CP 372, Winter 2019

INDIVIDUAL ASSIGNMENT!!!

NO GROUP WORK OR GROUP SUBMISSION!!!

Assignment 3 (10%) **Due Wednesday, April 3, 9:59 p.m.**

In this assignment you will implement a core functionality of the **Link State** routing algorithm for an autonomous system. The input for the algorithm is a weighted directed graph with **n** vertices (labeled with 1, 2, 3,…, n) that represents topology of a particular autonomous system. This is followed by one line list of "gateway routers" x_1, …, x_k, where x_i are distinct numbers from the set {1,2,….,n}. The output consists of **n-k** forwarding tables to gateway routers (one per router) which allows datagram forwarding from a source to a destination gateway router along the shortest path.

Your implementation must read input data from the system input and print the tables to the system output!!!!!!!

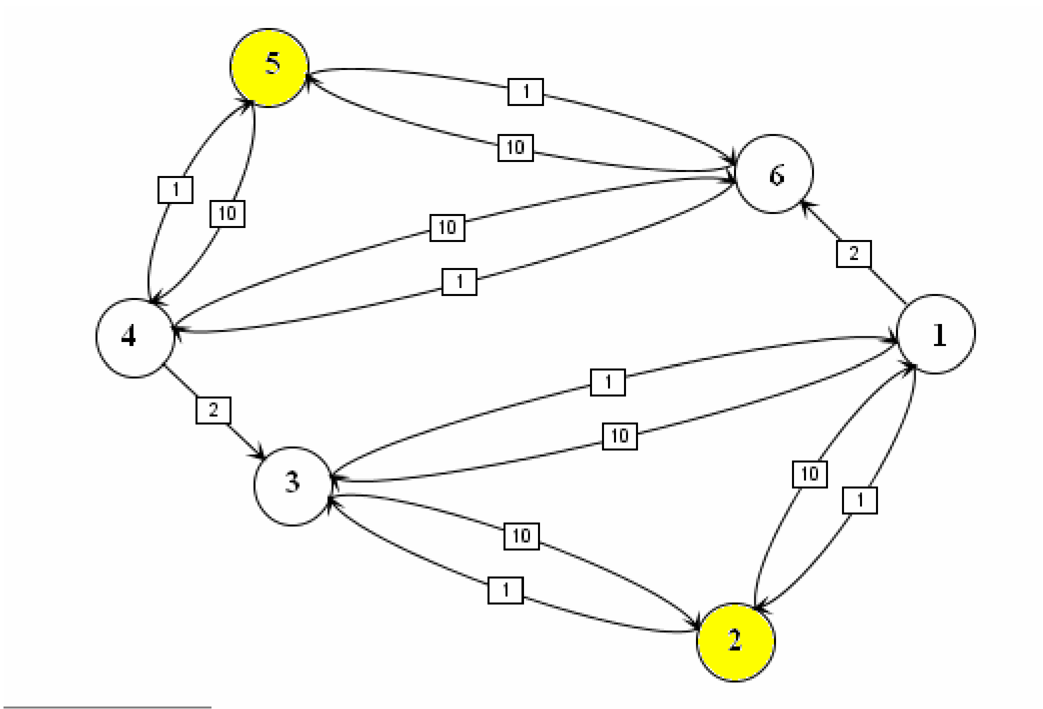
The input starts with single line containing the value of **n**, followed by n lines representing adjacency matrix with weights, and one more line representing a list of gateway routers as in the following example:

```
6

0 1 10 -1 -1 2

10 0 1 -1 -1 -1

1 10 0 -1 -1 -1

-1 -1 2 0 1 10

-1 -1 -1 10 0 1

-1 -1 -1 1 10 0

2 5
```

Note:

- -1 represents an infinite weight (absence of directed edge)
- the matrix is not necessary symmetric, as in the above example, where link from router 1 to 3 has weight 10 and backward link from router 3 to 1 has weight 1
- You can assume that gateway routers are listed in ascending order and that the list is not empty and **k<n**.

This input above represents the following graph with vertices 2 and 5 being gateway routers.



The output for this input should consist of 4 forwarding tables – each containing 2 entries that correspond to shortest path to area border routers. For example:

Forwarding Table for 1

To	Cost	Next Hop
2	1	2
5	4	6

…

…

Forwarding Table for 6

To	Cost	Next Hop
2	5	4
5	2	4

Note additionally:

- do not assume that every vertex is reachable from any other vertex (imagine graph with 2 connected components for example); if a particular destination is not reachable from the source print -1 as the “Cost” and -1 as the “Next Hop” destination
- you can assume correct input (no exceptions handling)
- Dijkstra algorithm allows one to compute not only single source shortest path weights, but also predecessor for every node along the shortest path from the source, which is useful when constructing forwarding table at the source

Implement Dijkstra algorithm that takes a graph and 1 source vertex, determines shortest path to all other vertices from the source and forms the forwarding table at the source. Call this algorithm **n-k** times to produce desired results.

What to submit:

- Single zip file named <yourLaurierUserName>.zip containing single Java file LinkState.java with your implementation.

Java file submitted have to use default package (**no package keyword**!!!).
It should be possible to compile your submission in a command line environment with

```
> javac LinkState.java
```

Test cases will be run with redirection of the input, such as for example

```
java LinkState < in1.txt
```

where file in1.txt will contain a particular input.

Bonus can be given for implementation of “alternative forwarding”: if there is an alternative path of the same weight from the source to particular destination, it might be useful to find it too. The entry in the forwarding table in this case might have two “next hop” elements (which is helpful for the alternative forwarding).