Traversal + Counting = Solution

**Question 1** (KT 3.9). There’s a natural intuition that two nodes that are far apart in a communication network—separated by many hops—have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here’s one that involves the susceptibility of paths to the deletion of nodes.

Suppose that an n-node undirected graph \( G = (V, E) \) contains two nodes \( s \) and \( t \) such that the distance between \( s \) and \( t \) is strictly greater than \( n/2 \). Show that there must exist some node \( v \), not equal to either \( s \) or \( t \), such that deleting \( v \) from \( G \) destroys all \( s \) – \( t \) paths. In other words, the graph obtained from \( G \) by deleting \( v \) contains no path from \( s \) to \( t \). Give an algorithm with running time \( O(m+n) \) to find such a node \( v \). Describe your algorithm in prose and prove that it works correctly. Make sure your algorithmic description is clear and concise. As a hint, imagine performing a breadth-first search from \( s \). How large is each layer \( L_i \), along the way to \( t \)?

**Tree Traversal & Graph Metrics**

It might seem odd to apply BFS or DFS to a tree—you just get the tree you started with! However, the traversal of the tree in a particular order can allow for efficient computation of useful quantities. To see this, let’s define some quantities.

**Definition 1.** The distance between vertices \( u \) and \( v \) in a graph \( G = (V, E) \), denoted \( \text{dist}(u, v) \), is given by \( \text{dist}(u, v) = \min \{ k : \text{there is a } u - v \text{ path of length } k \} \).

**Definition 2.** The diameter of a graph \( G \), denoted \( \text{diam}(G) \), is given by \( \text{diam}(G) = \max \{ \text{dist}(u, v) : u, v \in V \} \).

That is, the diameter of \( G \) is the distance between two “farthest-apart” vertices in \( G \).

**Question 2.** Give a linear time, that is \( O(n) \)-time, algorithm to find the diameter of a tree \( T = (V, E) \). Prove that your algorithm is correct. Suggestion. Consider modifying recursive DFS so that it also computes, for each vertex \( v \)

(a) The diameter of the subtree of \( T \) rooted at \( v \)

(b) The longest path from \( v \) to a leaf in the subtree of \( T \) rooted at \( v \)

Then show how, knowing this information for all children of some vertex \( u \), we can determine this information for \( u \) itself. (Why do you think we need to know both (a) and (b)?)

**Question 3.** Until recently, there was no known method for computing the diameter of a graph that didn’t first compute the shortest path between all pairs of nodes. When graphs are dense, all-pairs shortest paths is fairly expensive, so some people have explored quicker algorithms which estimate the diameter of the graph. Develop a linear-time algorithm that, given a graph \( G \), returns a diameter estimate that is always within a factor of \( 1/2 \) of the true diameter. That is, if the true diameter is \( \text{diam}(G) \) then you should return a value \( k \) where \( \text{diam}(G)/2 \leq k \leq \text{diam}(G) \).

**Applications of Traversals**

**Question 4.** Answer Question 11 from Chapter Three of your text Algorithm Design. In addition to designing the algorithm, justify its correctness and time and space requirements.

**Question 5.** In class we outlined a topological sorting algorithm that would produce, for any DAG \( G = (V, E) \), a total ordering of the vertices that was consistent with all of the edge directions; that is, an ordering \( v_1, \ldots, v_n \) such that every edge \( e = \{v_i, v_j\} \) satisfies \( i < j \). This question asks you to consider some of the more subtle aspects of that algorithm. So, before trying to solve this problem, please read Section 3.6 of your text carefully. Go on. Do it now. I’ll wait....

Welcome back. Recall that the algorithm, essentially, works as described in the pseudo-code on the next page. What makes it efficient is how Steps 3 and 5 are implemented (and integrated with one another). Done properly, the algorithm requires only \( O(n + m) \) space and time, in part because each vertex and edge are inspected only a constant number of times.
Algorithm 1 Topological Sort

Require: A DAG \( G = (V, E) \)
Ensure: An ordered list \( L = v_1, \ldots, v_n \) such that every edge \( e = \{v_i, v_j\} \) satisfies \( i < j \).

1: Create an empty list \( L \)
2: while \( V \neq \emptyset \) do
3: Select a \( v \in V \) with \( \text{indeg}(v) = 0 \)
4: Add \( v \) to the end of \( L \)
5: Delete \( v \) from \( V \) and all edges incident with (containing) \( v \) from \( E \)
6: end while
7: return \( \mathit{L} \)

a) Is it necessary to delete \( v \) and its edges from \( G \)? Precisely, suppose we have sufficient space to store \( G \) along with only an additional \( O(n) \) of space available (\( n \) is the number of vertices), but not enough space to copy \( G \). Suppose further that we do not want to delete \( G \) in the process of creating \( L \). Is it possible, with only minor modifications to the algorithm, to successfully compute \( L \) in \( O(n + m) \) time and space?

b) Could we have selected which vertex to remove next by looking for those with out-degree 0 instead of in-degree 0 (obviously we would then be adding new vertices to the beginning of \( L \) and not the end!)? I’d like you to consider this question. Precisely,

i) Suppose we change line 3 of the algorithm so that vertices of out-degree 0 (instead of in-degree 0) are selected, and we change line 4 so that vertices are added to the beginning of \( L \). How would this impact the running time of the algorithm? Note: You don’t have enough room to copy \( G \) or make the graph \( G_{\text{rev}} \).

Extra Credit Challenge Problem!

Question 6. There’s another well-known algorithm to compute the diameter of a tree that may surprise you. Here it is

- Pick any vertex \( v \) in \( T \)
- Find a vertex \( u \) that maximizes \( \text{dist}(v, u) \), using a traversal, for example
- Find a vertex \( w \) that maximizes \( \text{dist}(u, w) \)
- Return \( \text{dist}(u, w) \) as the diameter of \( T \).

Prove that this algorithm is correct. Suggestion: Draw some pictures. Consider the sub-tree \( T' \) that consists of the (unique!) paths from \( v \) to \( u \) and \( u \) to \( w \). Imagine some longer path exists—how does it connect to \( T' \)?