Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

**по курсу «Логика и основы алгоритмизации в инженерных задачах»**

на тему «Поиск расстояний во взвешенном графе»

Выполнили:

студент группы 20ВВ2

Александров В.С.

Кирюткин И.А.

Сафронов Д.В.

Принял:

к.т.н., доцент

Митрохин М.А.

**Пенза 2021**

**Цель работы:** Написать программу, которая генерирует матрицу смежности и осуществляет процедуру поиска.

**Лабораторные задания:**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного взвешенного 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.\* Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для ориентированного взвешенного графа G. Выведите матрицу

на экран и осуществите процедуру поиска расстояний, реализованную в

соответствии с приведенным выше описанием.

**Задание 2\***

1. Модернизируйте программу так, чтобы получить возможность запуска

программы с параметрами командной строки (см. описание ниже). В

качестве параметра должны указываться тип графа (взвешенный или нет) и

наличие ориентации его ребер (есть ориентация или нет).

**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <vector>

#include <queue> // очередь

using namespace std;

int orenG();

void BFS1(int mas[7][7], int DIST[7], int start)

{

queue<int> Queue;

int nodes[7]; // вершины графа

//for (int i = 0; i < 7; i++)

Queue.push(start); // помещаем в очередь первую вершину

DIST[start] = 0; // исходно все вершины равны 1000

while (!Queue.empty())

{ // пока очередь не пуста

start = Queue.front(); // извлекаем вершину

Queue.pop();

cout << start << endl; // выводим номер вершины

for (int j = 0; j < 7; j++)

{ // проверяем для нее все смежные вершины

if (mas[start][j] == 1 && DIST[j] > DIST[start] + mas[start][j])

{ // если вершина смежная и не обнаружена

Queue.push(j); // добавляем ее в очередь

//nodes[j] = 1; // отмечаем вершину как обнаруженную

DIST[j] = DIST[start] + mas[start][j];

}

}

}

}

void BFS2(int mas[5][5], int DIST[5], int start)

{

queue<int> Queue;

int nodes[5]; // вершины графа

//for (int i = 0; i < 7; i++)

Queue.push(start); // помещаем в очередь первую вершину

DIST[start] = 0; // исходно все вершины равны 1000

while (!Queue.empty())

{ // пока очередь не пуста

start = Queue.front(); // извлекаем вершину

Queue.pop();

cout << start << endl; // выводим номер вершины

for (int j = 0; j < 5; j++)

{ // проверяем для нее все смежные вершины

if (mas[start][j] == 1 && DIST[j] > DIST[start] + mas[start][j])

{ // если вершина смежная и не обнаружена

Queue.push(j); // добавляем ее в очередь

//nodes[j] = 1; // отмечаем вершину как обнаруженную

DIST[j] = DIST[start] + mas[start][j];

}

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

int graph[7][7];

char p1[5];

cout << "Введите тип графа (ОВ или НН или ОН) \n";

cin >> p1;

if (strcmp(p1, "ОВ")==0) {

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

graph[i][j] = rand() % 5;

}

}

}

if (strcmp(p1, "ОН") == 0) {

for (int i = 0; i < 7; i++) {

for (int j = 0; j < i + 1; j++) {

int num = rand() % 5;

graph[i][j] = num;

graph[j][i] = num;

}

}

}

if (strcmp(p1, "НН") == 0) {

for (int i = 0; i < 7; i++) {

for (int j = 0; j < i + 1; j++) {

int num = rand() % 2;

graph[i][j] = num;

graph[j][i] = num;

}

}

}

for (int i = 0; i < 7; i++) {

for (int j = 0; j <7; j++) {

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

}

printf("\n");

}

int start;

cout << "Введите начальную вершину \n";

cin >> start;

int DIST[7];

for (int i = 0; i < 7; i++) {

DIST[i] = 1000;

}

cout << "обход в ширину\n";

BFS1(graph, DIST, start);

for (int i = 0; i < 7; i++) {

printf("рассояние от %d до %d = %d\n", start, i, DIST[i]);

}

}

**Псевдокод:**

Вход: mas[7][7], DIST[7], start

Выход: Номер вершины через start

Алгоритм BFS1:

Помещаем в очередь первую вершину

DIST[start] = 0

Пока очередь не пуста

start = Queue.front()

выводим номер вершины

Для j От 0 До 7

Проверяем для нее все смежные вершины

Если (mas[start][j] = 1 И DIST[j] > DIST[start] + mas[start][j])

добавляем вершину в очередь

отмечаем вершину как обнаруженную

DIST[j] = DIST[start] + mas[start][j];

Вход: mas[5][5], DIST[5], start

Выход: Номер вершины через start

Алгоритм BFS2:

Помещаем в очередь первую вершину

DIST[start] = 0

Пока очередь не пуста

start = Queue.front()

выводим номер вершины

Для j От 0 До 5

Проверяем для нее все смежные вершины

Если (mas[start][j] = 1 И DIST[j] > DIST[start] + mas[start][j])

добавляем вершину в очередь

отмечаем вершину как обнаруженную

DIST[j] = DIST[start] + mas[start][j];

Вход: p1, start, DIST[7]

Выход: graph[i][j], start, DIST

Алгоритм main:

Если (strcmp(p1, "ОВ") =0)

Для i От 0 До 7

Для j От 0 До 7

graph[i][j] = rand() % 5

Если (strcmp(p1, "ОН") = 0) {

Для i От 0 До 7

Для j От 0 До 7

num = rand() % 5

graph[i][j] = num

graph[j][i] = num

Если (strcmp(p1, "НН") = 0)

Для i От 0 До 7

Для j От 0 До 7

num = rand() % 2

graph[i][j] = num

graph[j][i] = num

Для i От 0 До 7

DIST[i] = 1000

Вызов функции BFS1

**Результаты программы:**

**Вывод:** В результате обучения мы написали программу, которая генерирует матрицу смежности и осуществляет процедуру поиска расстояния.