НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Механико-математический факультет
Кафедра: Математика и компьютерные науки

Тлепбергенова Дарья Дулатовна

Отчет по предмету: Графы и алгоритмы

Алгоритм Дейкстры.
Поиск минимального пути в графе.

3 курс, группа 16121

Преподаватель:
Хомякова Екатерина Николаевна

Новосибирск, 2018 г.

1. Постановка задачи.

Для заданного взвешенного графа $G = (V, E)$ найти кратчайшие пути из заданной вершины s до всех остальных вершин. Веса всех рёбер неотрицательны.

2. Описание алгоритма.

Заведём массив $d[]$, в котором для каждой вершины v будем хранить текущую длину $d[v]$ кратчайшего пути из s в v. Изначально $d[s] = 0$, а для всех остальных вершин эта длина равна бесконечности (при реализации на компьютере обычно в качестве бесконечности выбирают просто достаточно большое число, заведомо большее возможной длины пути).

$$d[v] = \infty, v \neq s$$

Кроме того, для каждой вершины v будем хранить, помечена она ещё или нет, т.е. заведём булевский массив $u[]$. Изначально все вершины не помечены, т.е.

$$u[v] = false$$

Сам алгоритм Дейкстры состоит из n итераций. На очередной итерации выбирается вершина v с наименьшей величиной $d[v]$ среди ещё не помеченных, т.е.:

$$d[v] = \min_{p:u[p]=false} d[p]$$

(Понятно, что на первой итерации выбрана будет стартовая вершина s.) Выбранная таким образом вершина v отмечается помеченной. Далее, на текущей итерации, из вершины v производятся релаксации: просматриваются все рёбра (v, t_o), исходящие из вершины v, и для каждой такой вершины t_o алгоритм пытается улучшить значение $d[t_o]$. Пусть длина текущего ребра равна len, тогда в виде кода релаксация выглядит как:

$$d[t_o] = \min(d[t_o], d[v] + len)$$

На этом текущая итерация заканчивается, алгоритм переходит к следующей итерации (снова выбирается вершина с наименьшей величиной d, из неё производятся релаксации, и т.д.). При этом в конце концов, после n итераций, все вершины графа станут помеченными, и алгоритм свою работу завершает. Утверждается, что найденные значения $d[v]$ и есть искомые длины кратчайших путей из s в v.

Стоит заметить, что, если не все вершины графа достижимы из вершины s, то значения $d[v]$ для них так и останутся бесконечными.

Для простоты решения, возьмем $s = 1$ и будем рассматривать все минимальные пути из первой вершины во все остальные.

3. Код программы (на Java)

3.1. Main класс с тестами:

```
package ru.nsu.mmf.g16121.ddt.math;

import ru.nsu.mmf.g16121.ddt.main.Graph;
import java.util.Arrays;
import java.util.Scanner;

public class Dijkstra {
    public static void main(String[] args) {
        //Ex 1
        //обыкновенный граф
        double[][] adjacencyMatrix = {
                {0, 7, 9, 0, 0, 14},
                {7, 0, 10, 15, 0, 0},
                {9, 10, 0, 11, 0, 2},
                {0, 15, 11, 0, 6, 0},
                {0, 0, 0, 6, 0, 9},
                {14, 0, 2, 0, 9, 0}};

        Graph graph1 = new Graph(adjacencyMatrix);

        double[] minDistance = graph1.getMinDistance();
        System.out.println("Массив минимальных расстояний вершин:");
        System.out.println(Arrays.toString(minDistance));

        Scanner scanner = new Scanner(System.in);

        System.out.print("Введите номер вершины, до которой хотите" +
                " постороить мин путь (отсчет с 1): ");
        int endVertex = scanner.nextInt();

        graph1.printMinPath(endVertex);
        System.out.println();

        //Ex 2
        //ориентированный граф
        double[][] adjacencyMatrix2 = {
                {0, 10, 30, 50, 10},
                {0, 0, 0, 0, 0},
                {0, 0, 0, 0, 10},
                {0, 40, 20, 0, 0},
```

```
        {10, 0, 10, 30, 0}};

Graph graph2 = new Graph(adjacencyMatrix2);
minDistance = graph2.getMinDistance();
System.out.println("Массив минимальных расстояний вершин:");
System.out.println(Arrays.toString(minDistance));

System.out.print("Введите номер вершины, до которой хотите" +
        " постороить мин путь (отсчет с 1): ");
endVertex = scanner.nextInt();

graph2.printMinPath(endVertex);
System.out.println();

//Ex 3
double[][] adjacencyMatrix3 = {
        {0,2,1,4,0,0},
        {2,0,0,7,2.5,0},
        {0,0,0,5,10,4},
        {0,0,0,0,0,5},
        {0,0,0,0,0,4},
        {0,0,0,0,0,0}
};
Graph graph3 = new Graph(adjacencyMatrix3);
minDistance = graph3.getMinDistance();
System.out.println("Массив минимальных расстояний вершин:");
System.out.println(Arrays.toString(minDistance));

System.out.print("Введите номер вершины, до которой хотите" +
        " постороить мин путь (отсчет с 1): ");
endVertex = scanner.nextInt();

graph3.printMinPath(endVertex);
System.out.println();

//Ex 4
//с изолированными вершинами
double[][] adjacencyMatrix4 = {
        {0,0,0},
        {0,0,0},
        {0,0,0},
};
Graph graph4 = new Graph(adjacencyMatrix4);
minDistance = graph4.getMinDistance();
```

```
        System.out.println("Массив минимальных расстояний вершин:");
        System.out.println(Arrays.toString(minDistance));

        System.out.print("Введите номер вершины, до которой хотите" +
                " постороить мин путь (отсчет с 1): ");
        endVertex = scanner.nextInt();

        graph4.printMinPath(endVertex);
        System.out.println();
    }
}
```

3.2. Класс Graph:

```
    package ru.nsu.mmf.g16121.ddt.main;

/**
 * Класс Граф, основанный на матрице смежности, вершины отсчитываются
 * начиная с нулевой.
 */
public class Graph {
    private double[][] paths;
    private final int vertexCount;

    private static final double infinity = 10E5;
    private static final int infinityForInt = (int) infinity;
    private static final double eps = 1 / infinity;

    /**
     * Конструктор графа
     *
     * @param paths - матрица смежности
     */
    public Graph(double[][] paths) {
        vertexCount = paths.length;
        this.paths = new double[vertexCount][vertexCount];
        for (int i = 0; i < vertexCount; ++i) {
            System.arraycopy(paths[i], 0, this.paths[i], 0,
            vertexCount);
        }
    }

    /**
     * Поиск минимальных расстояний от 0 вершины до всех остальных с
```

```
 *помощью алгорима Дейкстры.
 *
 * @return - возвращает матрицу минимальных расстояний до каждой
 *из вершин
 */
public double[] getMinDistance() {
    double[] minDistance = new double[vertexCount];
    boolean[] visitedVertices = new boolean[vertexCount];

    //Изначальное объявление
    for (int i = 0; i < vertexCount; ++i) {
        minDistance[i] = infinity;
        visitedVertices[i] = false;
    }

    minDistance[0] = 0;
    int minIndexNow;
    double minDistanceNow;

    //шаг алгоритма
    do {
        minIndexNow = infinityForInt;
        minDistanceNow = infinity;

        for (int i = 0; i < vertexCount; ++i) {
            if (!visitedVertices[i] &&
            ((minDistance[i] - infinity) < -eps)) {
                minDistanceNow = minDistance[i];
                minIndexNow = i;
                break;
            }
        }

        if (minIndexNow != infinityForInt) {
            for (int i = 0; i < vertexCount; ++i) {
                if (paths[minIndexNow][i] > eps) {
                    double support =
                    minDistanceNow + paths[minIndexNow][i];
                    if ((support - minDistance[i]) < -eps) {
                        minDistance[i] = support;
                    }
                }
            }
            //отмечаем, что поситили  вершину
```

```
            visitedVertices[minIndexNow] = true;
        }
    } while (minIndexNow < infinityForInt);

    return minDistance;
}

/**
 * Данный метод выполняет поиск минимального пути и печатает
 * минимальный путь (по вершинам)
 * с 1 й вершины до @param endVertex
 */
public void printMinPath(int endVertex) {
    int[] minPath = new int[vertexCount];
    minPath[0] = endVertex;//так как отсчет вершин ведется с 0
    double[] minDistance = this.getMinDistance();
    double weight = minDistance[endVertex - 1];
    int endVertexNow = endVertex - 1;
    int vertexCountInPath = 1;
    boolean forStop = false;

    //записываем в массив minPath номера вершин через которые
    //проходит мин путь
    while (endVertexNow > 0) {
        forStop = false;
        for (int i = 0; i < vertexCount; ++i) {
            if (Math.abs(paths[i][endVertexNow]) >= eps) {
                double support = weight - paths[i][endVertexNow];
                if (Math.abs(support - minDistance[i]) < eps) {
                    forStop = true;
                    weight = support;
                    endVertexNow = i;
                    minPath[vertexCountInPath] = i + 1;
                    ++vertexCountInPath;
                }
            }
        }
        if (!forStop){
            System.out.println("пути нет");
            return;
        }
    }

    // выводим масив minPath на экран
```

```
        for (int i = vertexCountInPath - 1; i >= 0; --i) {
            System.out.print(minPath[i] + " ");
        }
    }
}
```

3.3. Выводы

Благодаря произведенным исследованиям алгоритма Дейкстры составили и запрограммировали поиск минимального пути в графе и вывод минимального пути.

Для решения поставленной задачи был применен математический аппарат теории графов, которая имеет широкое применение в Вооруженных Силах не только для поиска кратчайшего расстояния, но и для принятия оптимального решения и т.д. Рассмотренный метод Дейкстры является одним из самых быстрых для поиска кратчайших расстояний от некоторой вершины до всех остальных. Он лёгок для понимания и способен давать достаточно точные результаты.

Основные результаты работы сводятся к тому, что указывается важная роль применения теории графов в Вооруженных Силах, формулируются основные направления развития существующих методов, обуславливается выбор метода Дейкстры, разрабатывается математическая модель и проводится её экспериментальное исследование.