Homework 5
Com S 311
Due: Nov 3, 11:59 pm

5% bonus for submission by Nov 2, 11:59 pm

No late submissions will be accepted

Note: The "Do not grade" option is not available for this HW

For all algorithm design problems, part of the grade depends on efficiency. For every graph $G = (V, E)$, the vertex set is $\{1, 2, \cdots, n\}$. We use n to denote the number of vertices and m to denote number of edges. Unless otherwise stated, all graphs are represented as adjacency lists. For weighted graphs, assume that each entry in the adjacency list for a vertex u is a tuple of the form $(v, c(u, v))$, where $c(u, v)$ denotes the cost associated with the edge (u, v).

1. (40 points) Let $G = (V, E)$ be an undirected, connected and weighted graph and let T be a minimum spanning tree (MST) for G. Let G' be a graph obtained by subtracting some positive value δ from the weight of an edge $e \in E$ that is not in T. Give an algorithm that, given G, T, e and δ, computes a MST of G'. Prove correctness of your algorithm and state its runtime.

   Ans.

   We will first prove the following claim:

   Let C be a cycle in G and let $e = (u, v)$ be an edge with largest cost. Then there is an MST to which e does not belong to. We will prove using an exchange argument. Let T be an MST that has $e = (u, v)$ as an edge. Remove e from T, and let S be the set of vertices that can be reached from u and R be the set of vertices that can be reached from v (after removing e). Note that S and R are disjoint and their uinion is V. Since C is a cycle in G, there is a path from u to v (without using the edge (uv)). This path will start at $u \in S$ and end at $v \in R$, thus there is an edge $e' = (a, b)$ on this path such that $a \in S$ and $b \in R$. Consider the cost of e', it must be the case that $c(e') \leq c(e)$ (since e is a an edge with largest cost). If we remove e from T and e' to T, then the resulting tree will be a spanning tree whose cost is at most the cost of T.

   This suggests the following algorithm:

   (a) Input: $G, T, e = (u, v), x$.

   (b) Find the path P from u to v in T using BFS.

(c) Add the edge e to P to obtain a cycle C.

(d) Let e' be the edge with largest cost in C. Remove e' from T and add e

Correctness follows from the claim that we have shown. Since T has n nodes and $n-1$ edges and performing BFS in T takes $O(n)$ time. Finding the edge with largest cost in C takes $O(n)$ time. Thus the total time taken is $O(n)$.

2. (20 points) Let $G = (V, E)$ be a weighted, undirected graph and let S be a subset of V. We define $MST(G, S)$ to be a minimum spanning tree for G with the additional property that every vertex in S is a leaf node. Give an algorithm that gets G and S as input and outputs $MST(G, S)$, if it exists. State the runtime of your algorithm. *If you wish, you may assume that all edge weights are distinct (as the text does when proving the Cut Property (4.17)).*

   Ans. There are two ways to interpret this question. i) Find a tree that is an MST of G and having all nodes from S as nodes, ii) Look at all spanning trees of G that have every node from S as leaf nodes, find a tree with smallest cost among all such trees. Both interpretations will be accepted.

   Algorithm for the first interpretation

   (a) Find an MST T using Prim's algorithm. Check if every node of S is a leaf node and output T if so. If not, output "not possible".

   Time taken is the time taken for Prim's algorithm which is $O(n + m \log n)$. Correctness (not asked) follows from the claim that if every edge cost is unique, then is only one MST. Think about proving this claim.

   Algorithm for the second interpretation

   (a) Remove S from G and let G' be the resulting graph.

   (b) For evert $x \in S$, let E_x be the set of edge from x to vertices in G'.

   (c) If at least on of E_x is empty, then output "Not possible" and quit,

   (d) Compute MST T for G'.

   (e) For every $x \in S$, let let $e_x \in E_x$ be the edge with smallest cost. For every $x \in S$, add e_x to to T,

   Computing G' takes $O(m+n)$ rime, computing E_x takes $O(m+n)$ time. Computing T takes $O(n + m \log n)$. Last step takes $O(m + n)$ time. Total time is $O(n + m \log n)$.

3. (20 points) Let $G = (V, E)$ be a directed, weighted graph with positive edge weights. You are given an integer array d of length n, where n is the number of vertices of G, and you are given a vertex s. You can assume that every vertex is reachable from s. Come up with an $O(m)$ algorithm to determine the following: Is it the case that for every vertex v, $d[v]$ is the length of the shortest path from s to v? Here "length" and "shortest" refer to the sum of the edge weights for the path, not the number of edges in the path. (*Tip:* Remember that if $s, u_1, u_2, \ldots, u_k, v$ is a shortest path from s to v, then $s, u_1, u_2, \ldots, u_k$ must be a shortest path from s to u_k.)

Ans. Check $d[s] = 0$ and check the the following equality holds for every $v \in V$ $(v \neq s)$

$$d[v] = \min\{d[u] + c(u, v) \mid (u, v) \in E$$

The implement the above algorithm, we need to compute incoming vertices for every vertex. This can be done in $O(m+n)$ time. Let S_v be the set of incoming nodes to v. Now the above equality can be checked by computing $d[u] + c(u, v)$ for evert $u \in S$. Thus the taken to per node v is the $O(|S_v|)$. Thus the total time taken is $O(m)$ as $\sum_{v\in V}(|S_v|$ is the sum of indgrees which is m. Total time is $O(m + n)$.

4. (40 points) A bakery gets orders for many different kinds of cakes with fancy decorations. They have plenty of guys who can decorate cakes (at least n of them), and plenty of supplies for decoration, but there is only one oven, and it can only hold one cake at a time! Each cake C_i requires b_i minutes of baking time by itself in the oven, plus d_i minutes of decoration time, which can occur in parallel with other cakes being decorated or baked. Write an algorithm that, given a set of n cakes, determines a schedule $C_{i_1}, C_{i_2}, \ldots, C_{i_n}$ that such that all the cakes are completely finished in the shortest overall time. Derive the runtime and prove the correctness of your algorithm.

   Ans. Greedy Algorithm: Arrange the cakes in decreasing order of decoration time. This the schedule. Time taken is $O(n \log n)$. We will use an exchange argument to prove the correctness of the algorithm. For the correctness, we first assume that all decoration times are distinct.

   Let $c_1, c_2, \cdots c_n$ be an optimal schedule O with bake and decoration times being b_i and d_i for c_i. We say that this schedule has an inversion if $i < j$ but $d_i < d_j$. Note that schedule produced by the greedy algorithm as no inversions. In this schedule, let $g(\ell)$ be the time at which cake c_ℓ is done baking, and let $f(\ell)$ be the time at which cake c_ℓ is done with baking and decorating. Note that $f(\ell) = g(\ell) + d_\ell$. Also note that $g(\ell) = g(\ell - 1) + b_\ell$. Let $S = \max\{f(1), f(2), \cdots f(k)\}$. Thus all cakes are finished within S time units. Since O is an optimal schedule we can not have another schedule in which all cakes are finished in less than S time units.

   Observation: If an optimal schedule has an inversion, then there exist k such that $d_k < d_{k+1}$. Proof: Since optimal schedule as an inversion, there exist $i < j$ such that $d_i < d_j$. Consider $d_i, d_{i+1}, d_{i+2}, \cdots d_j$. Since $d_i < d_j$ it can be the case that $d_i > d_{i+1} > d_{i+2} \cdots d_j$. Thus there is a k between i and j such that $d_k < d_{k+1}$.

   Consider a schedule O' which is same as O except that c_k and c_{k+1} are swapped. In this schedule, let $g'(\ell)$ be the time at which cake c_ℓ is done baking, and let $f'(\ell)$ be the time at which cake c_ℓ is done with baking and decorating. Let $S' = \max\{f'(1), f'(2), \cdots f'(k)\}$.

   Since O is an optimal schedule $S' \geq S$.

   Note that $g'(1) = g(1), g'(2) = g(2), \cdots g'(k-1) = g(k-1)$ and $g'(k+2) = g(k+2), g'(k+3) = g(k+3) \cdots g'(n) = g(n)$ and thus $f'(1) = f(1), f'(2) = f(2), \cdots f'(k-1) = f(k-1)$ and $f'(k+2) = f(k+2), f'(k+3) = f(k+3) \cdots f'(n) = f(n)$.

   How do $f(k), f(k+1), f'(k)$ and $f'(k+1)$ compare? Note that in the new schedule c_{k+1} is baked after c_{k-1} and in the old schedule c_{k+1} is baked after c_{k-1} and c_k are done. Thus $g'(k+1) < g(k+1)$ and thus $f'(k+1) < f(k+1)$.

The old schedule bakes in the order $c_1, c_2, \cdots c_{k-1}, c_k, c_{k+1}$ and the new schedule bakes in the order $c_1, c_2, \cdots c_{k-1}, c_{k+1}, c_k$. Thus the time at which c_{k+1} is done baking in the old schedule is exactly the time at which c_k is done baking in the new schedule. Thus $g'(k) = g(k+1)$. Thus $f'(k) = g(k+1) + d_k$. Note that $f(k+1) = g(k+1) + d_{k+1}$. Since $d_k < d_{k+1}$, we have $f'(k) < f(k+1)$.

To summarize, $f'(k+1) < f(k+1)$ and $f'(k) < f(k+1)$ and $f(\ell) = f'(\ell)$ for evert $\ell \neq k \ \text{ or } \ k+1$. Thus

$$S' = \max\{f'(1), f'(2), \cdots f'(k)\} \leq S = \max\{f(1), f(2), \cdots f(k)\}$$

Since O is an optimal schedule S' must equal S. This implies that O' is also an optimal schedule.

Thus if there is an optimal schedule x inversions, then there is an optimal schedule with $x - 1$ inversions. Applying this iteratively, we arrive at an optimal schedule with no inversions which is the schedule produced by the greedy algorithm. Finally, it is easy to remove the assumption that all decoration times are distinct. Let O be an optimal schedule with no inversions that is different from the greedy schedule. These schedules only differ in scheduling the cakes which the decoration times are the same. If we have two cakes c_1 and c_2 whose decoration times are d, then if we schedule c_1 first and c_2 later then the time at which both the cakes are done is $b_1 + b_2 + d$, if the schedule were c_2, c_1 then also both the cakes are done by $b_2 + b_1 + d$.