Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №9

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Поиск расстояний в графе**»

Выполнил:

студент группы 22ВВП1

Воробьева М.М.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Цель работы:** освоить такую операцию, как поиск расстояний от одной вершины в другую в графах, представленных в виде матрицы и списка смежности, на основе алгоритмов обхода графов в ширину и глубину.

**Лабораторное задание:**

**Задание 1**

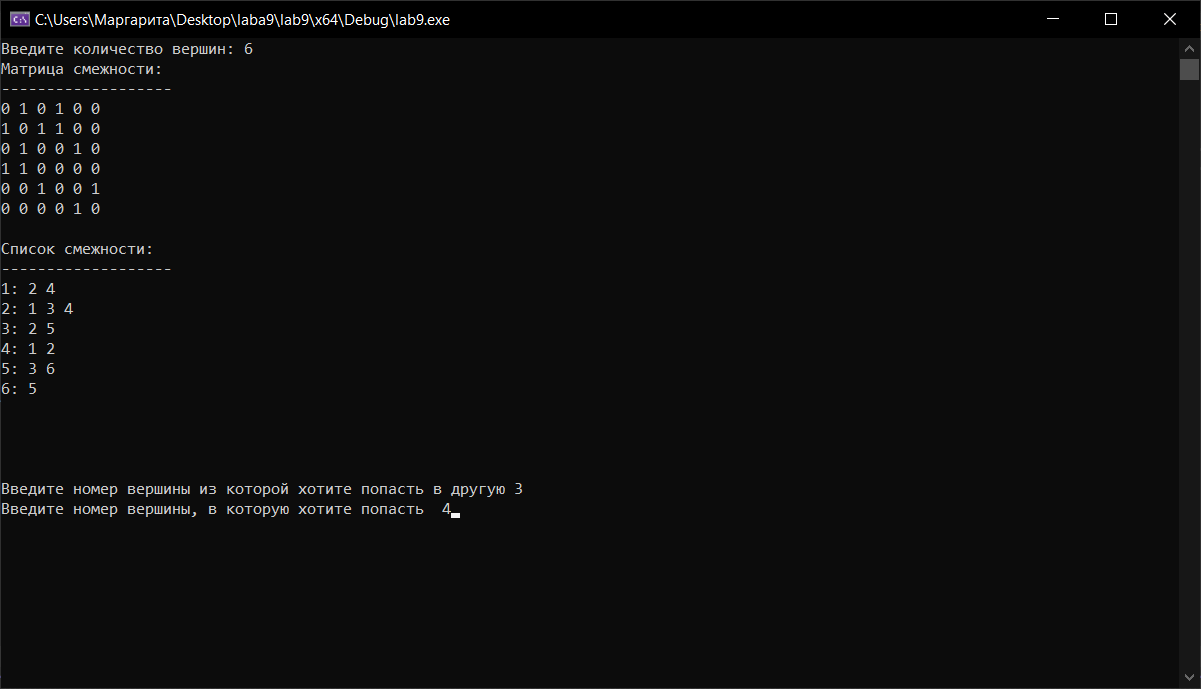
1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

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

**Задание 2\***

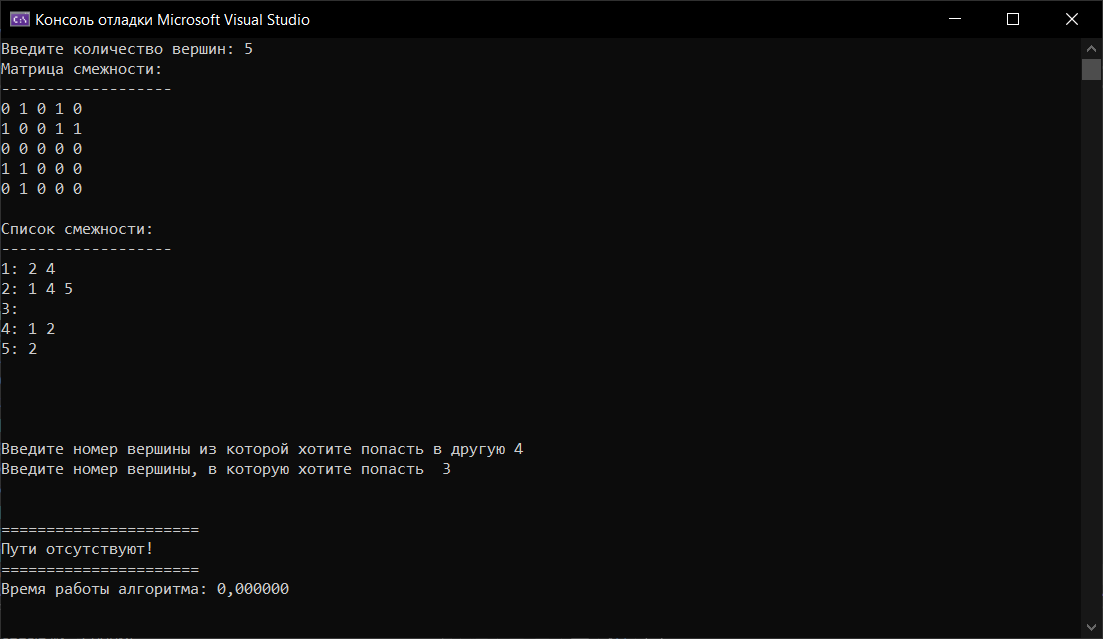
1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Результат работы программы:**





**Вывод:** в ходе выполнения лабораторной работы была освоена такая операция, как поиск расстояний от одной вершины в другую в графах, представленных в виде матрицы и списка смежности, на основе алгоритмов обхода графов в ширину и глубину.



**Листинг кода:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <queue>

using namespace std;

queue<int>Q;

void DFSD(vector < vector < int>>G, int s, vector<int>& dist, int d, int n = 0) {

clock\_t start = clock();

dist[s] = n;

for (int i = 1; i <= G.size() - 1; ++i) {

if (G[s][i] == 1 and dist[i] == -1) {

DFSD(G, i, dist, n + 1);

}

}

clock\_t end = clock();

if (dist[d] == -1) {

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

printf("Пути отсутствуют!\n");

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

}

else {

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

printf("Путь %d-> %d по МАТРИЦЕ(обход в глубину): %d\n",s, d, dist[d]);

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

}

printf("Время работы алгоритма: ");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n\n");

}

void DFSD\_List(vector < vector < int>>G, int s, vector<int>& dist,int d, int n = 0) {

clock\_t start = clock();

dist[s] = n;

for (int neighbor : G[s]) {

if (dist[neighbor] == -1) {

DFSD\_List(G, neighbor, dist, n + 1);

}

}

clock\_t end = clock();

if (dist[d] == -1) {

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

printf("Пути отсутствуют!\n");

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

}

else {

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

printf("Путь %d-> %d по СПИСКУ(обход в глубину): %d\n", s, d, dist[d]);

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

}

printf("Время работы алгоритма: ");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n\n");

}

void list\_BFS(int s, vector < vector < int>>adjacency\_list, vector<int>& dist,int d) {

int c = s;

clock\_t start = clock();

dist[s] = 0;

Q.push(s);

while (!Q.empty()) {

s = Q.front();

Q.pop();

for (int i: adjacency\_list[s]) { //добавляем всех непосещённых соседей.

if (dist[i] != 1 and adjacency\_list[s][i]==1) {

Q.push(i);

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

}

}

}

clock\_t end = clock();

if (dist[d] == -1) {

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

printf("Пути отсутствуют!\n");

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

}

else {

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

printf("Путь %d-> %d по СПИСКУ(обход в ширину): %d\n", c, d, dist[d]);

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

}

printf("Время работы алгоритма: ");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n\n");

}

void BFSD1(const vector < vector < int>> G, int s, vector<int>& dist, int d ) {

clock\_t start = clock();

int c = s;

Q.push(s);

dist[s] = 0;

while (!Q.empty()) {

s = Q.front();

Q.pop();

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

if (G[s][i] == 1 and dist[i] == -1) {

Q.push(i);

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

}

}

}

clock\_t end = clock();

if (dist[d] == -1) {

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

printf("Пути отсутствуют!\n");

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

}

else {

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

printf("Путь %d-> %d по МАТРИЦЕ(обход в ширину): %d\n", c, d, dist[d]);

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

}

printf("Время работы алгоритма: ");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n\n");

}

int swt = 0;

vector<vector<int>> cmej\_mat(int num\_vertices, double veroj) {

vector<vector<int>> cmej\_mat(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

int cont = 0;

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = i + 1; j <= num\_vertices; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < veroj) {

cmej\_mat[i][j] = 1;

cmej\_mat[j][i] = 1;

cont++;

}

}

}

if (cont == 0) {

swt = 1;

}

printf("Матрица смежности:\n");

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

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", cmej\_mat[i][j]);

}

printf("\n");

}

printf("\n");

return cmej\_mat;

}

vector<vector<int>> list\_cmej(vector<vector<int>> G) {

int num\_vertices = G.size() - 1;

vector<vector<int>> adjacency\_list(num\_vertices + 1);

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

if (G[i][j] == 1) {

adjacency\_list[i].push\_back(j);

}

}

}

return adjacency\_list;

}

void printList(vector<vector<int>>list) {

printf("Список смежности:\n");

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

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

printf("%d: ", i);

for (int j : list[i]) {

printf("%d ", j);

}

printf("\n");

}

}

int main() {

setlocale(LC\_ALL, "RUS");

int num\_vertices;

double veroj = 0.5;

int s,d;

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

auto G = cmej\_mat(num\_vertices, veroj);

auto list = list\_cmej(G);

printList(list);

printf("\n\n");

printf("\n\n");

printf("Введите номер вершины из которой хотите попасть в другую ");

scanf("%d", &s);

printf("Введите номер вершины, в которую хотите попасть ");

scanf("%d", &d);

printf("\n\n");

srand(static\_cast<unsigned int>(time(0)));

vector<int> dist(G.size(),-1);

if (swt == 1) {

printf("\n Обход произвести нельзя!\n");

}

else {

BFSD1(G,s,dist,d);

list\_BFS(s,G, dist, d);

DFSD(G, s, dist, d);

DFSD\_List(G, s, dist, d);

}

return 0;

}