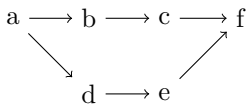
Assignment 3 solutions

Responsible TA: Claudi Lleyda Moltó

Task 1 - Topological ordering

How many topological orderings does the following graph have?



Solution. Note how a must come first, and f must be last. Indeed, these are the only ones to have no parents and no children, respectively. All our orderings will therefore be of the form $a****f$.

In all valid orderings, we must have b before c and d before e, but there are no more restrictions.

On one hand, if b comes first, that is $ab***f$, the next node cannot be e, since it would have appeared before d. Thus, it must be $abc**f$ or $abd**f$.

From $abc**f$ the only possible solution is $abcdef$, but from $abd**f$ both $abdcef$ and $abdecf$ are valid.

On the other hand, if d comes first we have a symmetrical scenario, where we can use the same reasoning to deduce that the only possible orderings are $adebcf$, $adbecf$ and $adbcef$.

This totals 6 different orderings, which are all the possible topological orderings in the graph.

The problem can be solved algorithmically, for example with the exercise code. $\Diamond$

Task 2 - Binary trees

A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Solution. We solve this by induction on the number of nodes in the tree. Let T be a binary tree and let $l(T)$ be the number of leaves of T, and $n(T)$ the number of nodes with two children in T. We want to prove that $n(T) = l(T) - 1$.

Indeed, if we have a tree with a single node, then we have no nodes with two children and exactly one leave, therefore $n(T) = 0 = 1 - 1 = l(T) - 1$, and we have shown the base case.

Assume this statement is true for a tree with $k > 1$ nodes. Consider now a tree T with $k + 1$ nodes and let v be a leaf in this tree and u its parent.

Consider now the binary tree T' obtained by removing v from the our tree. The new tree T' will have k nodes, and by the induction hypothesis, we know that it satisfies the equation $n(T') = l(T') - 1$.

Given that the original was a binary tree, one of two things must occur.

1. If u had no other child, then it becomes a leaf, thus $n(T') = n(T)$ and $l(T') = l(T)$.
2. If u had another child, then it now has a single child node, thus $n(T') = n(T) - 1$ and $l(T') = l(T) - 1$.

In both situations we get

$$n(T) = l(T) - 1. \qquad \Diamond$$

Task 3 - Matching DFS and BFS

Let $G = (V, E)$ be a connected graph and $u \in V$ one of its vertices. Suppose we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted, again, at u, and obtain the same tree T. Prove that $G = T$.

Solution. Take an edge $e = (a, b)$ in G but not in T. Since T is a depth-first search tree, one of the two ends of e must be an ancestor of the other. Without loss of generality, say a is an ancestor of b. Since T is a breadth-first search tree, the distance of a and b from u in T can differ by at most one.

But if a is an ancestor of b, and the distance from u to b in T is at most one greater than the distance from u to a, then a must in fact be the direct parent of b in T. From this it follows that (a, b) is an edge of T, contradiction our initial assumption that (a, b) did not belong to T. $\Diamond$