The Design and Analysis of Algorithms	Problem Set 2
CS 681 Fall 2003	Due Monday, September 29, 2002

Solve 4 of the following 5 problems. You may solve all 5 for extra credit. We will maintain a FAQ for the problem set on the course Web page.

(1) (15 points) Assume you are trying to create a spanning tree connected graph G, where each edge has a difficulty level (e.g., nodes are points on a map, and the difficulty level of the edge represents how difficult is the trail hike from one edge end the other). Difficulty levels are expressed as numbers, with higher number indicating a more difficult edge. Assume G has n nodes and m edges.

(a) Suppose you decide to view an edge *difficult* if its difficulty level is at least α. Give an $O(m)$ time algorithm that finds a spanning tree with as few difficult edges as possible.

(b) Now suppose that you want to distinguish k different levels of difficulties $\alpha_1 < \ldots < \alpha_k$. (E.g., old people view edges with difficulty at least α_1 difficult, really small kids view edges with difficulty level at least α_2 difficult, etc., while experienced hikers only view edges with difficulty level at least α_k difficult.) You would love to have a single tree that simultaneously has as few edges with difficulty at least α_i as possible for all i. Does such a tree always exists? If it exists, can you find it in $O(m \log n)$ time?

(2) (15 points) Let's go back to the original motivation for the minimum spanning tree problem: we are given a connected, undirected graph $G = (V, E)$ with positive edge lengths $\{\ell_e\}$, and we want to find a spanning subgraph of it. Now, suppose we are willing to settle for a subgraph $H = (V, F)$ that is "denser" than a tree, and we are interested in guaranteeing that for each pair of vertices $u, v \in V$, the length of the shortest u-v path in H is not much longer than the length of the shortest u-v path in G. By the *length* of a path P here, we mean the sum of ℓ_e over all edges e in P.

Here's a variant of Kruskal's algorithm designed to produce such a subgraph.

- First, we sort all the edges in order of increasing length. (You may assume all edge lengths are distinct.)
- We then construct a subgraph $H = (V, F)$ by considering each edge in order.
- When we come to edge $e = (u, v)$, we add e to the subgraph H if there is currently no u-v path in H. (This is what Kruskal's algorithm would do as well.) On the other hand, if there is a u-v path in H, we let d_{uv} denote the total length of the shortest such path; again, length is with respect to the values $\{\ell_e\}$. We add e to H if $3\ell_e < d_{uv}$.

In other words, we add an edge even when u and v are already in the same connected component, provided that the addition of the edge reduces their shortest-path distance by a sufficient amount.

Let $H = (V, F)$ be the subgraph of G returned by the algorithm.

(a) Prove that for every pair of nodes $u, v \in V$, the length of the shortest u-v path in H is at most 3 times the length of the shortest u-v path in G.

(b) Despite its ability to approximately preserve shortest-path distances, the subgraph H produced by the algorithm cannot be too dense. Let $f(n)$ denote the maximum number of edges that

can possibly be produced as the output of this algorithm, over all n-node input graphs with edge lengths. Prove that

$$\lim_{n\to\infty} \frac{f(n)}{n^2} = 0.$$

Hint: can you show that the minimum degree in this graph cannot be bigger than $\sqrt{n}$?

(3) (15 points) Consider a directed graph $G = (V, E)$ with a root $r \in V$ and nonnegative costs on the edges. In this problem we consider variants of the min-cost arborescence algorithm.

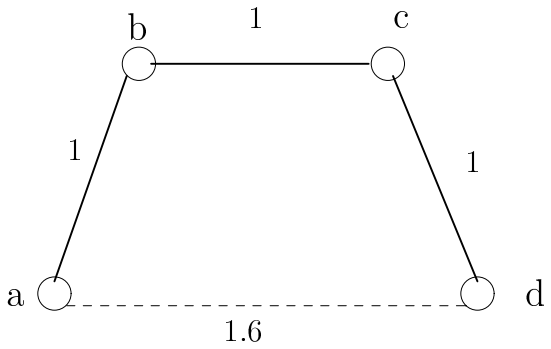
(a) The algorithm discussed in class works as follows: we modify the costs, consider the subgraph of zero-cost edges, look for a directed cycle in this subgraph, and contract it (if one exists). Argue briefly that instead of looking for cycles, we can instead identify and contract strongly connected components of this subgraph.

(b) In the course of the algorithm, we defined y_v to be the min cost of an edge entering v, and we modified the costs of all edges e entering node v to be $c'_e = c_e - y_v$. Suppose we instead use the following modified cost: $c''_e = \max(0, c_e - 2y_v)$. This new change is likely to turn more edges 0 cost. Suppose, now we find an arborescence T of 0 cost. Prove that this T has cost at most twice the cost of the minimum cost arborescence in the original graph.

(c) Assume you do not find an arborescence of 0 cost. Contract all 0-cost strongly connected components, and recursively apply the same procedure on the resulting graph till an arborescence is found. Prove that this T has cost at most twice the cost of the minimum cost arborescence in the original graph.

(4) (15 points) You are in charge of CluNet, a company working to build a service that will provide connectivity via a fiber-optic cable. They modeled the problem as a minimum cost spanning tree problem. There are n nodes that they need to connect, and they determined a set of m edges that they consider building (the other edges are either impossible to build, or are prohibitively expensive). These nodes and possible connections form an undirected graph $G = (V, E)$, and each possible edge $e \in E$ comes with a cost $c_e \geq 0$. They have also determined a minimum cost spanning tree T^* in this graph. However, to build the connection corresponding to edge e they need to enter negotiations with authorities in the region where the edge is traveling. Before they start these negotiations, they would like to know how important each edge is for the tree.

We define the *vitality* of an edge $e \in T^*$ as the increase in the cost of the MST caused by deleting edge e from the graph (i.e., the increased cost if the negotiations fail). The vitality is ∞ is after deleting edge e the graph is no longer connected. For example, in the graph below, the MST of the dark edges costs 3, and each edge in the tree has vitality .6 as any single edge can be replaced by the edge (a, d) at a .6 increase in the total cost.



(a) Assume you are given an MST T^* in the above graph G. Give an algorithm that computes the vitality of a given edge $e \in T^*$ in $O(m)$ time.
(b) Assume for now that the MST is a path. Give an algorithm that processes the path left to right, uses a priority queue (say a variant of a binomial heap) and computes the vitality of all edge of the path in $O(m \log n)$ time.
(c) Give an algorithm that computes the vitality of all edge of the MST in $O(m \log n)$ time. Hint: One can extend the solution from part (b) to trees. Alternatively, you can follow Kruskal's algorithm for MST, and compute vitalities at the times Kruskal encounters the alternate edges.

(5) (15 points) Consider the Fibonacci heap data structure from Wednesday's class (Sept 17th). Here we consider a delayed version of this data structure. It is implemented just like the regular Fibonacci heap with one change: we cut a node only after its *third* child is cut out (rather than the second). The cutting operation may cause cascading cuts up the tree as before.

Does this delayed version of the Fibonacci heap still have the same asymptotic cost for each operation as before? Either show that a sequence of a inserts, b delete-mins, c merges, and d decrements still costs $O(a + b \log n + c + d)$ time, or show that an arbitrary long sequence of operations can violate this bound.