CSC263 Assignment 6

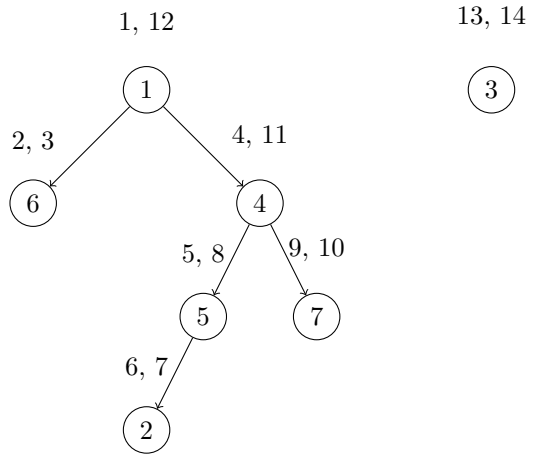
Yongzhen Huang, Xinyi Liu, Kewei Qiu

March 28th, 2019

Question 1

Wrote by: Xinyi Liu, Kewei Qiu
Check by: Yongzhen Huang

a)



b)

back edges: 0
forward edges: 2
cross-edges: 5

c)

To prove that it is possible to take all the courses in a sequential order that satisfies all the prerequisite requirements, we need Theorem 22.11 CLRS which we covered in lecture.

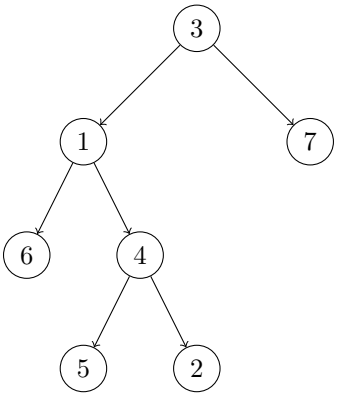
The theorem says that: For all directed graphs G and all DFS of G, DFS of G has a back edge $\Leftrightarrow$ G has a cycle.

Proof: From part(b) we know that G has no back edges, hence by Theorem 22.11 G has no cycle. As we discussed in tutorial, if a graph G has no cycle then we can use topological sort algorithm to produce a topological order over nodes of G, which means that it is possible to take all the courses in a sequential order that satisfies all the prerequisite requirements.

d)

From what we learnt in tutorial, since this graph is directed and acyclic, we can perform Topological Sort on the graph.
Then we generate this order: 3 1 4 7 5 2 6

e)



Question 2

Wrote by: Yongzhen Huang
Check by: Xinyi Liu, Kewei Qiu

We keep track of the minimum and maximum numbers we have assigned, so initially unknown.

First, we create an adjacency list representing a graph with size n and fill them in with each element from constrains as nodes. Then iterate through the constraints to add directed edges to this graph. For each constraint, the rule is that using the LHS of each constraint as the starting node of the edge and using the RHS of each constraint as the ending node of the edge. So what we need to do is going to the position which represents the RHS of constraint in adjacency list, and add RHS of the constraint into the linked list stored in this position. At the same time we also stores some information in the node of RHS element: a set of RHSs of constraints and types of constraints (equality or inequality).

Now we apply DepthFirstSearch on the graph generated by this adjacency list. Specifically, suppose we start with x_1, we assign a number to x_1 and update minimum and maximum to this number, then within the x_1 part of the adjacency list, for every equality we first check if the number has been discovered yet (i.e. assigned a number). If there is a number (i.e. discovered), we checkout if the equality is violated or not. If not violated, then we continue, otherwise return NIL. If not discovered, then we assign it to the same number as x_1 and therefore marked the variable as discovered. If there is an inequality, we check to see if the number is discovered yet. If discovered, then checkout if the number assigned to that variable satisfies the constraint; terminate if violates by returning NIL. If not discovered, then we examine the minimum and maximum number to assign a distinct number and update the minimum (if $x_1 > x_k$) or the maximum (if $x_1 < x_k$). We continue applying DFS like this until all nodes are discovered or if the algorithm returns NIL due to a violation. At the end, if the algorithm was not terminated early (no NIL returned), all constraints should have been satisfied.

Creating the adjacency list is $\mathcal{O}(n)$ since there are n elements. Adding to the adjacency list is $\mathcal{O}(m)$ since there are m constraints. Since at each step of Depth First Search the operation to check numbers is constant, the total cost of DFS is $|\mathcal{V}| + |\mathcal{E}|$ and thus $\mathcal{O}(n+m)$. Therefore, the total cost of this algorithm is $\mathcal{O}(m+n)$.

Question 3

Wrote by: Yongzhen Huang
Check by: Xinyi Liu, Kewei Qiu

Proof. Will prove by contradiction. For the sake of contradiction, suppose there is a minimum spanning tree of G that contains e_{max}. Since for every edge $e \in E$ there is a cycle of G that contains e, then there is a cycle $v_0, v_1, ..., v_k, v_0$ with edges $e_0, e_1, ..., e_k$, and without loss of generality, suppose v_0 to v_1 is $e_0 = e_{max}$. Note that the rest of the graph connects to this cycle in some fashion. So there is an edge e_i between some v_i, v_k that is not within the minimum spanning tree so that there is no cycle. Furthermore, the rest $e_j, j \neq i$ are all within the tree to ensure connectedness. Now, suppose we connect with edges $e_1, e_2, ..., e_k$ and leave out $e_0 = e_{max}$, these $v_0, ..., v_k$ are all still connected and note that the difference between this set of nodes and previous set of nodes including e_{max} is only $|e_{max} - e_i|$. Since each edge weight is distinct and e_{max} is the maximum, we get that $e_{max} - e_i \geq 1$. So having e_i instead of e_{max} is a minimum spanning tree within this cycle. But since the rest of nodes (i.e. $V \ \{v_0, ..., v_k\}$) are connected the same way, having e_{max} cannot be a minimum spanning tree! This contradicts with the initial assumption. Therefore, there is no minimum spanning tree that can include e_{max}.

Remark: Examing only the cycle containing e_{max}, the rest of the nodes are connected in some fashion. Note that such e_{max} is distinct, so to connect these $v_0, ..., v_k$ $k+1$ nodes, we need k edges; in fact, ANY k edges chosen from these $k+1$ edges will ensure connectedness. So we can only exclude one edge, the only way to reach the minimum is to remove e_{max}. ■