Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №10

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Поиск расстояний во взвешенном графе**»

Выполнили:

студенты группы 23ВВВ4

Брагин А.М.

Зарубин Я.

Герасимов К.

Приняли:

Деев М.В.

Юрова О.В.

Пенза 2024

**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.

Рисунок 1 – Граф

Таким образом, можно предложить следующую реализацию алгоритма

обхода в ширину.

Вход: G – матрица смежности графа, v – исходная вершина.

Выход: DIST – вектор расстояний до всех вершин от исходной.

Алгоритм ПОШ

1.1. для всех i положим DIST [i] = -1 пометим как &quot;не посещенную&quot;;

1.2. ВЫПОЛНЯТЬ BFSD (v).

1.3 для всех i вывести DIST [i] на экран;

Алгоритм BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний DIST [ x ] = 0;

2.4. ПОКА Q != Æ очередь не пуста ВЫПОЛНЯТЬ

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.9. ЕСЛИ G(v,i) = = 1И DIST = = -1

2.10. ТО

2.11. Поместить i в очередь Q.push(i);

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] + 1;

Реализация состоит из подготовительной части, в которой все вершины

помечаются как не поcещенные (п.1.1). В отличие от алгоритма BFS не

посещенные вершины помечаем -1, т.к. значение 0 и 1 могут быть

расстояниями. Расстояние 0 – от исходной вершины до самой себя.

В самой процедуре как и в алгоритме BFS сначала создается пустая

очередь (п. 2.1), в которую помещается исходная вершина, из которой начат

обход (п.2.2). Расстояние до этой вершины (п.2.3) устанавливается равным 0

(расстояние до самой себя).

Далее итерационно, пока очередь не опустеет, из нее извлекается

первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в

цикле просматривается v-я строка матрицы смежности графа G(v,i). Как

только алгоритм встречает смежную с v не посещенную вершину (п.2.9), эта

вершина помещается в очередь (п.2.11) и для нее обновляется вектор

расстояния (п.2.12). Расстояние до новой i-й вершины вычисляется как

расстояние до текущей v-й вершины плюс 1 (так как ребра нашего графа не

взвешенные).

После просмотра строки матрицы смежности алгоритм делает

следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Таким образом, если вершина помещается в очередь при просмотре

сроки матрицы смежности на 1-й итерации, то они находятся на 1 уровне

удаленности и расстояние до этих вершин будет равным 1.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 0 – расстояние от исходной

вершины до самой себя.

Далее, начинают просматриваться вершины первого уровня и

соответствующие им строки матрицы смежности. При добавлении смежных

с вершинами первого уровня вершин, расстояния до них будут равны 2.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 1 – расстояние от исходной

вершины до вершин 1 уровня.

После того, как все вершины первого уровня будут просмотрены и

извлечены из очереди, начнется просмотр вершин 2 уровня и

соответствующих им строк матрицы смежности. При добавлении смежных с

вершинами второго уровня вершин, расстояния до них будут равны 3.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 2 – расстояние от исходной

вершины до вершин 2 уровня.

И так далее, алгоритм проходит вершины по уровням, пока очередь не

опустеет.

**Практическая часть:**

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.

**Листинг**

#include <iostream>

#include <vector>

#include <queue>

#include <limits>

#include <cstdlib>

#include <ctime>

#define INF std::numeric\_limits<int>::max()

// Функция для генерации случайной матрицы смежности для неориентированного графа

void generateUndirectedGraph(std::vector<std::vector<int>>& matrix, int n, int max\_weight) {

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

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

if (i == j) {

matrix[i][j] = 0;

} else {

int weight = rand() % max\_weight + 1;

matrix[i][j] = matrix[j][i] = weight;

}

}

}

}

// Функция для генерации случайной матрицы смежности для ориентированного графа

void generateDirectedGraph(std::vector<std::vector<int>>& matrix, int n, int max\_weight) {

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

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

if (i == j) {

matrix[i][j] = 0;

} else {

int weight = rand() % max\_weight + 1;

matrix[i][j] = weight;

}

}

}

}

// Функция для вывода матрицы смежности

void printMatrix(const std::vector<std::vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int value : row) {

if (value == INF) {

std::cout << " INF";

} else {

std::cout << " " << value;

}

}

std::cout << "\n";

}

}

// Алгоритм Дейкстры с использованием std::queue

void dijkstra(const std::vector<std::vector<int>>& matrix, int start, std::vector<int>& dist) {

int n = matrix.size();

std::vector<bool> visited(n, false);

dist.assign(n, INF);

dist[start] = 0;

std::queue<int> queue;

queue.push(start);

while (!queue.empty()) {

int u = queue.front();

queue.pop();

if (visited[u]) continue;

visited[u] = true;

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

if (!visited[v] && matrix[u][v] != INF && dist[u] + matrix[u][v] < dist[v]) {

dist[v] = dist[u] + matrix[u][v];

queue.push(v);

}

}

}

}

// Вычисление радиуса, диаметра, центральных и периферийных вершин

void calculateGraphProperties(const std::vector<std::vector<int>>& matrix) {

int n = matrix.size();

std::vector<int> eccentricity(n);

int radius = INF, diameter = 0;

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

std::vector<int> dist;

dijkstra(matrix, i, dist);

int max\_dist = 0;

for (int d : dist) {

if (d != INF && d > max\_dist) {

max\_dist = d;

}

}

eccentricity[i] = max\_dist;

if (max\_dist < radius) radius = max\_dist;

if (max\_dist > diameter) diameter = max\_dist;

}

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

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

std::cout << "Центральные вершины: ";

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

if (eccentricity[i] == radius) {

std::cout << i << " ";

}

}

std::cout << "\n";

std::cout << "Периферийные вершины: ";

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

if (eccentricity[i] == diameter) {

std::cout << i << " ";

}

}

std::cout << "\n";

}

int main() {

srand(time(NULL));

int n, max\_weight;

std::cout << "Введите количество вершин графа: ";

std::cin >> n;

std::cout << "Введите максимальный вес ребра: ";

std::cin >> max\_weight;

std::vector<std::vector<int>> undirectedGraph(n, std::vector<int>(n, INF));

std::vector<std::vector<int>> directedGraph(n, std::vector<int>(n, INF));

std::cout << "Матрица смежности для неориентированного графа:\n";

generateUndirectedGraph(undirectedGraph, n, max\_weight);

printMatrix(undirectedGraph);

calculateGraphProperties(undirectedGraph);

std::cout << "\nМатрица смежности для ориентированного графа:\n";

generateDirectedGraph(directedGraph, n, max\_weight);

printMatrix(directedGraph);

calculateGraphProperties(directedGraph);

return 0;

}

**Результат работы программы:**

