Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 9

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

Выполнили:

студенты группы 22ВВВ2:

Зубриянова А.А.

Кондратьева В.И.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Поиск расстояний в графе.

**Цель работы**

Научиться выполнять поиск расстояний в графе.

**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.

**Описание метода решения задачи**

Задание 1:

1. Сгенерировали (используя генератор случайных чисел) матрицу смежности для неориентированного графа G.

2. Для сгенерированного графа осуществили процедуру поиска расстояний, реализованную с помощью обхода в ширину (функция BFSD()). При реализации алгоритма в качестве очереди использовали класс queue из стандартной библиотеки С++.

3.\* Реализовали процедуру поиска расстояний для графа, представленного списками смежности. Для этого использовали функцию first\_list() - обход в ширину.

Задание 2\*:

1. Реализовали процедуру поиска расстояний на основе обхода в глубину. Использовали рекурсивную функцию DFS1().

2. Реализовали процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности. Использовали функцию second\_list().

3. Оценили 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 <windows.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#include <conio.h>

#include <iostream>

#include <queue>

using namespace std;

queue <int> Q;

int size1;

int\*\* arr1;

int\* dist;

struct Node

{

int n; //Номер вершины (0..N-1)

Node\* next;

};

Node\* LA; //Массив структур - списков смежности [N]

Node\* make\_LA()

{

Node\* la, \* p;

la = (Node\*)malloc(sizeof(Node) \* size1);

for (int i = 0; i < size1; i++)

{

la[i].n = i;

la[i].next = NULL;

p = &la[i];

for (int j = 0; j < size1; j++)

{

if (arr1[i][j] == 1)

{

p->next = (Node\*)malloc(sizeof(Node));

p = p->next;

p->n = j;

p->next = NULL;

}

}

}

return la;

}

void print\_LA()

{

Node\* p;

for (int i = 0; i < size1; i++)

{

p = &LA[i];

while (p != NULL)

{

printf("%d > ", p->n);

p = p->next;

}

printf("\n");

}

}

void first\_list(int l) // l - вершина, с которой начинаем обход

{

Node\* p;

p = &LA[l];

dist[l] = 0;

Q.push(l);

while (!Q.empty())

{

l = Q.front();

Q.pop();

p = &LA[l];

while (p->next != NULL)

{

if (dist[p->next->n] != -1)

{

p = p->next;

}

else

{

dist[p->next->n] = dist[l] + 1;

p = p->next;

Q.push(p->n);

}

}

}

}

void second\_list(int l, int count)

{

Node\* p;

p = &LA[l];

dist[l] = count;

while (p->next != NULL)

{

if (dist[p->next->n] == 0)

{

dist[p->next->n] = 1;

l = p->next->n;

count++;

second\_list(l, count);

count--;

}

p = p->next;

}

}

void create\_matrix() {

for (int i = 0; i < size1; i++) {

for (int j = i; j < size1; j++) {

if (i == j) { //главная диагональ == 0

arr1[i][j] = 0;

}

else { //ниже главной диагонали отразить то, что выше

arr1[i][j] = rand() % 2;

arr1[j][i] = arr1[i][j];

}

}

}

}

void print\_matrix() {

printf(" ");

for (int i = 0; i < size1; i++) {

printf("%4d ", i);

}

printf("\n");

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

for (int i = 0; i < size1; i++) {

printf("%4d|", i);

for (int j = 0; j < size1; j++) {

printf("%4d ", arr1[i][j]);

}

printf("\n");

}

printf("\n");

}

void BFSD(int v)

{

Q.push(v);

dist[v] = 0;

while (!Q.empty()) {

v = Q.front();

Q.pop();

for (int i = 0; i < size1; i++) {

if ((arr1[v][i] == 1) && (dist[i] == -1)) {

Q.push(i);

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

}

}

}

}

void DFS1(int v, int count) {

dist[v] = count;

for (int i = 0; i < size1; i++) {

if ((arr1[v][i] == 1) && (dist[i] == -1)) {

count++;

DFS1(i, count);

count--;

}

}

}

int main() {

//srand(time(NULL));

setlocale(LC\_ALL, "Russian");

printf("Задание 1.1\n");

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size1);

printf("\n");

dist = (int\*)malloc(sizeof(int) \* size1);

arr1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr1[i] = (int\*)malloc(sizeof(int) \* size1);

}

clock\_t start, end;

double time1, time2;

printf("Матрица M1:\n");

create\_matrix();

print\_matrix();

int h;

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

printf("Задание 1.2 (обход в ширину через queue)\n");

printf("Введите вершину (от 0 до %d) для начала обхода: ", size1 - 1);

scanf("%d", &h);

int o = h;

int l = h;

int k = h;

int a = h;

int b = h;

int n = h;

printf("\n");

for (int i = 0; i < size1; i++) {

dist[i] = -1;

}

start = clock();

BFSD(o);

end = clock();

time1 = (double)(end - start) / CLOCKS\_PER\_SEC;

for (int i = 0; i < size1; i++) {

printf("Расстояние от %d вершины до %d вершины = %d\n", a, i, dist[i]);

}

for (int i = 0; i < size1; i++) {

dist[i] = -1;

}

printf("\n");

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

printf("Задание 1.3 (обход в ширину через списки)\n");

printf("Список смежности S1:\n");

LA = make\_LA();

print\_LA();

printf("\n");

first\_list(k);

for (int i = 0; i < size1; i++) {

printf("Расстояние от %d вершины до %d вершины = %d\n", a, i, dist[i]);

}

printf("\n");

for (int i = 0; i < size1; i++) {

dist[i] = -1;

}

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

printf("Задание 2.1 (обход в глубину)\n");

int c = 0;

start = clock();

DFS1(b, c);

end = clock();

time2 = (double)(end - start) / CLOCKS\_PER\_SEC;

for (int i = 0; i < size1; i++) {

printf("Расстояние от %d вершины до %d вершины = %d\n", a, i, dist[i]);

}

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

printf("Задание 2.2 (обход в глубину через списки)\n");

c = 0;

second\_list(n, c);

for (int i = 0; i < size1; i++) {

printf("Расстояние от %d вершины до %d вершины = %d\n", a, i, dist[i]);

}

printf("\n");

printf("Время работы алгоритма поиска расстояний на основе обхода в ширину: %lf\n", time1);

printf("Время работы алгоритма поиска расстояний на основе обхода в глубину: %lf\n", time2);

for (int i = 0; i < size1; ++i) free(arr1[i]);

free(arr1);

free(dist);

\_getch();

return 0;

}

**Результаты работы программы**

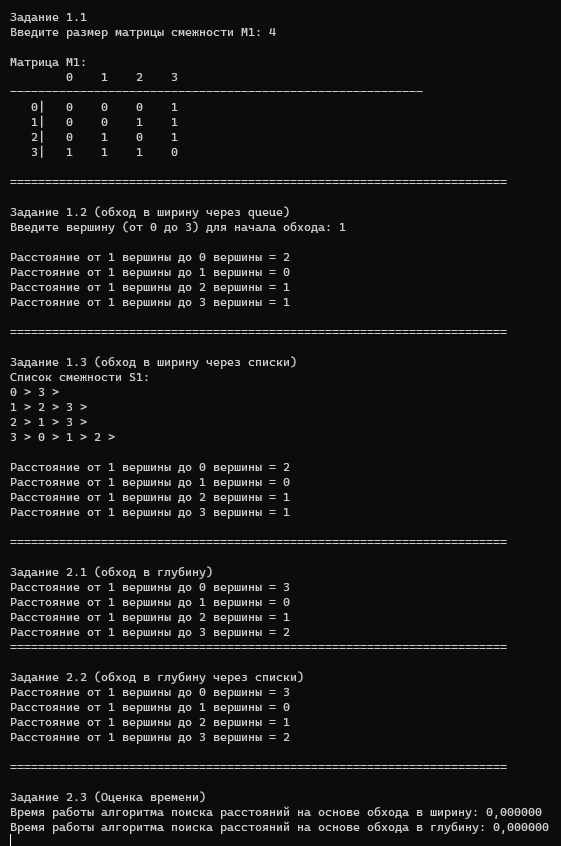


Рисунок 1 – Результат работы программы

Сравнение времени работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков:

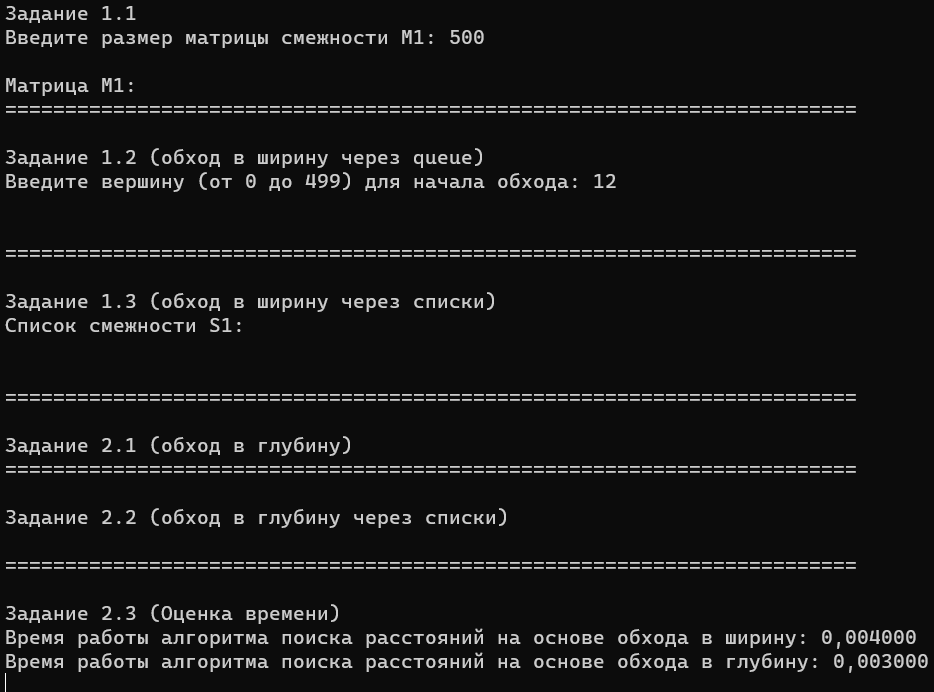


Рисунок 2 – Граф из 500 вершин

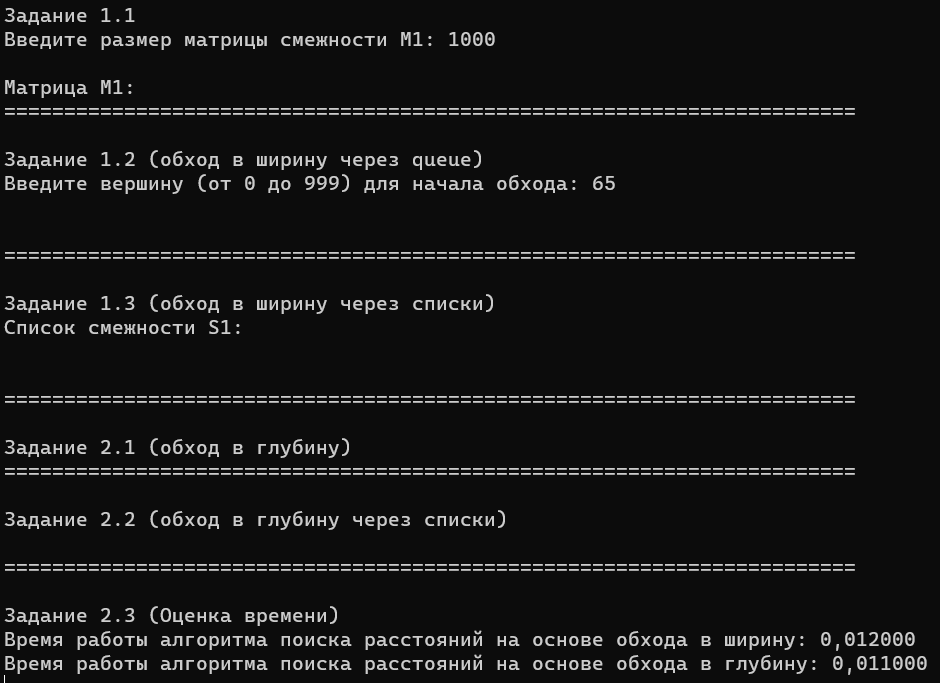


Рисунок 3 – Граф из 1000 вершин

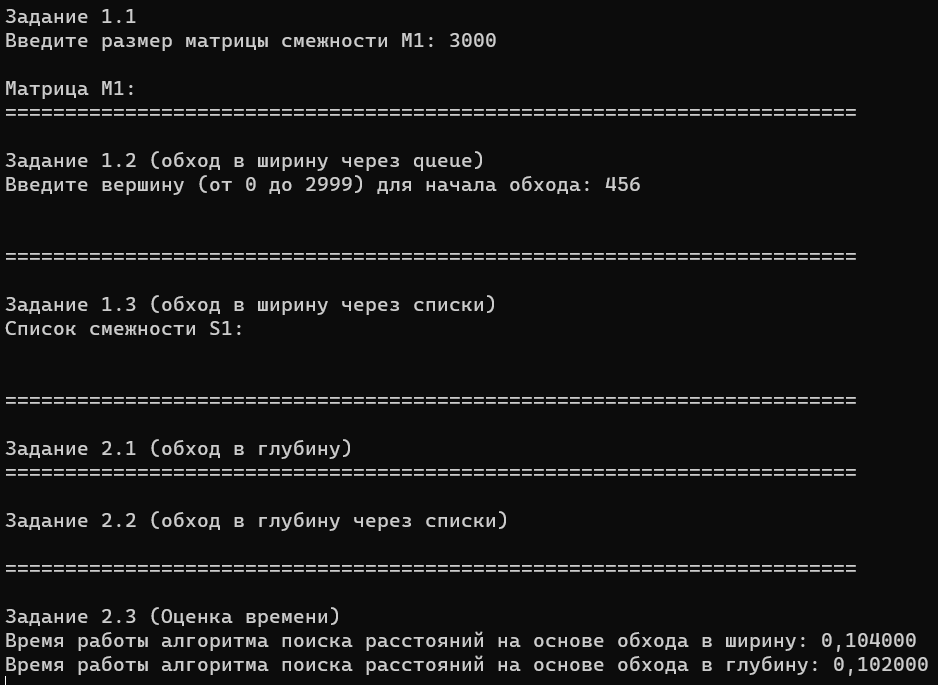


Рисунок 4 – Граф из 3000 вершин

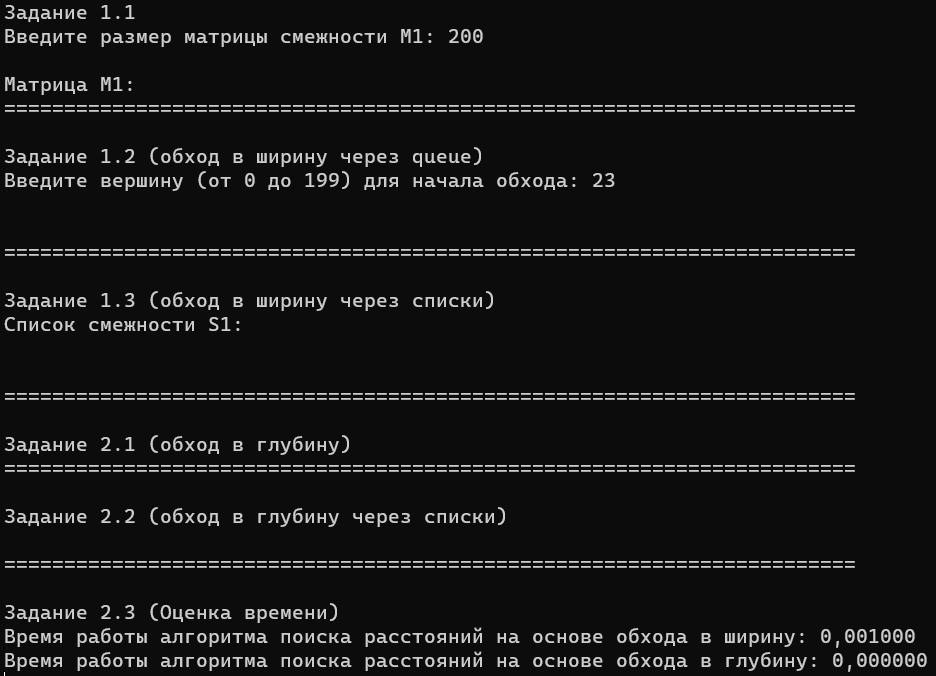


Рисунок 5 – Граф из 200 вершин

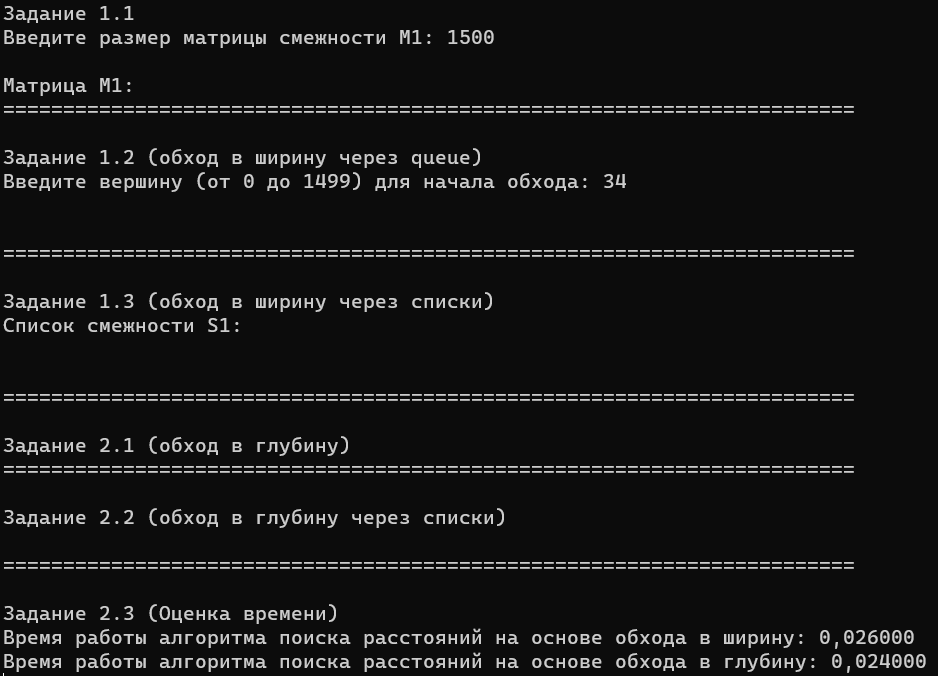


Рисунок 6 – Граф из 1500 вершин

|  |  |  |
| --- | --- | --- |
| **Количество вершин графа** | **Через обход в ширину** | **Через обход в глубину** |
| 200 | 0,001 | 0,000 |
| 500 | 0,004 | 0,003 |
| 1000 | 0,012 | 0,011 |
| 1500 | 0,026 | 0,024 |
| 3000 | 0,104 | 0,102 |

Таким образом, функция, реализованная через обход в глубину, работает быстрее, чем функция, реализованная через обход в ширину. Обход в ширину предполагает постоянное возвращение в предыдущую вершину, что делает данный алгоритм менее эффективным.

**Вывод**

В ходе выполнения лабораторной работы была разработана программа, выполняющая поиск расстояний в графе. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок.

Получили опыт в создании проектов в среде Microsoft Visual Studio, приобрели навыки программирования алгоритмов.