CS3000: Algorithms — Iraklis Tsekourakis

Homework 6

Name: Ryan Tietjen
Collaborators:

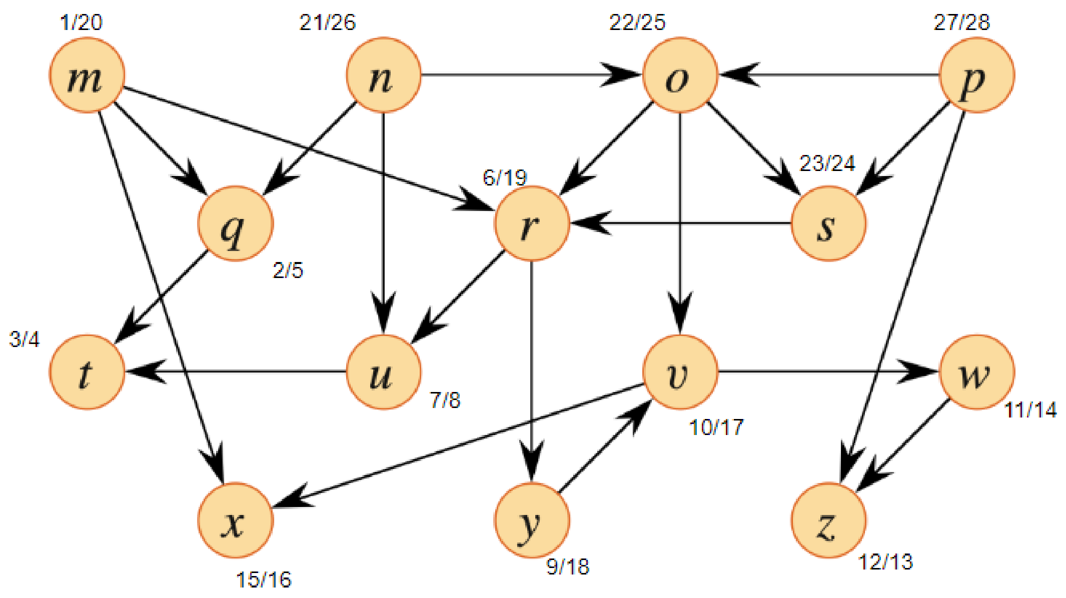
Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the instructions of the assignment in the pdf description of the assignment.

1. (10 points) Show the output of the DFS (discovery and finishing times), and the ordering of vertices produced by TOPOLOGICAL-SORT when it is run on the DAG shown. Assume that the for loop of the DFS procedure considers the vertices in alphabetical order and assume that each adjacency list is ordered alphabetically.

Solution:

The discovery time/finishing time of each vertex is as follows:



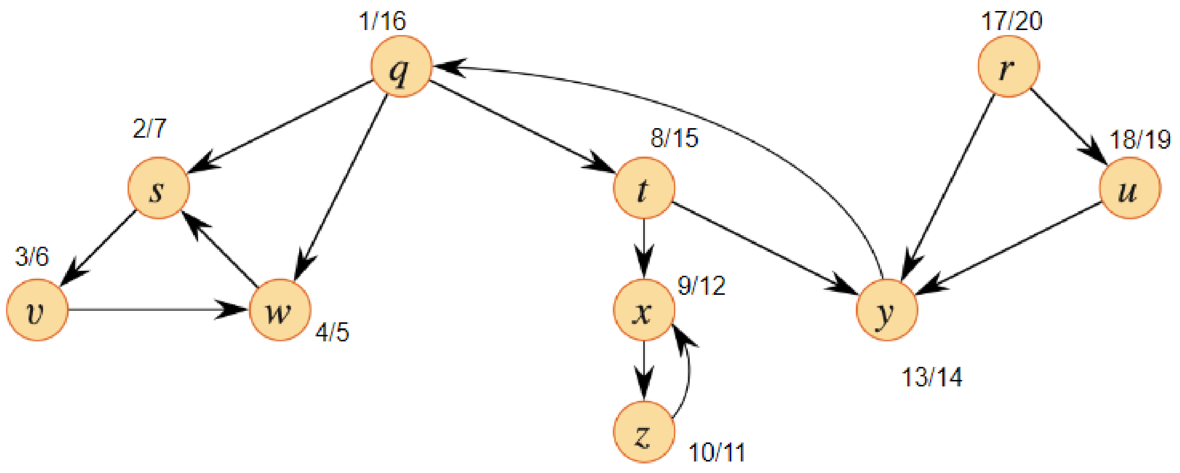
Topological ordering (<finish time> <vertex>):

28 p
26 n
25 o
24 s
20 m
19 r
18 y
17 v
16 x
14 w
13 z
8 u
5 q
4 t

2. (12 points)

Solution:

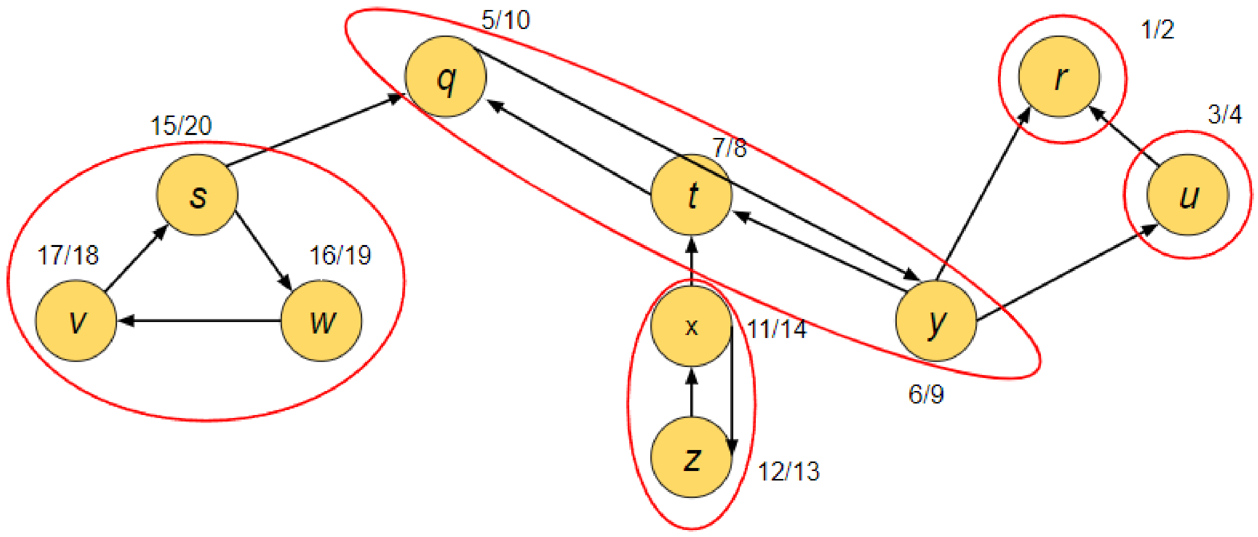
Step 1: Perform DFS on the graph:



Hence, we will consider nodes in their topological order(<finish time> <vertex>):

20 r
19 u
16 q
15 t
14 y
12 x
11 z
7 s
6 v
5 w

Steps 2/3: Find the transpose of the graph and perform DFS on it, considering the vertices by the decreasing finishing times found in the previous step (i.e. begin with r, u, q, ...). The non-discovered vertices that we reach from a vertex will be a part of the same SCC. The resulting SCCs are:



3. (12 points)

Solution:

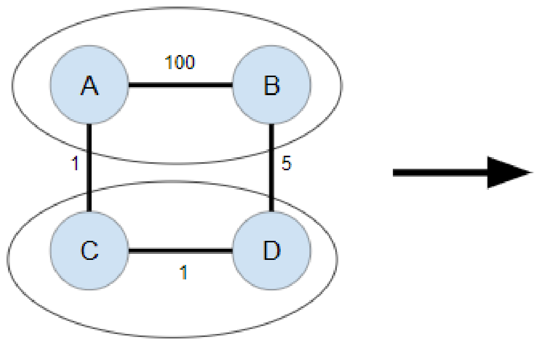


Figure 3.1: Graph G

Figure 3.2: Resulting MST

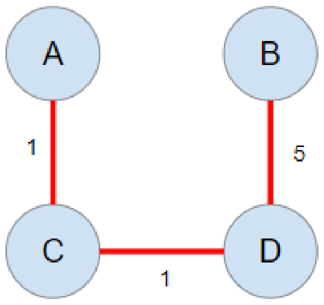


Figure 3.3: Correct MST

This algorithm fails to compute a correct MST:
Suppose we have an undirected Graph G (Figure 3.1), with vertices A, B, C, D and Edges:
-AB with weight 100.
-AC with weight 1.
-BD with weight 5.
-CD with weight 1.
Now, let V_1 contain A, B, and let V_2 contain C, D.
Then, $|V_1|$and $|V_2|$differ by 0 as required.
Also, V_1 contains only the edge AB, and V_2 contains only the edge CD (Figure 3.1).
Hence, the MST of V_1 contains only the edge AB, and the MST of V_2 contains only the edge CD.
Next, the minimum-weight edge that crosses the cut (V_1, V_2) would be edge AC with weight 1.
So, the MST produced by the algorithm (Figure 3.2) contains the edges AB, AD, and AC.
However, this is not the correct MST (cumulative weight of 102). The correct MST would contain the edges AC, CD, and BD with a cumulative weight of 7 (Figure 3.3).

4. (12 points)

Solution:

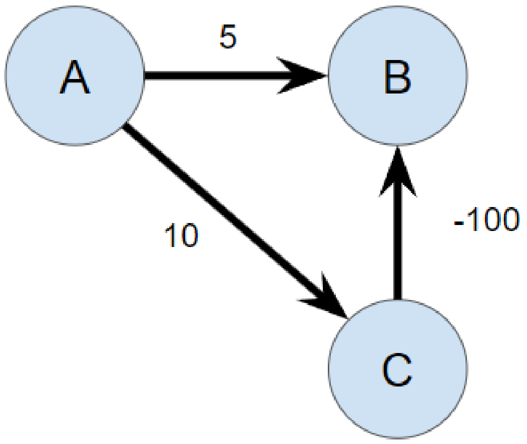
To terminate in $m+1$ passes, we must include an early stopping condition by checking if no relaxations have been made during a pass. If no relaxations occur during a pass, then it means that all shortest paths have been computed. Also, no relaxations will occur during the $(m+1)$th pass.

Algorithm 1: Modified Bellman-Ford

```
Function $MODIFIED-BELLMAN-FORD(G, w, s)$:
    $INIT-SINGLE-SOURCE(G, s)$
    For $1 \leq i \leq |G.V| - 1$
        $noChangesMade$ = True
        For each $edge(u, v) \in G.E$
            If $MODIFIED-RELAX(u, v, w)$ :
                $noChangesMade$ = False
        If $noChangesMade$ :
            break
    For each $edge(u, v) \in G.E$
        If $v.d > u.d + w(u, v)$ :
            Return False
    Return True
Function $MODIFIED-RELAX(u, v, w)$:
    If $v.d > u.d + w(u.v)$ :
        $v.d = u.d + w(u, v)$
        $v.\pi = u$
        Return True
    Return False
```

5. (12 points)

Solution:



Assume that we are trying to find the shortest path from vertex A to vertex B. In the above graph, Dijkstra's algorithm will produce the path A $\rightarrow$ B with a cumulative weight of 5. However, the correct solution would be A $\rightarrow$ B $\rightarrow$ C with a cumulative weight of -90.

The correctness proof fails when we have negative weight edges because Dijkstra's algorithm assumes that we have already found the shortest path to the vertex we are currently considering. Note that we obey the greedy property of following the "lightest" path. Hence, with only non-negative edges, adding an edge to a path can only increase the weight of the path, a property that Dijkstra's algorithm depends on.

However, when we have negative weight edges, the shortest path to a vertex might not necessarily be the path that initially appears to be the shortest. Once a vertex has been visited, Dijkstra's algorithm will no longer consider any other paths to it, assuming that the shortest path has already been found. But, with negative weight edges, there might exist a shorter path that the algorithm ignores because it initially appears as a longer path (as shown in the example above). In essence, adding an edge to a path might decrease the weight of the path, which Dijkstra's algorithm fails to account for. Hence, the property of the current vertex already having the shortest path to it no longer holds. Since the correctness proof relies on this property, it also does not hold.