Министерство науки и высшего образования РФ Пензенский государственный университет Кафедра «Вычислительная техника»

**ОТЧЕТ**

По лабораторной работе №9

по дисциплине «логика и основы алгоритмизации в инженерных задачах» на тему «Поиск расстояний в графе»

Выполнили: студенты группы 24ВВВ4

Королёв Д.В Алешин К.А

Приняли: Юрова О.В Деев М.В

Пенза 2024

**Цель работы** – научиться на практике реализации алгоритма поиска расстояния в графе

**Общие сведения** – Поиск расстояний – довольно распространенная задача анализа графов.

Для поиска расстояний можно использовать процедуры обхода графа. Для этого при каждом переходе в новую вершину необходимо запоминать, сколько шагов до нее мы сделали. При этом вектор, который хранил информацию о посещении вершин становится вектором расстояний. Довольно просто модернизировать для поиска расстояний в графе алгоритм обхода в ширину, т.к. этот алгоритм проходит вершины по уровням удаленности, то для не ориентированного графа для вершин каждого следующего уровня глубины расстояние от исходной вершины увеличивается на 1. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины.

**Листинг**

**Задание 1**

**1-2)**

#include <iostream>

#include <cstdlib>

#include <ctime>

#define MAX\_VERTICES 100

void generate\_adjacency\_matrix(int n, int matrix[MAX\_VERTICES][MAX\_VERTICES]) {

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

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

matrix[i][j] = 0;

}

}

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

for (int j = i + 1; j < n; j++) {

int edge = rand() % 2;

matrix[i][j] = edge;

matrix[j][i] = edge;

}

}

}

void DFSD(int v, int n, int matrix[MAX\_VERTICES][MAX\_VERTICES], int DIST[MAX\_VERTICES], int depth) {

DIST[v] = depth; // depth = расстояние

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

if (matrix[v][i] == 1 && DIST[i] == -1) {

DFSD(i, n, matrix, DIST, depth + 1);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int n;

while (1) {

std::cout << "Введите размер матрицы: ";

std::cin >> n;

if (n <= 0) {

break;

}

int matrix[MAX\_VERTICES][MAX\_VERTICES];

int DIST[MAX\_VERTICES];

srand((unsigned)time(NULL));

generate\_adjacency\_matrix(n, matrix);

std::cout << "Матрица смежности:\n";

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

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

std::cout << matrix[i][j] << " ";

}

std::cout << "\n";

}

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

DIST[i] = -1;

}

DFSD(0, n, matrix, DIST, 0);

std::cout << "Расстояния от вершины 0:\n";

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

if (DIST[i] == -1) {

std::cout << "Вершина " << i << ": недоступна\n";

}

else {

std::cout << "Вершина " << i << ": " << DIST[i] << "\n";

}

}

}

return 0;

}

**3)**

#include <iostream>

#include <queue>

struct Node {

int vertex;

Node\* next;

};

struct Graph {

int numVertices;

Node\*\* adjLists;

};

Node\* createNode(int vertex) {

Node\* newNode = new Node();

newNode->vertex = vertex;

newNode->next = nullptr;

return newNode;

}

Graph\* createGraph(int vertices) {

Graph\* graph = new Graph();

graph->numVertices = vertices;

graph->adjLists = new Node \* [vertices];

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

graph->adjLists[i] = nullptr;

}

return graph;

}

void addEdge(Graph\* graph, int src, int dest) {

Node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void BFSD(Graph\* graph, int start) {

int n = graph->numVertices;

int\* DIST = new int[n];

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

DIST[i] = -1;

}

std::queue<int> Q;

Q.push(start);

DIST[start] = 0;

while (!Q.empty()) {

int v = Q.front();

Q.pop();

Node\* temp = graph->adjLists[v];

while (temp) {

int adjVertex = temp->vertex;

if (DIST[adjVertex] == -1) {

Q.push(adjVertex);

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

}

temp = temp->next;

}

}

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

if (DIST[i] != -1) {

std::cout << "Расстояние от вершины " << start << " до вершины " << i << " равно " << DIST[i] << "\n";

}

else {

std::cout << "Вершина " << i << " недоступна из вершины " << start << "\n";

}

}

delete[] DIST;

}

int main() {

setlocale(LC\_ALL, "Rus");

int vertices = 10; // количество вершин

Graph\* graph = createGraph(vertices);

addEdge(graph, 0, 1);

addEdge(graph, 0, 4);

addEdge(graph, 1, 2); // ребро между 1 и 2

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

addEdge(graph, 2, 3);

addEdge(graph, 3, 4);

addEdge(graph, 4, 5);

addEdge(graph, 5, 7);

addEdge(graph, 2, 10);

std::cout << "Обход в ширину начиная с вершины 0:\n";

BFSD(graph, 0);

std::cout << "\n";

// Освобождение памяти

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

Node\* adjList = graph->adjLists[i];

while (adjList != nullptr) {

Node\* temp = adjList;

adjList = adjList->next;

delete temp;

}

}

delete[] graph->adjLists;

delete graph;

return 0;

}

**Задание 2**

**1)**

#include <iostream>

#include <cstdlib>

#include <ctime>

#define MAX\_VERTICES 100

void generate\_adjacency\_matrix(int n, int matrix[MAX\_VERTICES][MAX\_VERTICES]) {

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

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

matrix[i][j] = 0;

}

}

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

for (int j = i + 1; j < n; j++) {

int edge = rand() % 2;

matrix[i][j] = edge;

matrix[j][i] = edge;

}

}

}

void DFSD(int v, int n, int matrix[MAX\_VERTICES][MAX\_VERTICES], int DIST[MAX\_VERTICES], int depth) {

DIST[v] = depth; // depth = расстояние

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

if (matrix[v][i] == 1 && DIST[i] == -1) {

DFSD(i, n, matrix, DIST, depth + 1);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int n;

while (1) {

std::cout << "Введите размер матрицы: ";

std::cin >> n;

if (n == 0) {

break;

}

int matrix[MAX\_VERTICES][MAX\_VERTICES];

int DIST[MAX\_VERTICES];

srand((unsigned)time(NULL));

generate\_adjacency\_matrix(n, matrix);

std::cout << "Матрица смежности:\n";

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

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

std::cout << matrix[i][j] << " ";

}

std::cout << "\n";

}

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

DIST[i] = -1;

}

DFSD(0, n, matrix, DIST, 0);

std::cout << "Расстояния от вершины 0:\n";

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

if (DIST[i] == -1) {

std::cout << "Вершина " << i << " недоступна\n";

}

else {

std::cout << "Вершина " << i << " на расстоянии " << DIST[i] << "\n";

}

}

}

return 0;

}

**2)**

#include <iostream>

struct Node {

int vertex;

Node\* next;

};

struct Graph {

int numVertices;

Node\*\* adjLists;

};

Node\* createNode(int vertex) {

Node\* newNode = new Node();

newNode->vertex = vertex;

newNode->next = nullptr;

return newNode;

}

Graph\* createGraph(int vertices) {

Graph\* graph = new Graph();

graph->numVertices = vertices;

graph->adjLists = new Node \* [vertices];

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

graph->adjLists[i] = nullptr;

}

return graph;

}

void addEdge(Graph\* graph, int src, int dest) {

Node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void DFSD(Graph\* graph, int vertex, int\* DIST, int depth) {

DIST[vertex] = depth;

std::cout << "Расстояние от начальной вершины до вершины " << vertex << " = " << DIST[vertex] << "\n";

Node\* temp = graph->adjLists[vertex];

while (temp) {

int adjVertex = temp->vertex;

if (DIST[adjVertex] == -1) {

DFSD(graph, adjVertex, DIST, depth + 1);

}

temp = temp->next;

}

}

void DFS(Graph\* graph, int start) {

int n = graph->numVertices;

int\* DIST = new int[n];

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

DIST[i] = -1;

}

DFSD(graph, start, DIST, 0);

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

if (DIST[i] == -1) {

std::cout << "Вершина " << i << " недоступна из вершины " << start << "\n";

}

}

delete[] DIST;

}

int main() {

setlocale(LC\_ALL, "Rus");

int vertices = 10; // количество вершин

Graph\* graph = createGraph(vertices);

addEdge(graph, 0, 1);

addEdge(graph, 0, 4);

addEdge(graph, 1, 2); // ребро между 1 и 2

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

addEdge(graph, 2, 3);

addEdge(graph, 3, 4);

addEdge(graph, 4, 5);

addEdge(graph, 5, 7);

addEdge(graph, 2, 8);

std::cout << "Обход в глубину начиная с вершины 0:\n";

DFS(graph, 0);

std::cout << "\n";

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

Node\* adjList = graph->adjLists[i];

while (adjList != nullptr) {

Node\* temp = adjList;

adjList = adjList->next;

delete temp;

}

}

delete[] graph->adjLists;

delete graph;

return 0;

}

**3)**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

#include <locale.h>

#define MAX\_VERTICES 100

void generate\_adjacency\_matrix(int n, int matrix[MAX\_VERTICES][MAX\_VERTICES]) {

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

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

matrix[i][j] = 0;

}

}

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

for (int j = i + 1; j < n; j++) {

int edge = rand() % 2;

matrix[i][j] = edge;

matrix[j][i] = edge;

}

}

}

void DFSD\_iterative(int start\_vertex, int n, int matrix[MAX\_VERTICES][MAX\_VERTICES], int DIST[MAX\_VERTICES]) {

int stack[MAX\_VERTICES];

int top = -1;

stack[++top] = start\_vertex;

DIST[start\_vertex] = 0;

while (top != -1) {

int v = stack[top--];

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

if (matrix[v][i] == 1 && DIST[i] == -1) {

stack[++top] = i;

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

}

}

}

}

void BFSD(int v, int n, int matrix[MAX\_VERTICES][MAX\_VERTICES], int DIST[MAX\_VERTICES]) {

int queue[MAX\_VERTICES];

int head = 0, tail = 0;

queue[tail++] = v;

DIST[v] = 0;

while (head < tail) {

v = queue[head++];

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

if (matrix[v][i] == 1 && DIST[i] == -1) {

queue[tail++] = i;

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

}

}

}

}

void measure\_time(int n) {

setlocale(LC\_ALL, "Rus");

int matrix[MAX\_VERTICES][MAX\_VERTICES];

int DIST[MAX\_VERTICES];

generate\_adjacency\_matrix(n, matrix);

int start\_vertex = 0; // начальный узел

// Время для обхода основанного на обходе в ширину

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

DIST[i] = -1;

}

clock\_t start = clock();

BFSD(0, n, matrix, DIST);

clock\_t end = clock();

long double time\_taken\_bfsd = ((long double)(end - start)) / CLOCKS\_PER\_SEC;

// Время для обхода основанного на обходе в глубину

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

DIST[i] = -1;

}

start = clock();

DFSD\_iterative(0, n, matrix, DIST);

end = clock();

long double time\_taken\_dfsd = ((long double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Размер графа: %d, Время BFSD: %lf, Время DFSD: %lf\n", n, time\_taken\_bfsd, time\_taken\_dfsd);

}

int main() {

srand(time(NULL));

for (int n = 10; n <= 100; n += 10) {

measure\_time(n);

}

return 0;

}

Рисунок 1 – Результат выполнения 1-го задания 1-2 часть

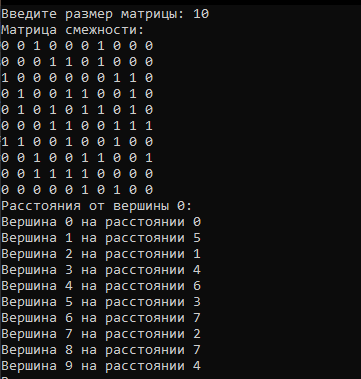


Рисунок 2 – Результат выполнения 1-го задания 3 часть

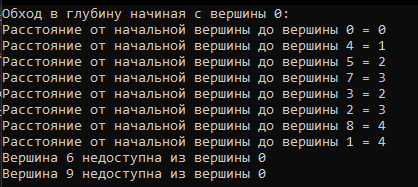


Рисунок 3 – Результат выполнения 2-го задания 1 часть

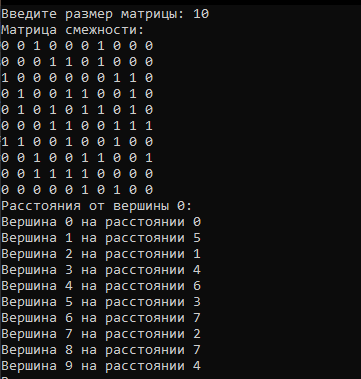


Рисунок 4 – Результат выполнения 2-го задания 2 часть

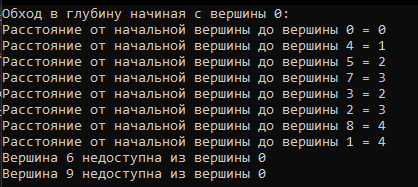
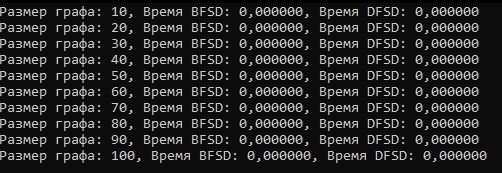


Рисунок 4 – Результат выполнения 2-го задания 3 часть



**Вывод**: В ходе выполнения лабораторной работы были изучены на практике навыки реализации алгоритма поиска путей в графе.