# True, False, Explain

Decide whether each of the following statements is true or false, and give a reason.

**Problem 1.** In a connected, weighted graph, every lowest weight edge is always in *some* minimum spanning tree.

**[True / False]**

**Solution.** True. It can be the first edge added by Kruskal's algorithm.

**Problem 2.** For a connected, weighted graph with n vertices and exactly $n$ edges, it is possible to find a minimum spanning tree in $O(n)$ time.

**[True / False]**

**Solution.** True. This graph only contains one cycle, which can be found by a DFS. Just remove the heaviest edge in that cycle.

**Problem 3.** Negating all the edge weights in a weighted undirected graph $G$ and then finding the minimum spanning tree gives us the maximum-weight spanning tree of the original graph $G$.

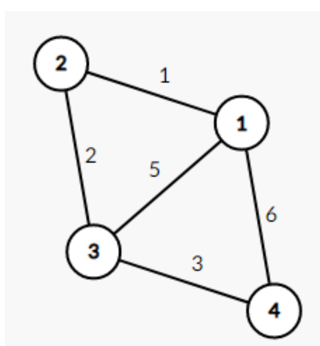
**[True / False]**

**Solution.** True.

**Problem 4.** In a graph with unique edge weights, the spanning tree of second lowest weight is unique.

**[True / False]**

**Solution.** False. For example, the spanning tree of second lowest weight has cost 9 for the graph below. You can verify that there is more than one way to connect the nodes up with cost 9.




**Problem 5.** If a graph has a unique shortest path $P$ from node $s$ to node $t$, and has a unique minimum spanning tree $T$, then every edge in $P$ must also be in $T$.

**[True / False]**

**Solution.** False. Counter-example is a triangular graph with edges of weight 3,4,5.

**Problem 6.** In a simple, undirected, connected, weighted graph with at least three vertices and unique edge weights, the heaviest edge in the graph is in no minimum spanning tree.

**[True / False]**

**Solution.** False. If the heaviest edge in the graph is the only edge connecting some vertex to the rest of the graph, then it must be in every minimum spanning tree.

**Problem 7.** Suppose that $T$ is a minimum spanning tree of $G$. If we increase the weight of each edge of $G$ by the same positive amount $\delta$, $T$ is not guaranteed to be a minimum spanning tree.

**[True / False]**

**Solution.** Yes. Since each tree in $G$ has $(|V|-1)$ edges, after the increase of weight, the total weight of each tree will be increased by the same amount $(|V|-1)\delta$. $T$ is therefore still the minimum spanning tree in the new graph.

**Problem 8.** An AVL tree is balanced, therefore a median of all elements in the tree is always at the root or one of its two children.

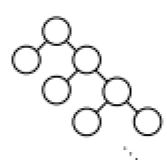
**[True / False]**

**Solution.** False. An AVL tree doesn't guarantee that the left and right subtrees will be equal sizes; it only guarantees that the heights of the trees are close. You can try to construct a counterexample.

**Problem 9.** If every node in a binary search tree has either 0 or 2 children, then performing a find operation on the tree takes $O(\log n)$ time.

**[True / False]**

**Solution.** False. One counterexample is the tree below. Since it has $O(n)$ height, a find operation can take $O(n)$ time.



**Problem 10.** Let $P$ be a shortest path from some vertex $s$ to some other vertex $t$ in a directed graph. If the weight of each edge in the graph is increased by one, $P$ will still be a shortest path from $s$ to $t$.

**[True / False]**

**Solution.** False. For example, you can construct a ring of vertices connected by edges of weight 0.

**Problem 11.** If a weighted directed graph $G$ is known to have no shortest paths longer than $k$ edges, then it suffices to run Bellman-Ford for only $k$ passes in order to solve the single-source shortest paths problem on $G$.

**[True / False]**

**Solution.** True. We need to run bellman ford for as many iterations as the maximum number of edges in any shortest path in the graph.

**Problem 12.** BFS takes $O(V+E)$ time irrespective of whether the graph is presented with an adjacency list or with an adjacency matrix.

**[True / False]**

**Solution.** False. With an adjacency matrix representation, visiting each vertex takes $O(V)$ time, as we must check all $n$ possible outgoing edges in the adjacency matrix. Thus, BFS will take $O(V^2)$ time using an adjacency matrix.

*Credits: Problems taken from MIT 6.006 Introduction to Algorithms / 6.046 Design and Analysis of Algorithms*