Министерство науки и высшего 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 22ВВВ1

Дунюшкин В.А.

Ивлюшин М.Д.

Проверили:

к.т.н, доцент Юрова О.В.

к.э.н, доцент Акифьев И.В.

Пенза 2023

**Цель работы:**

Научиться пользоваться алгоритмом поиска 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.

.

**Ход работы:**

**Ввод данных:**

Пользователь вводит количество вершин графа (n).

**Генерация матрицы смежности:**

Генерируется случайная матрица смежности (Matr). Значения в матрице равны 0 или 1, указывая отсутствие или наличие ребра между вершинами соответственно.

**Создание списка смежности и графа:**

На основе сгенерированной матрицы смежности создаются список смежности (list) и граф (graph).

**Поиск расстояний с использованием матрицы смежности:**

Пользователь вводит начальную и конечную вершины для поиска расстояния между ними в графе, представленном матрицей смежности.

Выполняется обход в ширину (RGM(graph, nachalo1, konec1, distances)) для нахождения кратчайшего пути.

Выполняется обход в глубину (GGM(graph, nachalo2, konec2, distances2)) для нахождения пути с использованием поиска в глубину.

**Создание и вывод списка смежности:**

Программа создает список смежности (list) на основе матрицы смежности и выводит его на экран.

**Поиск расстояний с использованием списка смежности:**

Пользователь вводит начальную и конечную вершины для поиска расстояния между ними в графе, представленном списком смежности.

Выполняется обход в ширину (GGM\_list(list, nachalo3, konec3, distances3)) для нахождения кратчайшего пути.

Выполняется обход в глубину (DFS\_list(list, nachalo4, konec4, distances4)) для нахождения пути с использованием поиска в глубину.

**Измерение времени выполнения обходов:**

Программа измеряет время выполнения обхода в ширину и глубину для матрицы смежности с помощью хронометража.

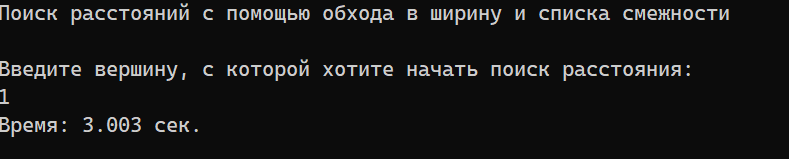
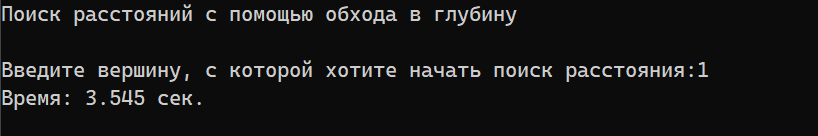
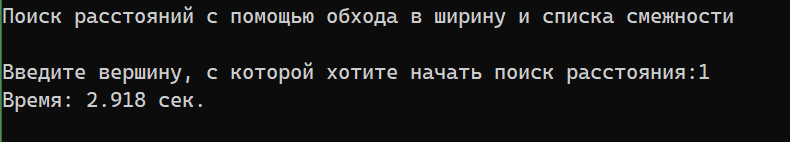
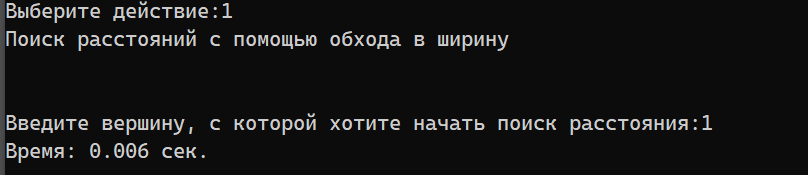
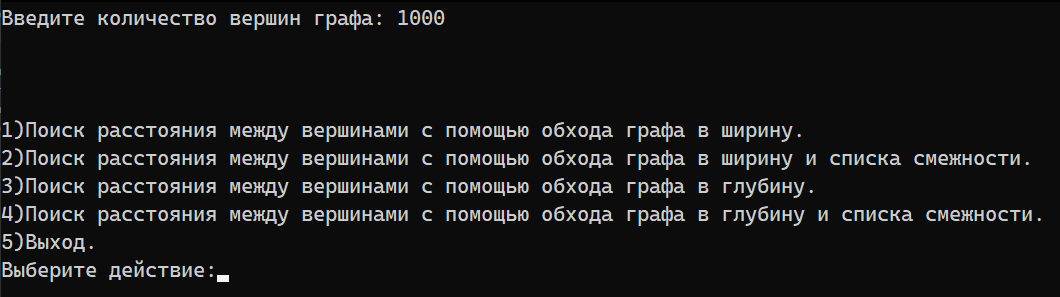
**Вывод результатов:**

Программа выводит кратчайшие расстояния для обходов в ширину и глубину для матрицы смежности, список смежности и время выполнения каждого обхода.

**Окончание работы программы:**

Программа завершает выполнение, выводя результаты на экран и завершая работу.

**Результаты работы программы:**



**Оценка:**

Время работы алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину зависит от порядка графа (количества вершин) и его структуры (количество ребер)

Для графа с n вершинами и m ребрами:

- Время работы алгоритма поиска в глубину составляет O(n + m), так как он посещает каждую вершину и каждое ребро один раз.

- Время работы алгоритма поиска в ширину составляет O(n + m), так как он посещает каждую вершину и каждое ребро один раз, но при этом используется очередь, что увеличивает время работы.

Таким образом, время работы обоих алгоритмов одинаковое для графов с одинаковым количеством вершин и ребер. Однако, если граф имеет сложную структуру (например, большое количество циклов), то время работы алгоритма поиска в глубину может быть больше, чем время работы алгоритма поиска в ширину.

**Вывод**

Лабораторная работа позволила изучить и реализовать методы поиска расстояний в графе. Обходы в ширину и глубину предоставляют эффективные способы нахождения кратчайших путей между вершинами. Работа с графом в виде матрицы смежности и списка смежности позволяет выбирать наиболее удобный способ представления данных в зависимости от поставленной задачи. Измерение времени выполнения обходов демонстрирует разницу в эффективности алгоритмов, что является важным критерием выбора метода при работе с большими графами.

**Листинг**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cstdio>

#include <queue>

using namespace std;

typedef struct Spis

{

int inf;

struct Spis\* next;

}SP;

int i, j, ver;

void DistWidth(int\*\* smej, int\* visit, int ver, int ver1)

{

queue<int> q;

visit[ver1 - 1] = 0;

q.push(ver1);

while (!q.empty())

{

int current = q.front();

q.pop();

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

{

if (smej[current - 1][i] == 1 && visit[i] == -1)

{

visit[i] = visit[current - 1] + 1;

q.push(i + 1);

}

}

}

}

void DistWidthSP(SP\*\* MassSpis, int\* visit, int ver)

{

queue<int> q;

visit[ver - 1] = 0;

q.push(ver);

while (!q.empty())

{

int currentVer = q.front();

q.pop();

SP\* current = MassSpis[currentVer - 1];

while (current != NULL)

{

int temp = current->inf;

if (visit[temp - 1] == -1)

{

visit[temp - 1] = visit[currentVer - 1] + 1;

q.push(temp);

}

current = current->next;

}

}

}

void DistDepth(int\*\* smej, int\* visit, int ver3, int dist, int ver)

{

if (visit[ver3 - 1] == -1 || dist < visit[ver3 - 1])

{

visit[ver3 - 1] = dist;

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

{

if (smej[ver3 - 1][i] == 1)

{

DistDepth(smej, visit, i + 1, dist + 1, ver);

}

}

}

}

void DistDepthSP(SP\*\* MassSpis, int\* visit, int ver, int dist)

{

if (visit[ver - 1] == -1 || dist < visit[ver - 1])

{

visit[ver - 1] = dist;

SP\* current = MassSpis[ver - 1];

while (current != NULL)

{

DistDepthSP(MassSpis, visit, current->inf, dist + 1);

current = current->next;

}

}

}

void main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* smej = 0; //Матрица смежности

int choice, ver1, ver2, ver3, ver4, k = 0;

SP\*\* MassSpis = NULL;//Массив списков смежности

int\* visit = 0;

bool prov1 = true, prov2 = true, prov3 = true, prov4 = true;

clock\_t start, end;

double TT = 0.0;

cout << "Введите количество вершин графа: ";

for (;;) {

cin >> ver;

if (cin.fail() || ver < 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер количество вершин графа! Повторите попытку: ";

}

else { break; }

}

cout << endl;

//Выделение памяти под массив смежности

smej = new int\* [ver];

if (smej == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

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

{

smej[i] = new int[ver];

}

//Выделение памяти под массив списков

MassSpis = new SP \* [ver];

if (MassSpis == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

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

{

MassSpis[i] = NULL;

}

//Генерация массива смежности

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

{

for (j = i; j < ver; j++)

{

if (i == j)

{

smej[i][j] = 0; // на главной диагонали нули

}

else

{

smej[i][j] = rand() % 2; // случайные значения 0 или 1

smej[j][i] = smej[i][j]; // симметрично заполнять значения для неориентированного графа

}

}

}

//Генерация списка смежности

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

{

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

{

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

{

SP\* newSP = new SP;

newSP->inf = j + 1;

newSP->next = MassSpis[i];

MassSpis[i] = newSP;

}

}

}

//Вывод матрицы смежности

/\* cout << "Матрица смежности:" << endl;

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

{

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

{

cout << smej[i][j] << " ";

}

cout << endl;

}\*/

cout << endl;

int isol = 0;

do

{

cout << endl;

cout << "1)Поиск расстояния между вершинами с помощью обхода графа в ширину." << endl;

cout << "2)Поиск расстояния между вершинами с помощью обхода графа в ширину и списка смежности." << endl;

cout << "3)Поиск расстояния между вершинами с помощью обхода графа в глубину." << endl;

cout << "4)Поиск расстояния между вершинами с помощью обхода графа в глубину и списка смежности." << endl;

cout << "5)Выход." << endl;

cout << "Выберите действие:";

cin >> choice;

switch (choice)

{

case 1:

isol = 0;

k = 0;

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

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

{

visit[i] = -1;

}

/\* system("cls");

cout << "Матрица смежности:" << endl;

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

{

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

{

cout << smej[i][j] << " ";

}

cout << endl;

}

cout << endl;

//Вывод списка смежности

cout << "Список смежности:" << endl << endl;

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

{

if (MassSpis[i] == NULL)

{

cout << "Вершина " << i + 1 << " не имеет смежных";

}

else

{

cout << "Вершина " << i + 1 << " смежна с:";

SP\* current = MassSpis[i];

while (current != NULL)

{

cout << current->inf << " ";

current = current->next;

}

}

cout << endl;

}

\*/

cout << "Поиск расстояний с помощью обхода в ширину" << endl << endl;

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать поиск расстояния:";

cin >> ver1;

if (cin.fail() || (ver1 < 1 || ver1 > ver)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

}

else

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

{

//k++;

if (smej[ver1 - 1][i] == 1 || smej[i][ver1 - 1] == 1)

{

k++;

prov1 = false;

break;

}

if (k == 0 && i == (ver - 1))

{

cout << "Вершина " << ver1 << " изолированная!" << endl;

isol = 1;

prov1 = false;

break;

}

}

} while (prov1);

start = clock();

DistWidth(smej, visit, ver, ver1);

end = clock();

/\* for (i = 0; i < ver; i++)

{

if (isol == 1)

{

cout << "Расстояние от вершины " << ver1 << " до вершины " << ver1 << " = 0" << endl;

break;

}

else if (isol != 1)

{

if (visit[i] != -1) { cout << "Расстояние от вершины " << ver1 << " до вершины " << i + 1 << " = " << visit[i] << endl; }

}

}\*/

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

cout << "Время: " << TT << " сек.";

cout << endl;

delete[] visit;

break;

case 2:

system("cls");

isol = 0;

/\* cout << "Матрица смежности:" << endl;

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

{

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

{

cout << smej[i][j] << " ";

}

cout << endl;

}

cout << endl;

cout << "Список смежности:" << endl << endl;

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

{

if (MassSpis[i] == NULL)

{

cout << "Вершина " << i + 1 << " не имеет смежных";

}

else

{

cout << "Вершина " << i + 1 << " смежна с:";

SP\* current = MassSpis[i];

while (current != NULL)

{

cout << current->inf << " ";

current = current->next;

}

}

cout << endl;

}\*/

cout << "Поиск расстояний с помощью обхода в ширину и списка смежности" << endl;

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

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

{

visit[i] = -1;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать поиск расстояния:";

cin >> ver2;

if (cin.fail() || ver2 < 1 || ver2 > ver) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

continue;

}

if (MassSpis[ver2 - 1] == 0) { cout << "Вершина " << ver2 << " изолированная! " << endl; prov2 = false; isol = 1; }

else prov2 = false;

} while (prov2);

cout << endl;

start = clock();

DistWidthSP(MassSpis, visit, ver2);

end = clock();

/\*

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

{

if (isol == 1)

{

cout << "Расстояние от вершины " << ver2 << " до вершины " << ver2 << " = 0" << endl;

break;

}

else if (isol != 1)

{

if (visit[i] != -1) { cout << "Расстояние от вершины " << ver2 << " до вершины " << i + 1 << " = " << visit[i] << endl; }

}

}\*/

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

cout << "Время: " << TT << " сек.";

cout << endl;

delete[] visit;

break;

case 3:

isol = 0;

/\* system("cls");

cout << "Матрица смежности:" << endl;

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

{

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

{

cout << smej[i][j] << " ";

}

cout << endl;

}

cout << endl;

cout << "Список смежности:" << endl << endl;

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

{

if (MassSpis[i] == NULL)

{

cout << "Вершина " << i + 1 << " не имеет смежных";

}

else

{

cout << "Вершина " << i + 1 << " смежна с:";

SP\* current = MassSpis[i];

while (current != NULL)

{

cout << current->inf << " ";

current = current->next;

}

}

cout << endl;

}\*/

cout << "Поиск расстояний с помощью обхода в глубину " << endl;

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

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

{

visit[i] = -1;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать поиск расстояния:";

cin >> ver3;

if (cin.fail() || (ver3 < 1 || ver3 > ver))

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

}

else

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

{

k++;

if (smej[ver3 - 1][i] == 1 || smej[i][ver3 - 1] == 1)

{

prov3 = false;

break;

}

else if (k == 1) {

cout << "Вершина " << ver3 << " изолированная!" << endl; isol = 1; prov3 = false; break;

}

}

} while (prov3);

start = clock();

DistDepth(smej, visit, ver3, 0, ver);

end = clock();

/\* for (i = 0; i < ver; i++)

{

if (isol == 1)

{

cout << "Расстояние от вершины " << ver3 << " до вершины " << ver3 << " = 0" << endl;

break;

}

else if (isol != 1)

{

if (visit[i] != -1) { cout << "Расстояние от вершины " << ver3 << " до вершины " << i + 1 << " = " << visit[i] << endl; }

}

}\*/

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

cout << "Время: " << TT << " сек.";

cout << endl;

delete[] visit;

break;

case 4:

system("cls");

isol = 1;

/\* cout << "Матрица смежности:" << endl;

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

{

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

{

cout << smej[i][j] << " ";

}

cout << endl;

}

cout << endl;

cout << "Список смежности:" << endl << endl;

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

{

if (MassSpis[i] == NULL)

{

cout << "Вершина " << i + 1 << " не имеет смежных";

}

else

{

cout << "Вершина " << i + 1 << " смежна с:";

SP\* current = MassSpis[i];

while (current != NULL)

{

cout << current->inf << " ";

current = current->next;

}

}

cout << endl;

}\*/

cout << "Поиск расстояний с помощью обхода в ширину и списка смежности" << endl;

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

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

{

visit[i] = -1;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать поиск расстояния:";

cin >> ver4;

if (cin.fail() || ver4 < 1 || ver4 > ver) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

continue;

}

if (MassSpis[ver4 - 1] == 0) { cout << "Вершина " << ver4 << " изолированная!" << endl; prov4 = false; break; }

else prov4 = false;

} while (prov4);

start = clock();

DistDepthSP(MassSpis, visit, ver4, 0);

end = clock();

/\* for (i = 0; i < ver; i++)

{

if (isol == 1)

{

cout << "Расстояние от вершины " << ver4 << " до вершины " << ver4 << " = 0" << endl;

break;

}

else if (isol != 1)

{

if (visit[i] != -1) { cout << "Расстояние от вершины " << ver4 << " до вершины " << i + 1 << " = " << visit[i] << endl; }

}

}\*/

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

cout << "Время: " << TT << " сек.";

cout << endl;

delete[] visit;

break;

case 5:

system("cls");

break;

default:

cout << endl;

cout << "Неверный выбор! Повторите попытку.";

cout << endl;

break;

}

} while (choice != 5);

delete[] MassSpis;

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

{

delete[] smej[i];

}

delete[] smej;

}