Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в ИС»

на тему «Поиск расстояний в графе»

Выполнили

студенты группы 20ВВ2:

Тумасов Вадим

Портнов Никита

Принял:

д.т.н Митрохин М.А.

д.т.н Юрова О.В.

Пенза 2021

**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..

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <ctime> // Библиотека времени

#include <cmath>

#include <locale.h>

#include <iostream> // Библиотека Ввода,Вывода

#include <conio.h>

#include < queue >

#include <time.h>

int\*\* matrix;

using namespace std;

int\* DIST;

int u = 0, n;

struct spisok\_smzh {

int versh;

spisok\_smzh\* next;

};

struct spiski\_smzh {

spisok\_smzh\* spisok;

spiski\_smzh\* next;

};

spiski\_smzh\* p\_spiski = NULL;

spiski\_smzh\* n\_spiski = NULL;

spiski\_smzh\* p\_spiski\_start = NULL;

spisok\_smzh\* p\_spisok = NULL;

spisok\_smzh\* p\_spisok\_start = NULL;

spisok\_smzh\* n\_spisok = NULL;

void bfs(int v, int n)

{

//clock\_t time;

//time = clock();

int p = 0, i = 0;

DIST = new int[n];

for (u = 0; u < n; u++)

{

DIST[u] = -1;

}

DIST[v] = 0;

//cout << v;

//i = s;

queue <int> plan; // план посещения в виде очереди

p\_spiski = p\_spiski\_start;

spisok\_smzh\* p\_spisok\_loc = NULL;

plan.push(v); // мы нумеруем с 0, а не с 1

//DIST[v] = 0; // отмечаем, что эта вершина уже заносилась в план

//int counter = 1; // начальную уже сосчитали

while (!plan.empty()) {

v = plan.front(); // посещаем следующую по плану вершину

plan.pop(); // удаляем ее из плана посещения

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

p\_spiski = p\_spiski->next;

p\_spisok\_loc = p\_spiski->spisok;

p\_spiski = p\_spiski\_start;

//cout « v;

//for (int i = 0; i < n; i++) { // перебираем соседние с ней

while (p\_spisok\_loc != NULL)

{

if (DIST[p\_spisok\_loc->versh] == -1) { // если новая, то

plan.push(p\_spisok\_loc->versh); // добавляем ее в план

//cout << i;

DIST[p\_spisok\_loc->versh] = DIST[v] + 1; // отмечаем, что уже не новая

//cout << DIST[i];

}

p\_spisok\_loc = p\_spisok\_loc->next;

}

//DIST[v] = DIST[v] + 1;

}

//time = clock() - time;

//cout << endl;

//printf("%.50f", (double)time / CLOCKS\_PER\_SEC);

cout << endl;

}

void bfsm(int v, int n)

{

//clock\_t time;

//time = clock();

int p = 0, i = 0;

DIST = new int[n];

for (u = 0; u < n; u++)

{

DIST[u] = -1;

}

DIST[v] = 0;

//cout << v;

//i = s;

queue <int> plan; // план посещения в виде очереди

plan.push(v); // мы нумеруем с 0, а не с 1

//DIST[v] = 0; // отмечаем, что эта вершина уже заносилась в план

//int counter = 1; // начальную уже сосчитали

while (!plan.empty()) {

v = plan.front(); // посещаем следующую по плану вершину

plan.pop(); // удаляем ее из плана посещения

//cout « v;

for (int i = 0; i < n; i++) { // перебираем соседние с ней

if (matrix[v][i] == 1 && DIST[i] == -1) { // если новая, то

plan.push(i); // добавляем ее в план

//cout << i;

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

//cout << DIST[i];

}

}

//DIST[v] = DIST[v] + 1;

}

//time = clock() - time;

//cout << endl;

//printf("%.50f", (double)time / CLOCKS\_PER\_SEC);

cout << endl;

}

void DFS(int v)

{

int r;

p\_spiski = p\_spiski\_start;

spisok\_smzh\* p\_spisok\_loc = NULL;

//cout << v << " ";

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

p\_spiski = p\_spiski->next;

p\_spisok\_loc = p\_spiski->spisok;

while (p\_spisok\_loc != NULL)

{

if (DIST[p\_spisok\_loc->versh] == -1 || ((DIST[v] + 1) < DIST[p\_spisok\_loc->versh]))

{

DIST[p\_spisok\_loc->versh] = DIST[v] + 1;

DFS(p\_spisok\_loc->versh);

}

p\_spisok\_loc = p\_spisok\_loc->next;

}

}

void DFSm(int v)

{

int r;

//cout << v << " ";

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

if (DIST[i] == -1 && matrix[v][i] == 1 || ((DIST[v] + 1) < DIST[i]))

{

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

DFSm(i);

}

}

}

int main()

{

int p = 0, i = 0, j, q;

int v = 0;

//int s;

setlocale(LC\_ALL, "Russian"); // подключение русского

srand(time(NULL)); // новый отсчёт времени

cout << "n="; // выводит данные

cin >> n; // помещает данные

matrix = new int\* [n];

//cout « "n=";

//cin » n;

cout << "Матрица: \n";

int z;

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

{

matrix[i] = new int[n];

}

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

{

for (p = 0; p < n; p++)

{

matrix[i][p] = rand() % 2;

}

}

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

{

for (p = 0; p < n; p++)

{

if (p == i)

{

matrix[i][p] = 0;

}

else

{

matrix[i][p] = matrix[p][i];

}

}

}

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

{

n\_spiski = (spiski\_smzh\*)malloc(sizeof(spiski\_smzh));

p\_spisok = NULL;

p\_spisok\_start = NULL;

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

{

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

{

n\_spisok = (spisok\_smzh\*)malloc(sizeof(spisok\_smzh));

n\_spisok->versh = j;

if (p\_spisok == NULL)

{

p\_spisok = n\_spisok;

p\_spisok\_start = p\_spisok;

}

else

{

p\_spisok->next = n\_spisok;

p\_spisok = n\_spisok;

}

n\_spisok->next = NULL;

}

}

n\_spiski->spisok = p\_spisok\_start;

if (p\_spiski == NULL)

{

p\_spiski = n\_spiski;

p\_spiski\_start = p\_spiski;

}

else

{

p\_spiski->next = n\_spiski;

p\_spiski = n\_spiski;

}

}

p\_spiski = p\_spiski\_start;

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

{

p\_spisok = p\_spiski->spisok;

p\_spisok\_start = p\_spisok;

cout << "\n";

cout << "{";

while (p\_spisok != NULL)

{

cout << p\_spisok->versh << " ";

p\_spisok = p\_spisok->next;

}

cout << "}" << " Вершина" << " " << i;

p\_spisok = p\_spisok\_start;

p\_spiski = p\_spiski->next;

}

cout << "\n";

p\_spiski = p\_spiski\_start;

cout << "\n";

cout << "\n";

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

{

// used[i] = false;

for (p = 0; p < n; p++)

cout << matrix[i][p] << " "; // выводит матрицу

cout << endl;

}

cout << "Стартовая вершина » ";

//cin » s;

cin >> v;

//bool\* vis = new bool[n];

cout << "Вектор расстояний: ";

clock\_t time;

time = clock();

bfs(v, n);

time = clock() - time;

cout << endl;

printf("%.50f", (double)time / CLOCKS\_PER\_SEC);

cout << endl;

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

{

if (DIST[i] == -1)

{

cout << "нет пути" << " ";

}

else

{

cout << " " << DIST[i] << " ";

}

}

for (u = 0; u < n; u++)

{

DIST[u] = -1;

}

cout << endl;

DIST[v] = 0;

clock\_t timeh1;

timeh1 = clock();

DFS(v);

timeh1 = clock() - timeh1;

cout << endl;

printf("%.50f", (double)timeh1 / CLOCKS\_PER\_SEC);

cout << endl;

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

{

if (DIST[i] == -1)

{

cout << "нет пути" << " ";

}

else

{

cout << " " << DIST[i] << " ";

}

}

cout << endl;

DIST[v] = 0;

for (u = 0; u < n; u++)

{

DIST[u] = -1;

}

clock\_t timeh2;

timeh2 = clock();

bfsm(v, n);

timeh2 = clock() - timeh2;

cout << endl;

printf("%.50f", (double)timeh2 / CLOCKS\_PER\_SEC);

cout << endl;

cout << endl;

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

{

if (DIST[i] == -1)

{

cout << "нет пути" << " ";

}

else

{

cout << " " << DIST[i] << " ";

}

}

//delete[] used;

cout << endl;

for (u = 0; u < n; u++)

{

DIST[u] = -1;

}

DIST[v] = 0;

clock\_t timeh;

timeh = clock();

DFSm(v);

timeh = clock() - timeh;

printf("%.50f", (double)timeh / CLOCKS\_PER\_SEC);

cout << endl;

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

{

if (DIST[i] == -1)

{

cout << "нет пути" << " ";

}

else

{

cout << " " << DIST[i] << " ";

}

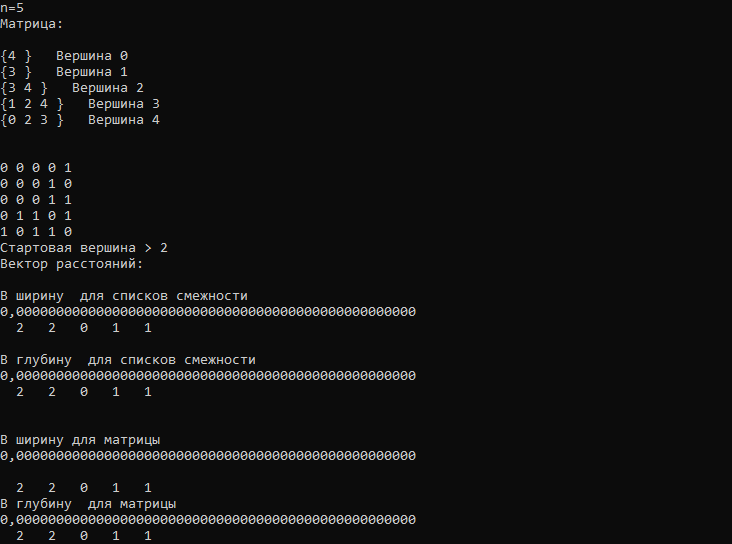
}

\_getch();

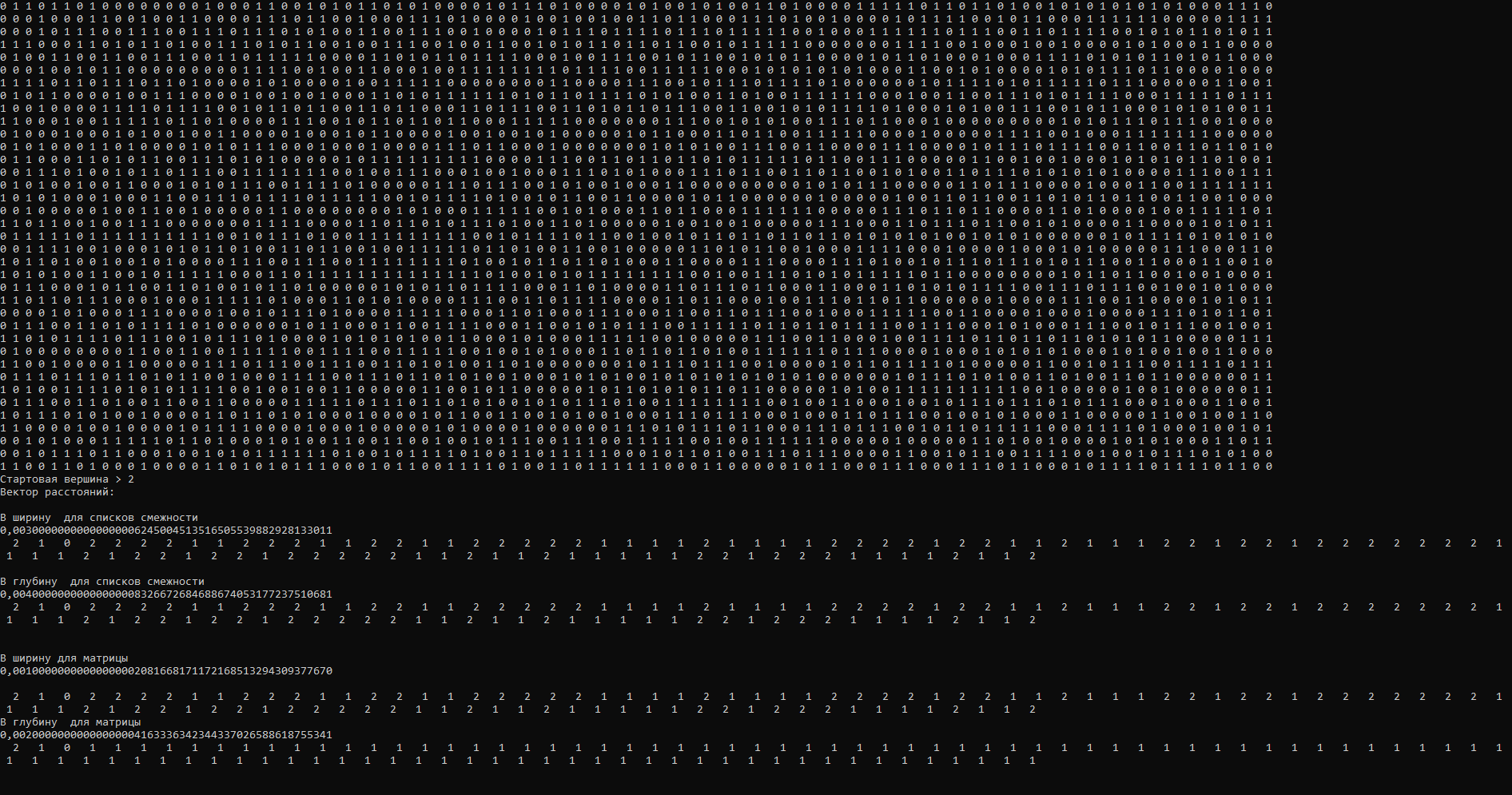
}

**Результат работы программы:**

Количество вершин=5



Количество вершин=100



**Вывод**: в результате выполнения данной работы мы научились выполнять поиск расстояний,используя DFS и BFS для списков смежности и матрицы смежности.Также,в результате выше приведенного эксперимента мы обнаружили,что самым эффективным способом найти расстояния является использование обхода в ширину для матрицы смежности.