4.22:

Answer:

We can provide a counterexample to show that T itself is not necessarily a minimum-cost spanning tree (MST). Suppose we have an undirected graph $G = (V, E)$ with four vertices v_1, v_2, v_3, v_4 and edges $e_1 = \{v_1, v_3\}, e_2 = \{v_1, v_4\}, e_3 = \{v_2, v_3\}, e_4 = \{v_2, v_4\}, e_5 = \{v_3, v_4\}$. All edges have cost 10 except for e_3, which has cost 5.

Consider the case when $T = \{e_1, e_5, e_4\}$. Each edge $e \in T$ belongs to an MST. Namely, the MST's $\{e_1, e_2, e_3\}, \{e_3, e_4, e_2\}, \{e_5, e_3, e_2\}$, which each have a cost of 25. However, T has a cost of 30 and hence is not an MST. QED.

4.25:

Answer:

The given set of points $P = \{p_1, p_2, ..., p_n\}$ and distance function d on the set P form an undirected graph $G = (P, E)$ that is both weighted and complete. Each edge $e_{ij} = \{p_i, p_j\} \in E$ has weight $w = d(p_i, p_j) > 0$. The algorithm to build a hierarchical metric τ on P can be described as follows.

Let T be the tree associated with τ. We place each of the n points in P at leaf nodes v_i in T. We run Kruskal's algorithm to find an MST on G, inserting edges from E in order of increasing weight as long as it does not create a cycle. Each time we insert an edge $\{p_x, p_y\}$ of weight w in Kruskal's algorithm, we also create a new node v_{xy} in T, make it the parent of v_x (associated with p_x) and v_y (associated with p_y), and assign v_{xy} a height of $h_{v_{xy}} = w$. We continue this process to build T from the bottom level up. When we insert an edge of weight w that merges components C_a and C_b in Kruskal's algorithm, we also create a new node v_{ab} in T, make it the parent of the subtrees that (by induction) consist of C_a and C_b, and assign v_{ab} a height of $h_{v_{ab}} = w$.

(i) τ is consistent with d
We can prove that for all pairs i, j we have $\tau(p_i, p_j) \leqslant d(p_i, p_j)$. Notice that for any p_i and p_j, $h_v = \tau(p_i, p_j)$ is equal to the weight of the edge that first merged the components containing p_i and p_j when Kruskal's algorithm was run on G. If the direct edge $\{p_i, p_j\}$ was considered before p_i and p_j were merged into the same component, then $\tau(p_i, p_j) = d(p_i, p_j)$. If p_i and p_j were merged into one component with an edge of weight w in Kruskal's algorithm before the direct edge was considered, then $w = \tau(p_i, p_j) \leqslant d(p_i, p_j)$ because Kruskal's algorithmoperates in order of increasing weight.

(ii) if τ' is any other hierarchical metric consistent with d, then $\tau'(p_i, p_j) \leqslant \tau(p_i, p_j)$ for each pair of points p_i and p_j.
We can prove this by contradiction. Assume τ' is another hierarchical metric such that $\tau'(p_i, p_j) > \tau(p_i, p_j)$. Let T' be the tree associated with τ', v' be the least common ancestor (LCA) of p_i and p_j in T', and T'_i and T'_j be the subtrees below v' containing the nodes associated with p_i and p_j. Let $h'_{v'}$ be the height of v' in T' and by our assumption $\tau'(p_i, p_j) = h'_{v'} > \tau(p_i, p_j)$. Notice there is a path M from p_i to p_j in the MST we constructed from G in our algorithm above because a tree is connected by definition. Because p_j is not in the subtree T'_i, M must contain a point that is not in T'_i. Let m be the first point along M not in T'_i, and m' be the point directly before m on M (m' is a point in T'_i). Then $d(m, m') \geqslant h'_{v'} > \tau(p_i, p_j)$ by our assumption and because the least common ancestor (LCA) of m and m' must have height greater than or equal to $h'_{v'}$. However, all edges of M were already present when Kruskal's algorithm merged the components containing p_i and p_j. So each edge must have length less than or equal to $\tau(p_i, p_j)$ and we have our contradiction as desired. $\Rightarrow\Leftarrow$.

Finally, note that this is a polynomial-time algorithm. We used Kruskal's algorithm to construct T, so it has running time $O(mlogn)$ on a graph with n nodes and m edges.

4.28:

Answer:

The criteria for a valid solution for CluNet is a spanning tree T of G with exactly k X-edges (the remaining edges will be owned by Y). The algorithm can be described as follows. First, we assign all X-edges to have weight 1 and Y-edges to have weight 10. Then we run a minimum spanning tree algorithm, which yields a spanning tree T_i that contains the maximum number of X-edges (due to the weights). Let i be the maximum number of X-edges. Next, we assign all X-edges to have weight 10 and Y-edges to have weight 1. Again, we run a minimum spanning tree algorithm, which this time yields a spanning tree T_j that contains the minimum number of X-edges. Let j be the minimum number of X-edges.

Case 1:
If $k < j$ or $k > i$, we return false because there is no valid solution.

Case 2:
If $k = j$, we return the spanning tree with the minimum number of X-edges.

Case 3:
If $k = i$, we return the spanning tree with the maximum number of X-edges.

Case 4:
If $j < k < i$, we can find the spanning tree with exactly k X-edges and return it. We do this by adding a X-edge to T_j from T_i to form a cycle whose edges are not all X-edges. Then we delete a Y-edge and we have found T_{j+1}. We repeat this process until we find T_k, then return it.

Note this is a polynomial-time algorithm. The two passes through a minimum spanning tree algorithm take $2 * n^2$ steps in the worst case, depending on implementation details. Cases 1 - 3 run in linear time because we need to count the number of X-edges and Case 4 takes at most n^2 steps to search for where to insert and delete edges. Hence, the total running time is $O(2n^2 + n + n^2) = O(3n^2 + n) = O(n^2)$.

4.30:

Answer:

The problem stipulates that $X \subseteq V$ is the set of k terminals that must be connected by edges and hence $|X| = k$. Let $Y = Z - X$, so Y is the set of non-terminal nodes in the minimum-weight Steiner tree T on $X \cup Z \subseteq V$. Notice that any node v in Y must have degree greater than 2 in T. If v had 1 edge, we should remove v from Y to get a lower weight Steiner tree. If v had 2 edges, again we should remove v from Y, this time adding an edge between the two neighbors of v. This yields a new Steiner tree with weight less than or equal to the original Steiner tree because the weights on G satisfy the triangle inequality $w_{ik} \leqslant w_{ij} + w_{jk}$. Let s be the number of nodes in our Steiner tree. Because the sum of the degrees in the tree is $2s - 2$, we know that the number of leaves is at least the number of nodes of degree greater than 2. Therefore $|Z| \leqslant k$ and we know how to find the minimum-weight Steiner tree. We can compute the MST on all sets $X \cup Z$ with $|Z| \leqslant k$ and the one with the lowest weight will be the minimum-weight Steiner tree. There is a maximum of $\binom{n}{2k}$ such sets to check, so the running time of the algorithm to find a minimum-weight Steiner tree on X will be $O(n^{O(k)})$, as desired.