```
CS 6301.002.  Implementation of advanced data structures and algorithms
Spring 2016;  Tue, Mar 8.
Long Project 3: Shortest paths

Ver 1.0: Initial description (Tue, Mar 8: 9:00 AM).
Max excellence credits: 1.0 (only for level 2).
Due: 11:59 PM, Sun, Mar 27 (1st deadline), Sun, Apr 10 (2nd deadline).
```

## Code base

Java library, Java examples discussed in class by instructor. You can reuse indexed heaps from sp0pq or lp2. Do not use code from outside sources.

## Project Description

This project has 2 levels. You are required to complete Level 1, and Level 2 is optional. In the first level, you need to implement the shortest path algorithms discussed in class. No excellence credits will be awarded for submitting level 1. In level 2, you need to output the number of shortest paths from the source s to each node of the graph. If that problem is not solvable because of a negative (or zero) cycle in the graph, then you need to output such a cycle.

## Level 1 (No EC)

Implementation of shortest path algorithms: BFS, Dijkstra's algorithm, DAG shortest paths, and Bellman-Ford algorithm. Write a program that behaves as follows: 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 shorest 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 "Unable to solve problem. Graph has a negative cycle".

## Level 2 (EC: 1.0)

**Number of simple shorest paths from source s:** This part is optional, and you are eligible for EC of up to 1.0. Consider Exercise 24.2-4 in Cormen et al (3rd ed). The number of paths in a DAG from source s to each vertex u in G can be easily calculated in linear time. If G does not contain a negative or zero cycle reachable from s, then it can be shown that the set of all shortest paths from s forms a DAG (by selecting only those edges e=(u,v) with v.distance = u.distance + e.Weight). By using the algorithm for counting paths in DAGs on this subgraph of G, we can find the number of shortest paths from s to each node of G. The input is a directed graph G=(V,E), with edge weights W:E-->Z (negative weights are possible). The output is the sum of the number of shortest paths (not necessarily disjoint) from s to each vertex u in the graph. If the graph has a negative or zero cycle, reachable from s, then print a message "Non-positive cycle in graph. DAC is not applicable". In addition, your program should output such a cycle.

## Input specification

A directed graph is given as input, in the format expected by readGraph method. If there is a command line argument, input is read from that file. Otherwise, read input from the console. Assume that the source is vertex 1. Do not process lines in the input after the input graph has been read.

## Output specification (Level 1)

In the first line of the output, print the name of the algorithm run (BFS, DAG, Dij, or B-F), the sum of shortest path lengths from s to every node of G that is reachable from s. If |V| is less than or equal to 100, in the next lines, output the lengths of shortest paths from s to each vertex u in the graph that is reachable from s, and the predecessor node of u in that shortest path. If there is no path from s to a vertex, print INF as the length. In the following example, node 8 is not reachable from s, and the output in line 1 is the sum of u.distance for all u for which u.distance != Infinity.

**Output specification (Level 2)**

In the first line of the output, print the sum of number of shortest paths from s to every node of G. If |V| is less than or equal to 100, in the next lines, for each vertex u, output the length of a shortest path from s to u, and the number of shortest paths from s to u. If there is no path from s to a vertex, print INF as the length, and the number of paths as 0.

```
Sample input (level 1):
8 12
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 (level 1):
Dij 20
1 0 -
2 2 1
3 3 4
4 1 1
5 3 4
6 6 7
7 5 4
8 INF -


Sample input (level 2):
7 8
1 2 2
1 3 3
2 4 5
3 4 4
4 5 1
5 1 -7
6 7 -1
7 6 -1
```

Note that in this example, there is a negative cycle, 6->7->6 of length -2. But, since there is no path from s to either of these nodes, and therefore, the algorithm can correctly output the answers. Also, there are 2 shortest paths from 1 to 5: 1->2->4->5, and 1->3->4->5. Both paths have length 8, but they are not disjoint.

```
Output (level 2):
7
1 0 1
2 2 1
3 3 1
4 7 2
5 8 2
6 INF 0
7 INF 0
```