Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №9

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

Выполнили ст. группы 22ВВВ1:

Лёвин А.Д.

Колобов И.О.

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. 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.

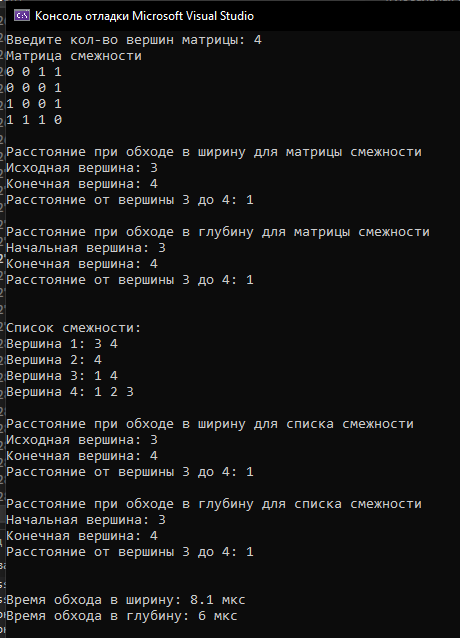
**Вывод результатов:**

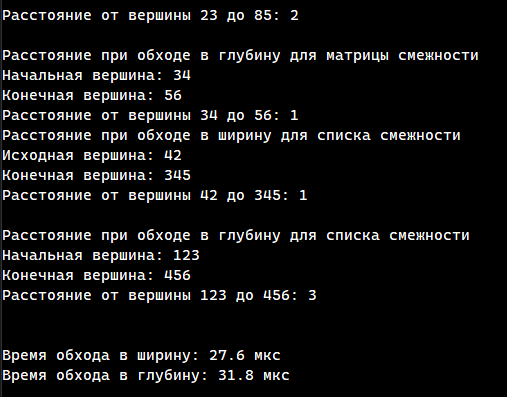
Программа выводит кратчайшие расстояния для обходов в ширину и глубину для матрицы смежности, список смежности и время выполнения каждого обхода.

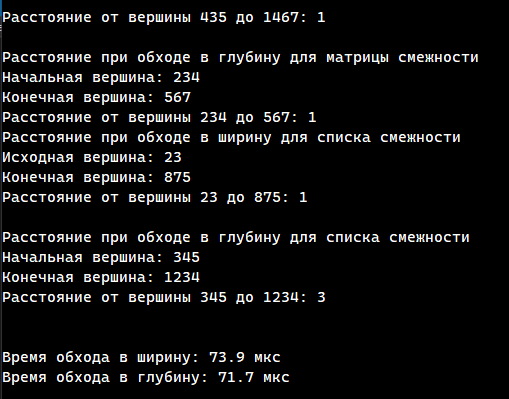
**Окончание работы программы:**

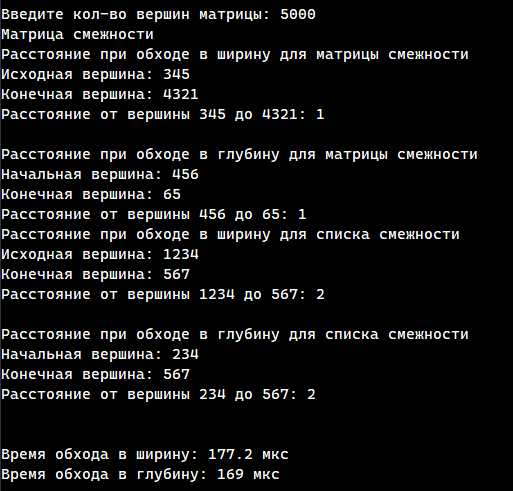
Программа завершает выполнение, выводя результаты на экран и завершая работу.

**Результаты работы программы:**



**Оценка скорости**   
Кол-во вершин 500:  


Кол-во вершин 2000**:  
**

Кол-во вершин 5000: ****

**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.  
 При оценки скорости обход в глубину лучше подходит для больших графов, показывая лучший результат, но обход в ширину лучше подходит для маленьких графов.

**Листинг:**

#include<iostream>

#include<cstdlib>

#include<stack>

#include<vector>

#include<algorithm>

#include<queue>

#include<chrono>

usingnamespacestd;

//Фунция для нахождения расстояния при обходе в ширину для матрицы смежности

void RGM(constvector<vector<int>>&graph, intnachalo1, intkonec1, vector<int>&distances) {

queue<int> q;

q.push(nachalo1);

distances[nachalo1] = 0;

while (!q.empty()) {

int v = q.front();

q.pop();

if (v == konec1) {

break;

}

for (inti = 0; i<graph[v].size(); i++) {

int to = graph[v][i];

if (distances[to] == -1) {

q.push(to);

distances[to] = distances[v] + 1;

}

}

}

}

// Функция для нахождения расстояния при обходе в глубину для матрицы смежности

void GGM(constvector<vector<int>>&graph, intnachalo2, intkonec2, vector<int>&distances2) {

stack<int> s;

s.push(nachalo2);

distances2[nachalo2] = 0;

while (!s.empty()) {

int v = s.top();

s.pop();

if (v == konec2) break;

for (inti = 0; i<graph[v].size(); ++i) {

int to = graph[v][i];

if (distances2[to] == -1) {

distances2[to] = distances2[v] + 1;

s.push(to);

}

}

}

}

//Фунция для нахождения расстояния при обходе в ширину для списка смежности

voidGGM\_list(constvector<vector<int>>&list, intnachalo3, intkonec3, vector<int>&distances3) {

queue<int> q;

q.push(nachalo3);

distances3[nachalo3] = 0;

while (!q.empty()) {

int v = q.front();

q.pop();

if (v == konec3) break;

for (inti = 0; i<list[v].size(); ++i) {

int to = list[v][i];

if (distances3[to] == -1) {

distances3[to] = distances3[v] + 1;

q.push(to);

}

}

}

}

// Функция для поиска расстояния в глубину для списка смежности

voidDFS\_list(constvector<vector<int>>&list1, intnachalo4, intfinish1, vector<int>&distances4) {

stack<int> s;

s.push(nachalo4);

distances4[nachalo4] = 0;

while (!s.empty()) {

int v = s.top();

s.pop();

if (v == finish1) break;

for (inti = 0; i<list1[v].size(); ++i) {

int to = list1[v][i];

if (distances4[to] == -1) {

distances4[to] = distances4[v] + 1;

s.push(to);

}

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int n;

cout<<"Введите кол-во вершин матрицы: ";

cin>> n;

vector<vector<int>>Matr(n, vector<int>(n));

vector<vector<int>> list(n);

vector<vector<int>> graph(n, vector<int>(n));

// Матрица 1

cout<<"Матрица смежности "<<endl;

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

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

if (i == j) {

Matr[i][j] = 0;

}

else {

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

Matr[j][i] = Matr[i][j];

}

}

}

// Выводматрицы 1

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

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

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

}

cout<<endl;

}

cout<<"\n";

// Заполнение графа на основе матрицы смежности

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

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

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

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

}

}

}

//Нахождения расстояния при обходе в ширину для матрицы смежности

cout<<"Расстояние при обходе в ширину для матрицы смежности\n";

int nachalo1;

cout<<"Исходная вершина: ";

cin>> nachalo1;

nachalo1--;

int konec1;

cout<<"Конечная вершина: ";

cin>> konec1;

konec1--;

vector<int> distances(n, -1);

RGM(graph, nachalo1, konec1, distances);

if (distances[konec1] != -1) {

cout<<"Расстояниеотвершины "<< nachalo1 + 1 <<" до "<< konec1 + 1 <<": "<< distances[konec1]<<endl;

}

else {

cout<<"Пути не существует";

}

cout<<"\n";

//Обход в глубину

cout<<"Расстояние при обходе в глубину для матрицы смежности\n";

int nachalo2;

cout<<"Начальная вершина: ";

cin>> nachalo2;

nachalo2--;

int konec2;

cout<<"Конечная вершина: ";

cin>> konec2;

konec2--;

// Нахождение расстояния при обходе в глубину

vector<int> distances2(n, -1);

GGM(graph, nachalo2, konec2, distances2);

if (distances2[konec2] != -1) {

cout<<"Расстояниеотвершины "<< nachalo2 + 1 <<" до "<< konec2 + 1 <<": "<< distances2[konec2]<<endl;

}

else {

cout<<"Путинет"<<endl;

}

// Заполнение списка смежности на основе матрицы

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

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

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

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

//list[j].push\_back(i); // добавляем обратное ребро

}

}

}

// Вывод списка смежности

cout<<"\n";

cout<<"\nСписоксмежности:\n";

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

cout<<"Вершина "<<i + 1 <<": ";

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

cout<< list[i][j] + 1 <<" ";

}

cout<<"\n";

}

cout<<"\n";

//Нахождения расстояния при обходе в ширину для списка смежности

cout<<"Расстояние при обходе в ширину для списка смежности\n";

int nachalo3;

cout<<"Исходная вершина: ";

cin>> nachalo3;

nachalo3--;

int konec3;

cout<<"Конечная вершина: ";

cin>> konec3;

konec3--;

vector<int> distances3(n, -1);

GGM\_list(list, nachalo3, konec3, distances3);

if (distances3[konec3] != -1) {

cout<<"Расстояниеотвершины "<< nachalo3 + 1 <<" до "<< konec3 + 1 <<": "<< distances3[konec3]<<endl;

}

else {

cout<<"Пути не существует";

}

// Нахождения расстояния при обходе в глубину для списка смежности

cout<<"\n";

cout<<"Расстояние при обходе в глубину для списка смежности\n";

int nachalo4;

cout<<"Начальная вершина: ";

cin>> nachalo4;

nachalo4--;

int konec4;

cout<<"Конечная вершина: ";

cin>> konec4;

konec4--;

// Поискрасстояния

vector<int> distances4(n, -1);

DFS\_list(list, nachalo4, konec4, distances4);

if (distances4[konec4] != -1) {

cout<<"Расстояниеотвершины "<< nachalo4 + 1 <<" до "<< konec4 + 1 <<": "<< distances4[konec4]<<endl;

}

else {

cout<<"Путинет"<<endl;

}

cout<<"\n";

cout<<"\n";

auto start = chrono::high\_resolution\_clock::now();

RGM(graph, nachalo1, konec1, distances);

auto end = chrono::high\_resolution\_clock::now();

auto duration = end - start;

chrono::duration<double, micro>microsec = chrono::duration\_cast<chrono::duration<double, micro>>(duration);

cout<<"Времяобходавширину: "<<microsec.count() <<" мкс"<<endl;

start=chrono::high\_resolution\_clock::now();

GGM(graph, nachalo2, konec2, distances2);

end=chrono::high\_resolution\_clock::now();

duration= end - start;

microsec=chrono::duration\_cast<chrono::duration<double, micro>>(duration);

cout<<"Времяобходавглубину: "<<microsec.count() <<" мкс"<<endl;

return 0;

}