Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №10

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний во взвешенном графе»

Выполнили ст. группы 22ВВВ1:

Колобов И.О.

Лёвин А.Д.

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Научиться пользоваться алгоритмом поиска расстояний во взвешенном графе.

**Лабораторное задание:**

**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. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

**Задание 3\***

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).

.

**Ход работы:**

**Создание графа**

**1.** Алгоритм генерации графа в зависимости от указанных параметров (-w и -d):

Программа создает граф заданного размера (введенного пользователем).

Для каждой пары вершин, отличных друг от друга (i != j):

Если указан параметр -w (взвешенный граф), устанавливает случайный вес на ребре от i к j.

Если не указан -w (не взвешенный граф), генерирует 0 или 1 для определения наличия ребра.

Если не указан -d (ненаправленный граф), делает рёбра симметричными.

**2.** Вывод графа

Выводит матрицу смежности и списки смежности созданного графа.

**3.** Поиск в ширину (RGM)

Функция RGM реализует алгоритм поиска в ширину для нахождения кратчайших путей от стартовой вершины ко всем остальным.

Использует очередь для обхода вершин по уровням и обновляет массив visited с расстояниями до каждой вершины.

**4.** Нахождение радиуса, диаметра и эксцентриситетов вершин

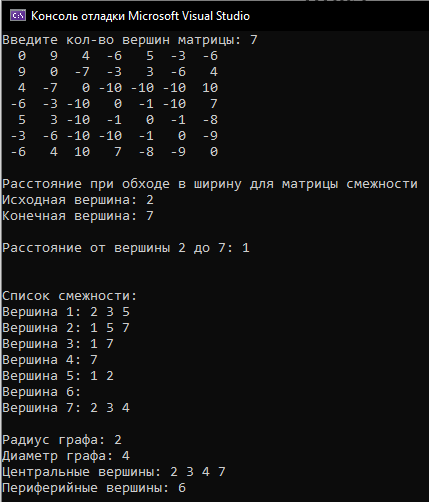
В меню выполнено вычисление эксцентриситетов для каждой вершины с помощью обхода графа с помощью BFS.

Эксцентриситет вершины - это максимальное расстояние от этой вершины до всех остальных вершин.

Радиус - минимальный эксцентриситет.

Диаметр - максимальный эксцентриситет.

**Результаты работы программы:**



**Вывод:**

Лабораторная работа позволила понять и применить алгоритм поиска кратчайшего пути в графе с весом на рёбрах. Код программы, реализованный в рамках лабораторной работы, позволяет генерировать взвешенные графы, хранить их представление в виде матрицы смежности или списков смежности, а также выполнять поиск кратчайших путей с помощью алгоритма поиска в ширину. Эта работа помогла углубить понимание работы с взвешенными графами и их особенностей, а также научиться вычислять и использовать кратчайшие пути в других задачах.

**Листинг:**

#include <iostream>

#include <cstdlib>

#include <stack>

#include <vector>

#include <algorithm>

#include <queue>

#include <cstring>

#include <iomanip> // Для использования std::setw

using namespace std;

struct GraphSettings {

bool weighted;

bool directed;

};

//Задаём параметры

GraphSettings arguments(int argc, char\* argv[]) {

GraphSettings settings{ false, false };

for (int i = 1; i < argc; i++) {

if (strcmp(argv[i], "-w") == 0) {

settings.weighted = true;

}

else if (strcmp(argv[i], "-d") == 0) {

settings.directed = true;

}

}

return settings;

}

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

void RGM(const vector<vector<int>>& graph, int nachalo1, int konec1, 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 (int i = 0; i < graph[v].size(); i++) {

int to = graph[v][i];

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

q.push(to);

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

}

}

}

}

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

GraphSettings graphSettings = arguments(argc, argv);

int n;

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

cin >> n;

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

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

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

int\*\* adjMatrix = new int\* [n];

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

adjMatrix[i] = new int[n];

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

if (i == j) adjMatrix[i][j] = 0;

else adjMatrix[i][j] = rand() % 10;

}

}

//Генерация рёбер для различных параметров

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

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

if (i == j) {

adjMatrix[i][j] = 0;

}

else {

if (graphSettings.weighted == 1 && graphSettings.directed == 1) {

adjMatrix[i][j] = (-10) + rand() % 21;

adjMatrix[j][i] = adjMatrix[i][j];

}

if (graphSettings.weighted == 0 && graphSettings.directed == 1) {

adjMatrix[i][j] = (-1) + rand() % 3;

adjMatrix[j][i] = adjMatrix[i][j];

}

if (graphSettings.weighted == 1 && graphSettings.directed == 0) {

adjMatrix[i][j] = rand() % 10;

adjMatrix[j][i] = adjMatrix[i][j];

}

if (graphSettings.weighted == 0 && graphSettings.directed == 0) {

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

adjMatrix[j][i] = adjMatrix[i][j];

}

}

}

}

// вывод матрицы

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

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

std::cout << std::setw(3) << adjMatrix[i][j] << " ";

}

cout << endl;

}

// Заполнение графа на основе матрицы смежности

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

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

if (adjMatrix[i][j] > 0) {

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

}

}

}

//Нахождения расстояния при обходе в ширину для матрицы смежности

cout << "\nРасстояние 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 << "\nРасстояние от вершины " << nachalo1 + 1 << " до " << konec1 + 1 << ": " << distances[konec1] << endl;

cout << "\n";

}

else {

cout << "\nПути не существует\n";

}

// Заполнение списка смежности на основе матрицы

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

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

if (adjMatrix[i][j] > 0) {

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

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

}

}

}

// Вывод списка смежности

cout << "\nСписок смежности:\n";

for (int i = 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";

//Радиус, диаметр и вершины

vector<int> eccentricities(n);

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

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

fill(visited.begin(), visited.end(), -1);

RGM(graph, start, -1, visited);

int max\_distance = 0;

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

if (visited[i] > max\_distance) {

max\_distance = visited[i];

}

}

eccentricities[start] = max\_distance;

}

int diameter = 0;

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

if (eccentricities[i] > diameter) {

diameter = eccentricities[i];

}

}

int radius = diameter;

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

if (eccentricities[i] < radius) {

radius = eccentricities[i];

}

}

cout << "Радиус графа: " << radius << "\n";

cout << "Диаметр графа: " << diameter << "\n";

cout << "Центральные вершины: ";

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

if (eccentricities[i] == radius) {

cout << i + 1 << " ";

}

}

cout << "\n";

cout << "Периферийные вершины: ";

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

if (eccentricities[i] == diameter) {

cout << i + 1 << " ";

}

}

cout << "\n";

// освобождение памяти

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

delete[] adjMatrix[i];

}

delete[] adjMatrix;

return 0;

}