Lab 3
Shortest Paths Algorithms

1 Overview

In this assignment, you're required to implement two shortest paths algorithms which are Dijkstra and Bellman-Ford.

2 Dijkstra Algorithm

This algorithm finds shortest paths from the source to all other nodes in the graph, producing a shortest path tree. Its time complexity is $O(|V|^2)$ but can reach less than that when using priority queue. Dijkstra algorithm can't handle negative weights. But, it is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.

3 Bellman-Ford Algorithm

The Bellman-Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is capable of handling graphs in which some of the edge weights are negative numbers. It works in $O(|V||E|)$ time and $O(|V|)$ space complexities where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph.

4 Input Graph Structure

Input file will contain several lines that describe a **directed** graph structure as follows. First line contains two integers **V** and **E** which determine number of vertices and edges respectively. This line is followed by **E** lines describing the edges in the graph. Each of the **E** lines contain 3 numbers: i, j, w separated by a single space, meaning that there is a weighted edge from vertex i to vertex j $(0 \leq i, j \leq V - 1)$, and the weight of the edge is w, where w may be negative or positive.

5 Interfaces

```
package eg.edu.alexu.csd.filestructure.graphs;

import java.util.ArrayList;

public interface IGraph {
        /**
         * Constructs a graph with the number of vertices and set of edges
         * provided in the file.
         * Input file will contain several lines that describe the graph structure as follows.
         * First line contains two integers V and E which determine number of vertices and
         * edges respectively. This line is followed by E lines describing the edges in
         * the graph. Each of the E lines contain 3 numbers: i, j, w separated by a single
         * space. i,j represents the nodes that are connected through this edge (0 <= i, j < V)
         * and w determines the weight on this edge. Weights may be negative or positive.
         * @param file        The graph file
         */
        public void readGraph(java.io.File file);

        /**
         * Return the size of the graph
         * @return        Size of the graph
         */
        public int size();

        /**
         * Returns the set of vertices in the graph
         * @return        List of vertices
         */
        public ArrayList<Integer> getVertices();

        /**
         * Returns a list of the neighboring vertices to the vertex v
         * @param v        ID of a vertex in the graph
         * @return         A list of vertices adjacent to v
         */
        public ArrayList<Integer> getNeighbors(int v);

        /**
         * Runs the Dijkstra single-source shortest path algorithm on the graph
         * @param src                ID of the source vertex for the shortest path algorithm
         * @param distances        An array to be filled with distances from src to all vertices in the graph
         */
        public void runDijkstra(int src, int[] distances);

        /**
         * Returns the order of vertices with which Dijkstra processed them
         * @return         List of vertices ordered processed by Dijkstra's SSSP algorithm
         */
        public ArrayList<Integer> getDijkstraProcessedOrder();

        /**
         * Runs the Bellman-Ford single-source shortest path algorithm on the graph
         * @param src        ID of the source vertex for the shortest path algorithm
         * @param distances        An array to be filled with distances from src to all vertices in the graph
         * @return                False if the graph contains a negative cycle. True otherwise.
         */
        public boolean runBellmanFord(int src, int[] distances);

}
```

6 Notes

- You **must work in groups** of two or three
- You will need to submit a report including; problem statement, pseudo code or flowchart describing the algorithms used and data structures used, important code snippets and sample runs.
- Try very hard to clean up your implementation. Remove all unused variables. Do not write redundant and repeated code. You may use Checkstyle with your IDE to ensure that your code style follows the JAVA coding style standards

7 References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.