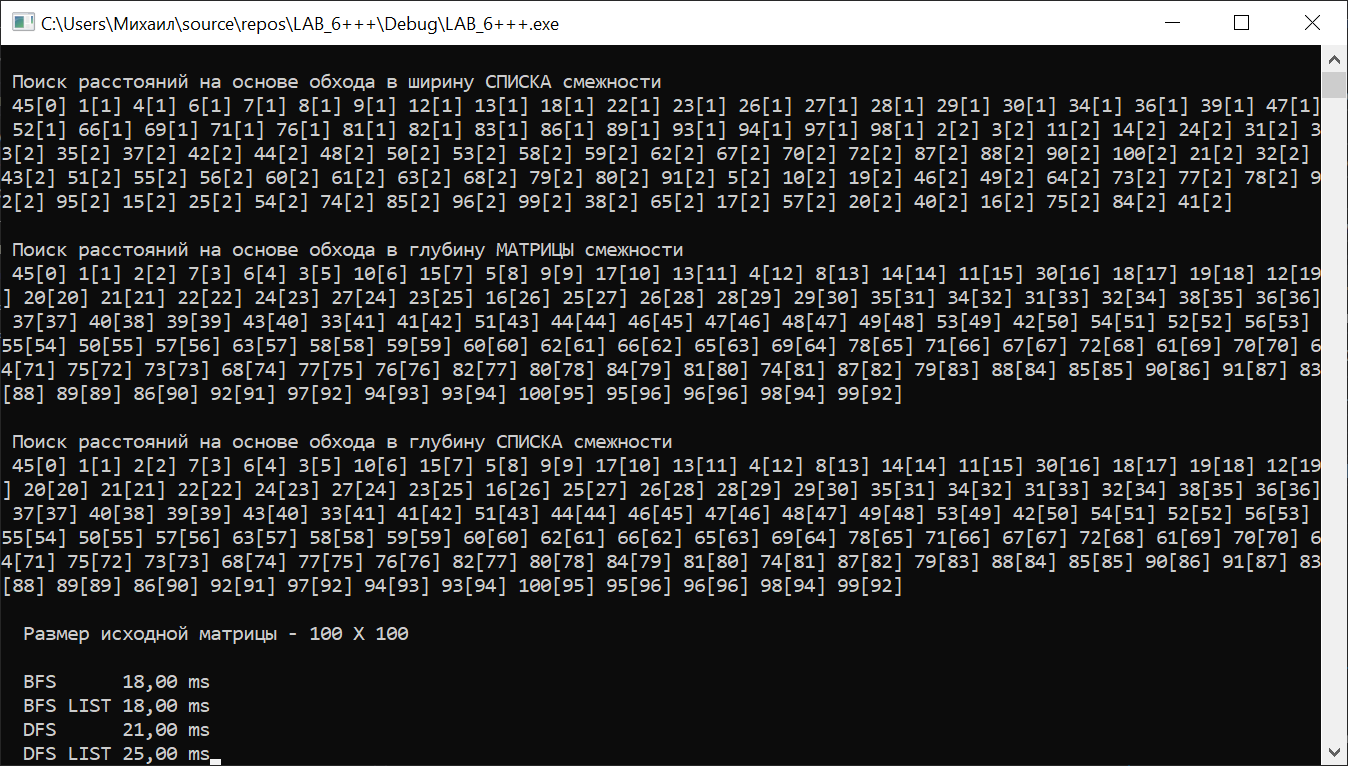
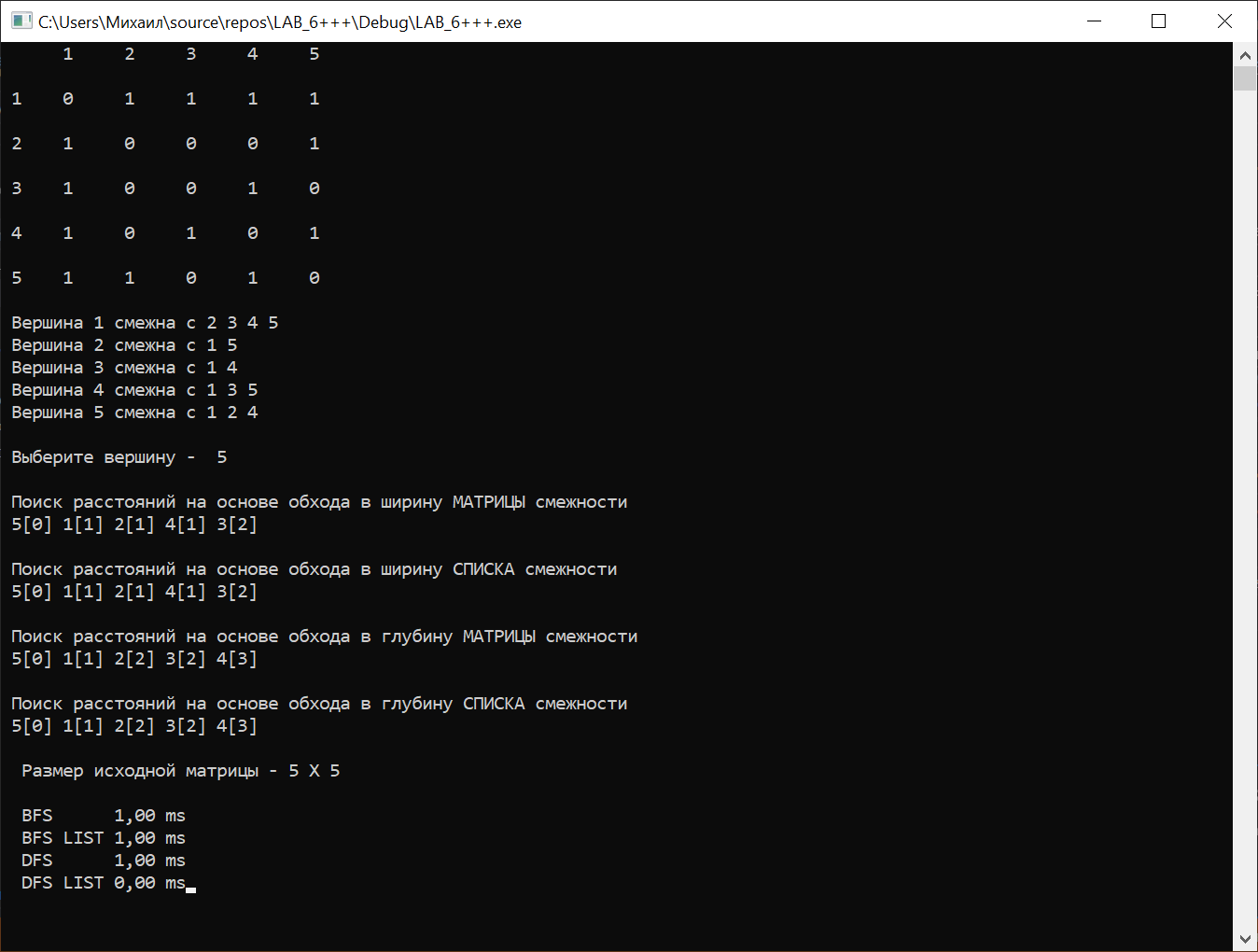


Рис. 1 Результат работы

Сравнили результата работы обхода в глубину и в ширину графов разных размеров, представленных матрицам смежности и списками смежности. Результат представлен на диаграмме.



**Вывод:**

В результате работы был реализован поиск расстояний в графе на основе обхода графа в ширину. А также сравнили обход в глубину и в ширину на матрице и списке смежности.