Due: 1 pm on Wednesday, March 19

Assignment 4

• **NOTE**
We will refer to "design an algorithm, prove its correctness and analyze the running time" as a **3-part solution**.

1 Dominating Set in a Tree

Let $T = (V, E)$ be an undirected tree with $|V| = n$ nodes. The distance between two nodes u and v, denoted $d(u, v)$, is the number of edges on the unique shortest path between them.

A subset $S \subseteq V$ is called a **k-dominating set** of T if every node $v \in V$ satisfies:

$$\exists s \in S \, : \, d(v, s) \le k.$$

Give an algorithm that find a k-dominating set S with the minimal cardinality $|S|$. **Please give a 3-part solution**.

2 Bottleneck of Spanning Trees

Let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all *distinct*. Let $G' = (V, T)$ be a spanning tree of G; we define the *bottleneck edge* of G' to be the edge in T with the greatest cost. A spanning tree $G' = (V, T)$ is a *minimum-bottleneck spanning tree* if there is no other spanning tree of G with a cheaper bottleneck edge.

1) Is every minimum-bottleneck spanning tree of G a minimum spanning tree of G? Prove or give a counterexample.

2) Is every minimum spanning tree of G a minimum-bottleneck spanning tree of G? Prove or give a counterexample.

3 Minimum Spanning k-Forest

Given a graph $G = (V, E)$ with nonnegative weights, a spanning k-forest is a cycle-free collection of edges $F \subseteq E$ such that the graph with the same vertices as G but only the edges in F has k connected components. For example, consider a fully connected graph $G = (V, E)$ with vertices $V = \{A, B, C, D, E\}$. One spanning 2-forest of this graph is $F = \{(A, C), (B, D), (D, E)\}$, because the graph with vertices V and edges F has components $\{A, C\}, \{B, D, E\}$

The minimum spanning k-forest is defined as the spanning k-forest with the minimum total edge weight. (Note that when $k = 1$, this is equivalent to the minimum spanning tree). In this problem, you will design an algorithm to find the minimum spanning k-forest. For simplicity, you may assume that all edges in G have *distinct* weights.

1) Define a j-partition of a graph G to be a partition of the vertices V into j (non-empty) sets. That is, a j-partition is a list of j sets of vertices $\Pi = \{S_1, S_2 \dots S_j\}$ such that every S_i includes at least one vertex, and every vertex in G appears in exactly one S_i. For example, if the vertices of the graph are $\{A, B, C, D, E\}$, one 3-partition is to split the vertices into the sets $\Pi = \{\{A, B\}, \{C\}, \{D, E\}\}$. Define an edge (u, v) to be crossing a j-partition $\Pi = \{S_1, S_2 \dots S_j\}$ if the set in Π containing u and the set in Π containing v are different sets. For example, for the 3-partition $\Pi = \{\{A, B\}, \{C\}, \{D, E\}\}$, an edge from A to C would cross Π. Prove that for any j-partition Π of a graph G, if $j > k$ then the lightest edge crossing Π must be in the minimum spanning k-forest of G.

2) Give an efficient algorithm for finding the minimum spanning k-forest. **Please give a 3-part solution**.

4 Clock Tree Synthesis

In the design automation of VLSI chips, clock trees are crucial components. Consider the following model of clock tree synthesis (CTS), given a complete binary tree with n leaves, where n is a power of two. Each edge e of the tree has an associated, positive length l_e. The *distance* from the root to a given leaf is the sum of the lengths of all the edges on the path from the root to the leaf.

The root generates a *clock signal* which is propagated along the edges to the leaves. We assume that the time it takes for the signal to reach a given leaf is proportional to the distance from the root to the leaf. In realistic circuits, the distance is the net delay caused by parasite resistance/capacitance (RC), and the leaves represent sequential logic instances (e.g., flip-flops), which must be completely *synchronized*, i.e., all should receive the clock signal at the same time. To make this happen, we will have to increase the lengths of certain edges, so that all root-to-leaf paths have the same length (we're not able to shrink edge lengths). If we achieve this, then the tree (with its new edge lengths) will be said to have *zero skew*. Our goal is to achieve zero skew in a way that keeps the sum of all the edge lengths as small as possible.

Give an algorithm that increases the lengths of certain edges so that the resulting tree has zero skew and the total edge length is as small as possible. **Please give a 3-part solution.**

Example. Consider the tree in Fig. 1, in which letters name the nodes and numbers indicate the edge lengths. The unique optimal solution for this instance would be to take the three length-1 edges and increase each of their lengths to 2. The resulting tree has zero skew, and the total edge length is 12, the smallest possible.

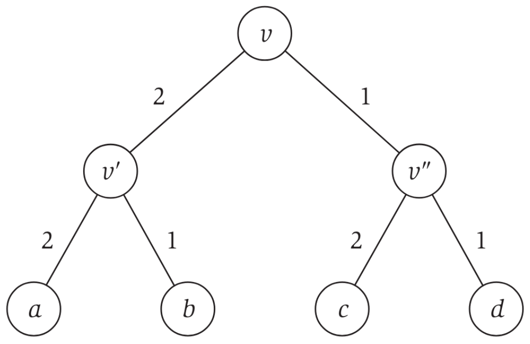

Figure 1: An instance of the zero-skew CTS problem.