Jason Lai
204995126
Dis 1B

CS 180
May 1, 2019

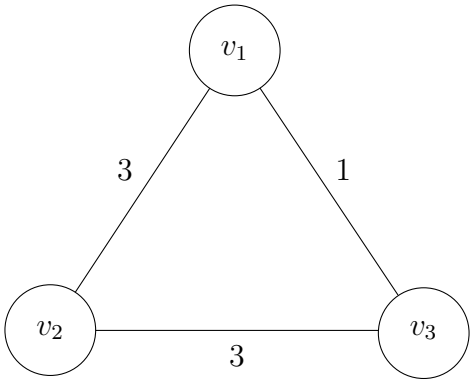
Homework 4

1 Problem 22

Consider any undirected graph $G = (V, E)$, with a cost $ce \geq 0$ on each edge, where the costs may not all be different. Then, given a spanning tree $T \subseteq E$ with the guarantee that for every $e \in T$, e belongs to some minimum-cost spanning tree in G, we cannot conclude that T itself is a minimum-cost spanning tree in G.

For instance, consider a graph K with three vertices v_1, v_2, v_3, and edges (v_1, v_2), (v_2, v_3), both of cost 3 and a single edge (v_1, v_3) of cost 1. Then, it can be observed that all edges of graph K belong to a possible minimum-cost spanning tree, however, the spanning tree consisting only of the two edges of cost 3 would not be a minimum-cost spanning tree, thereby showing that it cannot be concluded that a spanning tree with edges that belong to different, possible minimum-cost spanning trees is also a minimum-cost spanning tree itself.

Graph K



2 Problem 25

Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$, together with a distance function d on the set P, where d is simply a function on pairs of points in P with the properties that $d(p_i, p_j) = d(p_j, p_i) > 0$ if $i \neq j$, and that $d(p_i, p_i) = 0$ for each i, we want to construct a hierarchical metric τ when given d using an algorithm that runs in polynomial time. A hierarchical metric is understood to be, given a tree T, where each node of T has a height associated with it starting from the leaves being at height 0 and increasing at each level until the root at the tallest height is met, a hierarchical metric takes two leaves and returns the least common ancestor of the two nodes. In effect, taking the set of points P to be the leaves of tree T, then the hierarchical metric τ should be defined such that $\tau(p_i, p_j) = h_v$, where h_v is the height of the least common ancestor v of the points p_i and p_j. In essence, the hierarchical metric τ should be analogous to the distance function d, however, it must also meet the following constraints: τ is both (i) consistent with d, meaning that for all pairs i, j, we have $\tau(p_i, p_j) \leq d(p_i, p_j)$, and (ii), if given another hierarchical metric τ' that is also consistent with d, then $\tau'(p_i, p_j) \leq \tau(p_i, p_j)$ for each pair of points p_i and p_j.

To build a hierarchical metric τ from the distance function d on the set of points P, we use Kruskal's algorithm with initially disconnected points from P that are the subtrees that make up the forest F. Using Kruskal's algorithm, each time two subtrees are connected, instead of drawing an edge connecting the two subtrees, a new node is inserted as the parent to the two subtrees and edges are drawn from that node to the root of the two subtrees; the parent node is at height $d(p_i, p_j)$. Since this is mostly just Kruskal's algorithm, this runs in polynomial time. This is consistent with d, as the least common ancestor between two points p_i and p_j is at most at height $d(p_i, p_j)$, so $\tau(p_i, p_j) \leq d(p_i, p_j)$. Now consider a different hierarchical metric τ' where $\tau'(p_i, p_j) > \tau(p_i, p_j)$, however since by the algorithm, the least common ancestor of p_i and p_j is at most $d(p_i, p_j)$, then $\tau'(p_i, p_j)$ is at most $\tau(p_i, p_j)$, hence $\tau'(p_i, p_j) \leq \tau(p_i, p_j)$.

3 Problem 28

A polynomial-time algorithm that takes G, with each edge labeled X or Y, and either returns a spanning tree with exactly k edges labeled X, or reports correctly that no such tree exists is as follows:

Given a connected graph G with n vertices and m edges, then first, label all edges e with the same weight. Then, using a greedy algorithm that finds minimum spanning trees, such as Prim's or Kruskal's (ideally Prim's), find all possible minimum spanning trees by starting at any arbitrary vertex that hasn't been used yet. Since all edges e in the graph G have the same weight, all possible spanning trees in G are minimum spanning trees, so this algorithm produces all possible spanning trees for the graph G in the worst case. As each spanning tree is found, keep track of how many edges belonging to company X have been included. Should a spanning tree be constructed with exactly k edges, then return this tree. Otherwise, continue this algorithm until all vertices have been used as the starting vertex; should the algorithm terminate after starting with every possible vertex without finding a tree that has exactly k edges belonging to company X, then the algorithm reports that no such tree exists.

When this algorithm is used on a graph G with n vertices and m edges, the time complexity is approximately $O(n^3 + m) \approx O(n^3)$, which is in polynomial time (granted, this is really slow). This comes from the fact that the labelling of weights for all edges in G will take $O(m)$ time and, in the worst case, to execute an minimum spanning tree algorithm the naive way will take $O(n^2)$ time for each of the n vertices, which will take $O(n^3)$ time, hence $O(n^3 + m)$. However, through the use of binary heaps, the algorithm can be reduced to $O(nm\log n + m)$ $\approx O(n^2\log n)$, from the improvement of the minimum spanning tree algorithm to $O(m\log n)$. Furthermore, in the case of Prim's algorithm, the use of Fibonacci heaps can further reduce the minimum spanning tree algorithm to $O(m + \log n)$, so a fully optimized algorithm can improve the complexity to $O(m(n + 1) + n\log n) \approx O(n^2)$.

4 Problem 30

Given a graph $G = (V, E)$ with n nodes, wherein each pair of nodes is joined by a positively weighted edge (a complete graph), and these weighted edges satisfy the triangle inequality, we want to find a minimum-weight Steiner tree on X in $O(n^{O(k)})$ time, where X is set of k terminals in the set of V. A Steiner tree on X is understood to be, given a subset X of k terminals where $X \subseteq V$, then a tree T that spans through the set of X is a Steiner tree; however, this means that the possible Steiner tree can include nodes in the graph G not included in the subset X, so we call the set of additional nodes and the terminals of X to be a larger subset Z such that $X \subseteq Z \subseteq V$, and the Steiner tree is the spanning subtree of the subgraph $G[Z]$.

To find the minimum-weight Steiner tree, we first observe that the set of nodes Z of the Steiner tree is composed of two subsets: the subset X of k terminals and the subset Y of the additional nodes used to connect the k terminals of X. Since the Steiner tree is a spanning subtree of the subgraph $G[Z]$, and $G[Z]$ is a subgraph of the complete graph G, then the set of nodes Z in $G[Z]$ all have at least degree 3, since it is a complete graph and satisfies the triangle inequality. However, when finding the Steiner tree on the k terminals of X, only the additionals nodes Y have at least degree 3, since a property of undirected graphs implies that a node of degree 2 can have its two incident edges replaced with a single edge between the two adjacent nodes. Given that the k terminals have a degree less than 3, it follows that the number of additonal nodes Y is less than k, since the additional nodes Y have at least degree 3, which means they connect more terminals than additional nodes. Then, to find the minimum-weight Steiner tree, use a minimum spanning tree algorithm, such as Kruskal's or Prim's, on every possible combination of k terminals in X and additional nodes Y from the set of all n nodes in V. Since there are ${}_n\mathrm{C}_{2k}$ such sets to choose from, the time complexity of this algorithm is $\approx O(n^{O(k)})$.