CPSC 320 2024W1: Spanning Tree Tutorial Problem

A spanning algorithm

Let $G = (V, E)$ denote a connected, undirected graph with $n \geq 2$ nodes and m weighted edges. Let $\text{wt}(e)$ denote the weight of edge e of G. The following algorithm is similar but not identical to Kruskal's minimum spanning tree algorithm. (This version of the algorithm is interesting because it can be implemented efficiently on a multi-processor computer. Roughly this is because the steps for each connected component C can all be handled by different processors.)

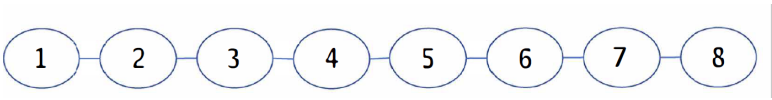
Algorithm Spanning($G = (V, E)$, wt())

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
  E-new $= \emptyset$
  For each connected component C of $G' = (V, E')$
    Find an edge $e = (u, v) \in E$ of minimum weight $\text{wt}(e)$ that
    connects a node u in C to a node v that is *not* in C
    E-new $= E$-new $\cup \{e\}$
  $E' = E' \cup E$-new
Return G'

1. How many iterations of the While loop will be executed in the *worst* case?
   ○ $\Theta(1)$ ● $\Theta(\log n)$ ○ $\Theta(n)$

2. Describe an input graph with n nodes on which this worst case behaviour can happen, and explain briefly what edge choices would lead to the worst case behaviour.

   Consider the following graph (assume all edges have weight 1):



   (we can easily extend it to larger numbers of nodes). Suppose that at each stage the components are "paired up" and that each component chooses the edge that connects to the component it's paired with. For instance, during the first iteration, A and B would both pick the edge (A, B), C and D would both pick the edge (C, D), etc. During the second iteration, the component $\{A, B\}$ would pick the edge (B, C), as would the component $\{C, D\}$. And finally during the third iteration both components $\{A, B, C, D\}$ and $\{E, F, G, H\}$ would pick the edge (D, E). In each iteration the number of connected components is halved, and so the algorithm will need $\lg n$ stages before it gets to a single connected component, for n a power of 2. When n is not a power of 2, at least $\lg n - 1$ stages are needed.

3. Which of the following statements is true? Choose one, and justify your answer briefly.

   ○ The algorithm always returns a tree on *all* inputs G.

   ● The algorithm returns a tree on *some* inputs G but may not return a tree on other inputs.

   ○ The algorithm /never/ returns a tree on any input G.

   Clearly the algorithm returns a tree whenever the graph G is a tree. On the other hand, consider a graph with vertices A, B and C, and edges (A, B), (B, C) and (C, A) all of which have weight 1. If in the first iteration A picks the edge (A, B), B picks the edge (B, C) and C picks the edge (C, A), then the algorithm will produce the original graph G, which is a triangle.

4. How many iterations of the While loop will be executed in the *best* case?

   ● $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$

5. Describe an input graph with n nodes on which this best case behaviour can happen, and explain briefly what edge choices would lead to the best case behaviour.

   This can happen for any complete graph where all edges have weight 1 if in the first iteration every vertex except one picks the edge that connects it to the remaining vertex.

6. Explain why the algorithm always returns a tree on all inputs $G = (V, E)$ where all edges of E have different weights.

   Suppose to the contrary that the algorithm does not return a tree. Consider the iteration where a cycle is first created, connecting k previously disjoint components $C_1, C_2, \ldots, C_k$. Let w_i be the weight of the edge connecting C_i with C_{i+1}, $1 \leq i \leq k-1$ and w_k be the weight of the edge connecting C_k with C_1.

   Without loss of generality (renumber nodes if needed), assume that w_1 is the min weight of the w_i. So C_1 and C_2 "choose" the edge with weight w_1. It follows that C_3 must have chosen the edge with weight w_2 (since C_2 didn't choose that edge). Since C_2 did not choose that edge, it must be that $w_2 > w_1$. Continuing on, C_{i+1} must have chosen the edge with weight w_i, $1 \leq i \leq k-1$, and $w_i > w_{i-1}$. But then, neither C_k nor C_1 choose the edge with weight w_k, a contradiction.

7. Explain why the tree returned by the algorithm is a minimum spanning tree on all inputs $G = (V, E)$ where all edges of E have different weights.

   We can use the Cut Property (Kleinberg and Tardos, 4.17): Assume that all edge costs are distinct. Let S be any subset of nodes that is neither empty nor equal to all of V, and let edge $e = (v, w)$ be the minimum-cost edge with one end in S and the other in $V - S$. Then every minimum spanning tree contains the edge e.

   Every edge added by algorithm Spanning satisfies the cut property, where at each iteration, each connected component C is a subset that is neither empty nor equal to all of V, and the edge found has minimum weight (cost). So, every such edge is in every minimum spanning tree.