Name Member 1 : Febronia Ashraf Tawfik **ID : 29**

Name Member 2 : Neveen Samir Nagy **ID : 58**

Data Structures 2 - Bonus Lab
Shortest Paths Algorithms

1)Problem Statement :

We implement two shortest paths algorithms which are Dijkstra and Bellman-Ford.

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.

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.

---

2) Pseudo Code :

Dijkstra Algorithm

```
Arrays.fill(distances, Integer.MAX_VALUE / 2);
distances[src] = 0
PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
ArrayList<Integer> vertice = getVertices();
for (int i = 0; i < vertice.size(); i++)
pq.add(vertice.get(i))
endFor
pq.add(src)
while (!pq.isEmpty())
int vertex = pq.poll()
adjacent = getNeighbors(vertex)
for (int i = 0→ adjacent.size())
int distance = distances[vertex] + graph[vertex][adjacent.get(i)]
```

```
if (distance < distances[adjacent.get(i)])
pq.remove(adjacent.get(i));
distances[adjacent.get(i)] = distance;
pq.add(adjacent.get(i));
endIF
endFor
endWhile

end
```

Bellman-Ford Algorithm

```
Arrays.fill(distances, Integer.MAX_VALUE / 2);
distances[src] = 0
ArrayList<Pair<Integer, Integer>> edges
for (int i = 0→ vertices)
for (int j = 0→ vertices)
if (graph[i][j] != 0)
Pair<Integer, Integer> p = new Pair<Integer, Integer>(i, j)
edges.add(p)
endIF
endFor
endFor
for (int i = 1→ vertices)
for (int j = 0→ edges.size())
Pair<Integer, Integer> p = edges.get(j);
if (distances[(int) p.getKey()]
+ graph[(int) p.getKey()][(int) p.getValue()] < distances[(int) p.getValue()])
distances[(int) p.getValue()] = distances[(int) p.getKey()]
+ graph[(int) p.getKey()][(int) p.getValue()]
endIF
endFor
endFor
for (int j = 0→ edges.size())
Pair<Integer, Integer> p = edges.get(j);
if (distances[(int) p.getKey()]
+ graph[(int) p.getKey()][(int) p.getValue()] < distances[(int) p.getValue()])
return false
endIF
endFor
return true
end
```

3) Snipping :

```
29  public void readGraph(File file) {
30      // TODO Auto-generated method stub
31      if (file == null || !file.exists()) {
32          throw new RuntimeErrorException(null);
33      }
34      try {
35          BufferedReader reader = new BufferedReader(new FileReader(file));
36          String line = "";
37          String first = "^([0-9][0-9]*|1[0-9]+)+(\\s)+([0-9][0-9]*|1[0-9]+)+$";
38          String second = "^([0-9][0-9]*|1[0-9]+)+(\\s)+([0-9][0-9]*|1[0-9]+)+(\\s)+([-+])?+([0-9][0-9]*|1[0-9]+)
39          Pattern rFirst = Pattern.compile(first);
40          Pattern rSecond = Pattern.compile(second);
41          try {
42              while ((line = reader.readLine()) != null) {
43                  Matcher m1 = rFirst.matcher(line);
44                  Matcher m2 = rSecond.matcher(line);
45                  if (m1.find()) {
46                      vertices = Integer.valueOf(m1.group(1));
47                      edges = Integer.valueOf(m1.group(3));
48                      graph = new int[vertices][vertices];
49                      for (int i = 0; i < vertices; i++) {
50                          for (int j = 0; j < vertices; j++) {
51                              graph[i][j] = 0;
52                          }
53                      }
54                  } else if (m2.find()) {
55                      size++;
56                      int weight = Integer.valueOf(m2.group(6));
57                      if (m2.group(5) != null) {
58                          weight = -1 * weight;
59                      }
60                      graph[Integer.valueOf(m2.group(1))][Integer.valueOf(m2.group(3))] = weight;
61                  }
62              }
63              if (size < edges) {
64                  throw new RuntimeErrorException(null);
65              }
66
```

```java
@Override
public int size() {
    // TODO Auto-generated method stub
    return edges;
}

@Override
public ArrayList<Integer> getVertices() {
    // TODO Auto-generated method stub
    ArrayList<Integer> vertex = new ArrayList<Integer>();
    for (int i = 0; i < vertices; i++) {
        vertex.add(i);
    }
    return vertex;
}

@Override
public ArrayList<Integer> getNeighbors(int v) {
    // TODO Auto-generated method stub
    ArrayList<Integer> adjacent = new ArrayList<Integer>();
    if (v >= vertices || v < 0) {
        throw new RuntimeErrorException(null);
    }
    for (int i = 0; i < vertices; i++) {
        if (graph[v][i] != 0) {
            adjacent.add(i);
        }
    }
    return adjacent;
}
```

```java
public void runDijkstra(int src, int[] distances) {
    // TODO Auto-generated method stub
    predecessor = new int[vertices][2];
    Arrays.fill(distances, Integer.MAX_VALUE / 2);
    distances[src] = 0;
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    ArrayList<Integer> vertice = getVertices();
    for (int i = 0; i < vertice.size(); i++) {
        pq.add(vertice.get(i));
    }
    pq.add(src);
    while (!pq.isEmpty()) {
        int vertex = pq.poll();
        ArrayList<Integer> adjacent = new ArrayList<Integer>();
        adjacent = getNeighbors(vertex);
        for (int i = 0; i < adjacent.size(); i++) {
            int distance = distances[vertex] + graph[vertex][adjacent.get(i)];
            if (distance < distances[adjacent.get(i)]) {
                pq.remove(adjacent.get(i));
                distances[adjacent.get(i)] = distance;
                predecessor[adjacent.get(i)][0] = distance;
                predecessor[adjacent.get(i)][1] = adjacent.get(i);
                pq.add(adjacent.get(i));
            }
        }
    }
}
```

```java
    @Override
    public ArrayList<Integer> getDijkstraProcessedOrder() {
        // TODO Auto-generated method stub
        ArrayList<Integer> pre = new ArrayList<Integer>();
        for (int i = 0; i < vertices; i++) {
            for (int j = i + 1; j < vertices; j++) {
                if (predecessor[i][0] > predecessor[j][0]) {
                    int[] temp = predecessor[j];
                    predecessor[j] = predecessor[i];
                    predecessor[i] = temp;
                }
            }
            pre.add(predecessor[i][1]);
        }
        return pre;
    }
```

```java
    @Override
    public boolean runBellmanFord(int src, int[] distances) {
        Arrays.fill(distances, Integer.MAX_VALUE / 2);
        distances[src] = 0;
        ArrayList<Pair<Integer, Integer>> edges = new ArrayList<Pair<Integer, Integer>>();
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                if (graph[i][j] != 0) {
                    Pair<Integer, Integer> p = new Pair<Integer, Integer>(i, j);
                    edges.add(p);
                }
            }
        }
        for (int i = 1; i < vertices; i++) {
            for (int j = 0; j < edges.size(); j++) {
                Pair<Integer, Integer> p = edges.get(j);
                if (distances[(int) p.getKey()]
                        + graph[(int) p.getKey()][(int) p.getValue()] < distances[(int) p.getValue()]) {
                    distances[(int) p.getValue()] = distances[(int) p.getKey()]
                            + graph[(int) p.getKey()][(int) p.getValue()];
                }
            }
        }
        for (int j = 0; j < edges.size(); j++) {
            Pair<Integer, Integer> p = edges.get(j);
            if (distances[(int) p.getKey()]
                    + graph[(int) p.getKey()][(int) p.getValue()] < distances[(int) p.getValue()]) {
                return false;
            }
        }
        return true;
    }

}
```