Name: Olorunfemi-Ojo Daniel Tomiwa
Matric No: 190805503
Course: CSC 320

Bellman Ford's Algorithm Pseudocode

```
for i = 1 to (number of vertices in graph) – 1
   foreach vertex in graph
      foreach edge connected to vertex
         if vertex weight + edge value < weight of vertex connected to edge
            weight of vertex connected to edge = vertex weight + edge value
```

Bellman-Ford's algorithm is an algorithm for finding the shortest path between a source node and all other nodes in a weighted directed graph. It runs n -1 times, where n = number of nodes in the graph. In every iteration, the algorithm goes over edges connected to the node in focus and updates the distance to the nodes connected via those edges if the resulting path is shorter. At the start, the first node is assigned a weight of 0 and others, infinity. If the node in focus weight plus the distance of the edge to the connected node is greater than the connected node's weight, the connected node is assigned the node in focus weight plus the distance of the edge. It supports negative weights. It has a time complexity of O(V * E).

Dijkstra's Algorithm Pseudocode

```
Select the first vertex
for i = 1 to (number of vertices in graph) -1
   foreach edge connected to the selected vertex
      if selected vertex weight + edge value < weight of vertex connected to
edge and vertex connected to edge has not been chosen before
         weight of vertex connected to edge = vertex weight + edge value

   Select the vertex with the lowest weight that has not been selected before
```

Dijkstra's algorithm is another algorithm for finding the shortest path between a start node and other nodes in weighted graph. Unlike Bellman Ford's, a negative weight may result in wrong answers. It iterates n-1 times and is a greedy algorithm. The start node is assigned a weight of 0 and others, infinity. The algorithm selects the node with the lowest value that has not been selected before and checks if the weight plus the edge distance between the connected nodes is greater than the connected nodes weight. If it is less than the connected nodes weight, the connected node is assigned the value. It has a time complexity of O((E+V)*LogV) = O(ELogV).

Implementation

```java
package com.ot.grephq;

import java.util.ArrayList;
import java.util.List;

/**
 * Representation of a node in a graph
```

```java
 */
public class Vertex {

	private char id;
	private int weight;
	private List<Edge> edges;

	public Vertex(char id, int weight) {
		this.id = id;
		this.weight = weight;
		this.edges = new ArrayList<Edge>();
	}

	public Vertex(char id, int weight, List<Edge> edges) {
		this.id = id;
		this.weight = weight;
		this.edges = edges;
	}

	public char getId() {
		return id;
	}

	public int getWeight() {
		return weight;
	}

	public void setWeight(int weight) {
		this.weight = weight;
	}

	public List<Edge> getEdges(){
		return edges;
	}

	public void addEdge(Edge edge) {
		edges.add(edge);
	}
}


package com.ot.grephq;

/**
 * Representation of the edge between two vertices in a graph
 */
public class Edge {

	private Vertex vertex;
	private int weight;

	public Edge(Vertex vertex, int weight) {
		this.vertex = vertex;
		this.weight = weight;
	}
```

```
	public int getWeight() {
		return weight;
	}

	public Vertex getVertex() {
		return vertex;
	}
}
```

```
package com.ot.grephq;

import java.util.List;

/**
 * Representation of a graph
 */
public class Graph {

	private List<Vertex> vertices;

	public Graph(List<Vertex> vertices) {
		this.vertices = vertices;
	}

	public List<Vertex> getVertices(){
		return vertices;
	}
}
```

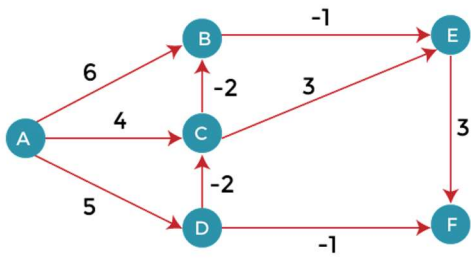
```
package com.ot.grephq;

import java.util.ArrayList;
import java.util.Arrays;r
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CSC320 {

	public static void main(String[] args) {

		Graph graph = createGraphForBellmanFord();
		bellmanFordAlgorithm(graph);
		System.out.println("Bellman Ford");
		System.out.println("------------");
		for(Vertex vertex : graph.getVertices()) {
			System.out.println(vertex.getId() + ": " + vertex.getWeight());
		}

		graph = createGraphForDijkstra();
		dijkstraAlgorithm(graph);
		System.out.println("\nDijkstra");
```

```
		System.out.println("----------");
		for(Vertex vertex : graph.getVertices()) {
			System.out.println(vertex.getId() + ": " + vertex.getWeight());
		}
	}

	/**
	 * Create a graph for Bellman Ford's algorithm
	 *
	 * @return Graph
	 */
	public static Graph createGraphForBellmanFord() {
		Vertex a = new Vertex('A', 0);
		Vertex b = new Vertex('B', Integer.MAX_VALUE);
		Vertex c = new Vertex('C', Integer.MAX_VALUE);
		Vertex d = new Vertex('D', Integer.MAX_VALUE);
		Vertex e = new Vertex('E', Integer.MAX_VALUE);
		Vertex f = new Vertex('F', Integer.MAX_VALUE);

		a.addEdge(new Edge(b, 6));
		a.addEdge(new Edge(c, 4));
		a.addEdge(new Edge(d, 5));

		b.addEdge(new Edge(e, -1));

		c.addEdge(new Edge(b, -2));
		c.addEdge(new Edge(e, 3));

		d.addEdge(new Edge(c, -2));
		d.addEdge(new Edge(f, -1));

		e.addEdge(new Edge(f, 3));

		List<Vertex> vertices = new ArrayList<Vertex>(Arrays.asList(a, b, c, d,
e, f));

		return new Graph(vertices);
	}

	/**
	 * Create a graph for Dijkstra's algorithm
	 *
	 * @return
	 */
	public static Graph createGraphForDijkstra() {
		Vertex a = new Vertex('0', 0);
		Vertex b = new Vertex('1', Integer.MAX_VALUE);
		Vertex c = new Vertex('2', Integer.MAX_VALUE);
		Vertex d = new Vertex('3', Integer.MAX_VALUE);
		Vertex e = new Vertex('4', Integer.MAX_VALUE);
		Vertex f = new Vertex('5', Integer.MAX_VALUE);
		Vertex g = new Vertex('6', Integer.MAX_VALUE);
		Vertex h = new Vertex('7', Integer.MAX_VALUE);
		Vertex i = new Vertex('8', Integer.MAX_VALUE);
```

```
		a.addEdge(new Edge(b, 4));
		a.addEdge(new Edge(h, 8));

		b.addEdge(new Edge(a, 4));
		b.addEdge(new Edge(c, 8));
		b.addEdge(new Edge(h, 11));

		c.addEdge(new Edge(b, 8));
		c.addEdge(new Edge(d, 7));
		c.addEdge(new Edge(f, 4));
		c.addEdge(new Edge(i, 2));

		d.addEdge(new Edge(c, 7));
		d.addEdge(new Edge(e, 9));
		d.addEdge(new Edge(f, 14));

		e.addEdge(new Edge(d, 9));
		e.addEdge(new Edge(f, 10));

		f.addEdge(new Edge(c, 4));
		f.addEdge(new Edge(d, 14));
		f.addEdge(new Edge(e, 10));
		f.addEdge(new Edge(g, 2));

		g.addEdge(new Edge(f, 2));
		g.addEdge(new Edge(h, 1));
		g.addEdge(new Edge(i, 6));

		h.addEdge(new Edge(a, 8));
		h.addEdge(new Edge(b, 11));
		h.addEdge(new Edge(g, 1));
		h.addEdge(new Edge(i, 7));

		i.addEdge(new Edge(c, 2));
		i.addEdge(new Edge(g, 6));
		i.addEdge(new Edge(h, 7));

		List<Vertex> vertices = new ArrayList<Vertex>(Arrays.asList(a, b, c, d,
e, f, g, h, i));

		return new Graph(vertices);
	}

	/**
	 * Implementation of Bellman Ford's algorithm
	 *
	 * @param graph
	 */
	public static void bellmanFordAlgorithm(Graph graph) {
		for(int i = 1; (i <= graph.getVertices().size() - 1); i++) {
			for(Vertex vertex : graph.getVertices()) {
				for(Edge edge : vertex.getEdges()) {
					if((vertex.getWeight() + edge.getWeight()) <
edge.getVertex().getWeight())
```

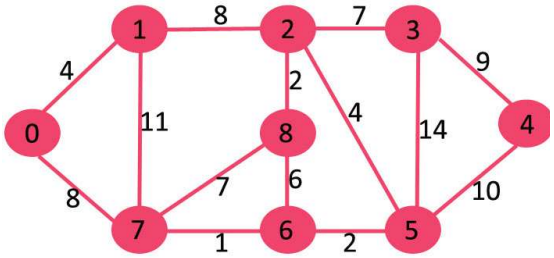
```
                                        edge.getVertex().setWeight(vertex.getWeight()
+ edge.getWeight());
                                }
                        }
                }
        }

        /**
         * Implementation of Dijkstra's algorithm
         *
         * @param graph
         */
        public static void dijkstraAlgorithm(Graph graph) {
                Map<Vertex, Boolean> selected = new HashMap<>() {{
                        for(Vertex vertex : graph.getVertices())
                                put(vertex, false);
                }};

                Vertex selectedVertex = graph.getVertices().get(0);

                for (int i = 1; i <= (graph.getVertices().size() - 1); i++) {
                        selected.put(selectedVertex, true);

                        for(Edge edge : selectedVertex.getEdges()) {
                                if((selectedVertex.getWeight() + edge.getWeight()) <
edge.getVertex().getWeight() && !selected.get(edge.getVertex()))

        edge.getVertex().setWeight(selectedVertex.getWeight() + edge.getWeight());
                        }

                        int score = Integer.MAX_VALUE;
                        for(Vertex vertex : graph.getVertices()) {
                                if(!selected.get(vertex) && vertex.getWeight() < score) {
                                        selectedVertex = vertex;
                                        score = vertex.getWeight();
                                }
                        }
                }
        }

}
```

Bellman Ford




```
Bellman Ford
------------
A: 0
B: 1
C: 3
D: 5
E: 0
F: 3
```

Dijkstra




```
Dijkstra
----------
0: 0
1: 4
2: 12
3: 19
4: 21
5: 11
6: 9
7: 8
8: 14
```

In the context of routing, these algorithms can be applied to determine the shortest path between a source and destination node. The values calculated by the algorithms represent the cost of passing through a particular path. By examining the resulting distance values, routers can make the best decision in selecting the shortest path to the destination node or network. For example, the router can use the values gotten from the algorithm to determine the fastest way of getting to **F** from **A** in the example above. Start from the destination node **F** and follow the predecessor nodes based on the distance values until you reach the source node **A**. This will give you the shortest path which is **A -> B -> E -> F.**

Other Applications of Bellman Ford's and Dijkstra's Algorithm

- These algorithms are used by navigation systems to find the quickest path to a location.
- Bellman ford is used to tell what set of actions to take to maximize profit.
- They are used in for static code analysis in compiler design.
- They are used in social networks to analyse connectivity and influence between individuals on the platform.