**Implementation of Advanced Data Structures and Algorithms**

**Project – 8(Basic)**

**Suraj Sangani (sns140230)**

**Abstract:**

Given a directed graph as input, if the graph has uniform weights (i.e., same positive weights for all edges), then it runs BFS to find shortest paths. Otherwise, if the graph is a directed, acyclic graph (DAG), then it runs DAG shortest paths. Otherwise, if the graph has only nonnegative weights, then it runs Dijkstra's algorithm. If all these test fail, then it runs the Bellman-Ford algorithm. If the graph has negative cycles, then it prints the message "Booyah!!"

**Input Specification:**

The input is taken from stdin (Standard input), and contains a sequence of lines. The first line of input has 4 integers, |V|, |E|, s and t. The next |E| lines of input have 3 integers in each line: u, v, w, indicating a directed edge (u,v) of weight w. Nodes are numbered 1..|V|. Limits: |V| is between 3 and 1,000,000. |E| is at most 10,000,000.

**Development Platform:**

Eclipse Java EE IDE Luna 4.4.0;

Windows 7, 8.00GB RAM, 64-bit OS

**Methodology:**

1. Input is read from the Standard input/output.
2. User Input is read using Scanner class.
3. Input Format :
   1. **<|**V|><|E|><|S|><|D|>
   2. **<**u><v><w>
   3. Where V represents the number of vertices in the graph, E represents the number of lines of input to be expected after the first line, S is the source vertex of the graph and D is the destination vertex of the path. The notations u, v, w represent the source, destination and weight of the edge respectively.
4. The graph is stored as an adjacency lists which uses an array of linked lists.
5. The program first calls the LoadGraph() function which loads the graph into memory. An instance of the GraphNode class is created - the DFSadjlist which stores the vertices along with the outgoing edges to other vertices.
6. The addEdge function is called from within the LoadGraph() function which simply adds the edge i.e. the source, destination and weight of the edge to the adjList and the DFSadjlist.
7. The weights of the edges are stored in an array.
8. If the weights are uniform, then Breadth First Search is run in order to find out shortest path from the source to the destination.
9. If the weights are non-uniform and non-negative then Dijkstra’s Algorithm is run to find out the shortest path from the source to the destination.
10. If the weights are negative but no cycles exist then the algorithm to find a shortest path from source to destination is run using Directed Acyclic Graphs algoritm. The cycle detection function returns a Boolean value which indicates whether the graph contains cycles or not.
11. If all these conditions fail, then the Bellman-Ford algorithm is run.
12. If there are negative cycles in the graph, “Booyah” is printed.
13. The user has to type “Output” in order to print the answer.

**Test Cases:**

Input:

8 12 1 5

1 2 2

1 4 1

2 5 10

2 4 3

5 7 6

3 1 4

3 6 5

4 3 2

7 6 1

4 5 2

4 7 4

4 6 8

Output

Dij 3 0msec

1 0 -

2 1 1

3 4 4

4 2 1

5 3 2

6 7 4

7 6 4

8 INF -

**Discussion** **of** **Results:**

The program correctly finds out the shortest path from source to all nodes using all 4 algorithms in a reasonable amount of time –

1. Breadth First Search

2. Dijkstra’s Algorithm

3. DAG

4. Bellman Ford

**Conclusion:**

The shortest path from source to destination is correctly found out using this project.

**References:**

* Breadth First Search – Introduction to Algorithms Third Edition – Cormen, Leiserson, Rivest, Stein
* [**https://www.youtube.com/watch?v=iTW2yFYd1Nc**](https://www.youtube.com/watch?v=iTW2yFYd1Nc)
* [**https://www.youtube.com/watch?v=gdmfOwyQlcI**](https://www.youtube.com/watch?v=gdmfOwyQlcI)
* http://www.utdallas.edu/~sizheng/CS4349.d/l-notes.d/L17.pdf