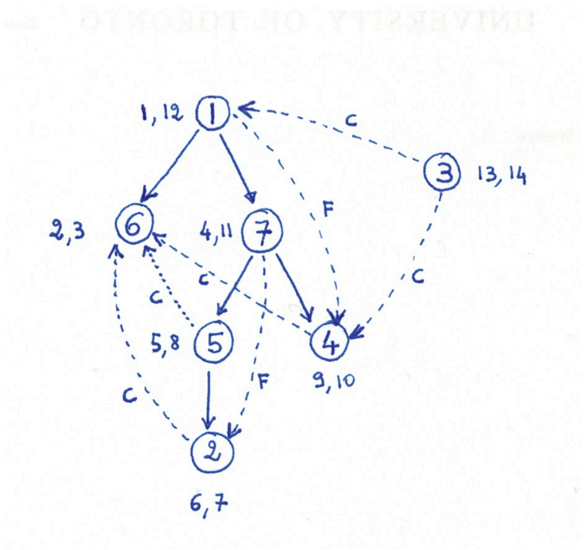
Computer Science CSC263H
St. George Campus

March 29, 2018
University of Toronto

Solutions for Homework Assignment #6

## Answer to Question 1.

**a.**



**b.** The above DFS has 0 back edges, 2 forward edges, and 5 cross edges.

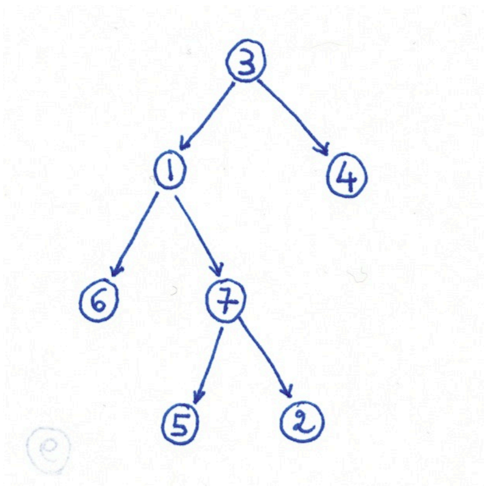
**c.** By part (b), some DFS of $G$ has no back-edges. In class we proved that:

**Theorem**: For every directed graph $G$ and every DFS of $G$, $G$ has a cycle iff the DFS of $G$ has a back edge.

Thus, $G$ has no cycles. Therefore there is a topological sort of $G$, i.e., the courses can be taken in an order that satisfies all the prerequisites.

**d.** The topological sort algorithm outputs all the nodes of $G$ in order of decreasing $f[]$ "finish" times. This gives the following list: 3, 1, 7, 4, 5, 2, 6.

**e.** Draw a Breadth-First Search tree of $G$ that ***starts at node* 3** and explores the edges in the order of appearance in the above adjacency lists.



**Answer to Question 2.**

Let $D \subseteq E$ be the set of damaged roads. Consider $G$ as a weighted graph, where the weight (cost) $c_e$ of edge $e$ is equal to 0 if $e$ corresponds to an undamaged road, and is equal to the cost of repairing $e$, if $e$ corresponds to a damaged road. We will model the problem as a minimum spanning tree problem. Our goal is to connect all cities using either undamaged roads, which cost 0, or using repaired damaged roads. Then, the minimum possible cost to connect the cities equals the cost of the minimum spanning tree in the weighted graph $G$. If $T = (V, E_T)$ is a minimum spanning tree of $G$, then the set of nodes to repair will be $S = E_T \cap D$.

In order to compute $T$ and $S$ in the desired running time, we modify Kruskal's algorithm. The main idea is *not* to sort the undamaged roads. The modified algorithm is below. We use $m$ for $|E|$. We will use the Disjoint Set Forest Union-Find data structure in the algorithm.

```
for $i = 1$ to $n$
    MAKE-SET($i$)
$S = \emptyset$
Initialize array $D[1..k]$
$j = 0$
for $i = 1$ to $m$
    if $R[i].cost == 0$
        $(u,v) = R[i].edge$
        $x =$ FIND($u$)
        $y =$ FIND($v$)
        if $x \neq y$
            UNION($x, y$)
    else $j = j + 1$
        $D[j] = R[i]$
Sort $D$ in non-decreasing order by cost.
for $i = 1$ to $k$
    $(u,v) = D[i].edge$
    $x =$ FIND($u$)
    $y =$ FIND($v$)
    if $x \neq y$
        UNION($x, y$)
        Add $(u, v)$ to $S$.
```

If $E_T$ is the set of edges $(u, v)$ for which UNION is called (in both For-loops), then $T = (V, E_T)$ is a minimum spanning tree, because this is the set of edges that is computed by Kruskal's algorithm as well: first all the edges of cost 0 are inspected, and then all edges of positive cost, in sorted order. The set $S$ output by the algorithm is exactly the subset of edges in $E_T$ which are damaged, as described above. This proves the correctness of the algorithm. Initializing the Union-Find data structure takes time $O(n)$, initializing the array $D$ takes time $O(k)$. The running time of both For-loops is dominated by the cost of Union and Find operations. Since we have $n$ elements, and perform at most $m$ Find operations, and the graph $G$ is connected, so $m \geq n - 1$, the total running time of Union and Find operations is $O(m \log^* m)$. Finally, we have the running time to sort the size $k$ array $D$. Using either QuickSort or MergeSort, this will take $O(k \log k)$ operations. Adding up these running time gives the bound $O(m \log^* m + k \log k)$.