Lab 7

1 Task 7.1

Write a program which for a given adjacency matrix A and a topological order O of vertices of a weighted Directed Acyclic Graph (see lecture "Paths in DAG (Critical tasks)") finds the longest paths from the source vertex to all other vertices (that is, from the first vertex in the topological order O to all other vertices). For simplicity of the implementation assume that graph edges have **only positive weights** and an entry 0 in the adjacency matrix represents **that there is no edge** (lack of an edge) [1].

You may use the code skeleton available on the course webpage (`https://undefiened.github.io/tnsl20/lab7.m`) in which we have added code to initialize vectors l and p and added code to plot the graph and highlight the longest path from the first to the last vertex (you may tweak the code to visualize the longest path to other vertices). Write the necessary code in the place of comment "% WRITE YOUR CODE HERE". Please note that it is important to use name p for the vector of predecessors to make plotting work (otherwise change the plotting code in the bottom of the script).

In order to implement the code for finding the longest path you may use

[1] For a more general version allowing working with weighted DAGs having edges with 0 and negative weights you may set entries of adjacency matrix representing lack of an edge to ∞ and modify the algorithm so that it only works with entries $A[u, v] < \infty$. Alternatively, you may use "NaN" (in Matlab "NaN" means "Not a Number") instead of 0.

the following pseudocode:

Algorithm 1: Longest paths in DAG

```
Input  : Adjacency matrix $A$, topological order $O$
Output: A vector $p$ of predecessors in the longest paths tree from
        the source vertex
$l$ := vector of zeros with the same size as $O$ (vector of vertices
  labels);
$p$ := vector of zeros with the same size as $O$ (vector of predecessors);
for $i$ = 2 TO length(O) do
    $v := O[i]$;
    for $j$ = 1 TO $i$ do
        $u = O[j]$;
        if $A[u, v] > 0$ then
            $w := A[u, v]$;
            if $l[v] < l[u] + w$ then
                $l[v] := l[u] + w$;
                $p[v] := u$;
            end
        end
    end
end
```

The result of this algorithm will be a vector p which will store predecessors in the longest paths tree for all vertices from which you can easily figure out the longest path. For example, if you have a vector $p = [0, 1, 1, 2, 2, 2]$ as a result of running the algorithm, we know that the previous vertex in the longest path for the last vertex is vertex 2. And for vertex 2 the previous vertex in the longest path is vertex 1. That is, the longest path from vertex 1 to the last vertex is $1 \to 2 \to 6$.

Try to change weights of the edges in the adjacency matrix from 1 to something else to see how the longest path changes (make sure that you do not introduce new cycles and that O still represents a correct topological order).