CO453: Network Design – Winter 2007

Instructor: Chaitanya Swamy

Assignment 1

Due: January 24, 2007 after class

You must give a proof of correctness and a running-time analysis of any algorithm you design. The running time should be stated using the $O(.)$ notation: recall that we say $f(n) = O\big(g(n)\big)$ if there exist *constants* $n_0,\ c \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. All your algorithms must run in polynomial time unless otherwise stated. You may use any proof or algorithm done in class directly.

Throughout $G = (V, E)$ with $|V| = n,\ |E| = m$.

Q1 [from Kleinberg-Tardos with some modifications]:

Your friends are planning an expedition to a small town deep in the Canadian north. Having researched all the travel options, they have drawn up a directed graph whose nodes represent intermediate destinations, and edges represent the roads between them. They have also learned that extreme weather, especially in the winter, causes roads to become quite slow, and the time taken can depend on the time of the year or even the time of the day. Fortunately, they have found the ultimate website AskGod.com, which (among other things) answers the following type of query: given an edge $e = (u, v)$ and a proposed starting time t at location u, the website accurately returns the time $f_e(t)$ at which one will arrive at location v. The website guarantees that for every edge e, $f_e(t) \geq t$ (they assume you do not have access to a Starship Enterprise that can take you back in time), and that $f_e(.)$ is a monotonic increasing function (i.e, you do not arrive earlier by starting later). Other than that, the functions f_e may be arbitrary. For example, in areas where the travel time does not depend on the starting time, we would have $f_e(t) = t + c_e$, where c_e is the time needed to travel across the road represented by edge e.

Knowing your expertise in network design, your friends seek your help in determining the fastest way to travel though the directed graph from the starting point to their intended destination. Give a polynomial time algorithm for this problem. Treat each query to a function f_e as a single computational step. **(10 marks)**

Q2: In all parts below, assume that the edge costs are distinct.

(a) Consider the following algorithm for finding a minimum spanning tree (MST).

1) Initialize $T = \emptyset$.
2) If the graph consists of a single node, return T.
3) [Otherwise] For every node v, find $e = (u, v) \in E$ such that $c_{uv} = \min_{e=(u',v)\in E} c_e$ and add e to T.
4) Contract the components of the graph (V, T) into supernodes to get the graph $G' = (V', E')$.
5) Recursively run the algorithm to find a tree T' in G'.
6) Set $T \leftarrow T \cup T'$ and return T.

This algorithm is called *Boruvka's algorithm*. Either prove that Boruvka's algorithm finds an MST, or give a counterexample. **(5 marks)**

(b) Consider the following MST algorithm motivated by the cut-property proved in class.

> Find an edge $e = (u, v)$ that is the minimum-cost edge across some cut (A, B) where $B = V \setminus A$ and add e to the MST. Recursively run the algorithm on the subgraphs $(A, E[A])$ and $(B, E[B])$ to obtain trees T_A and T_B respectively, where $E[S]$ denotes all edges with both endpoints in S, that is, $E[S] = \{(u, v) \in E : u \in S, v \in S\}$. Return $\{e\} \cup T_A \cup T_B$.

Either prove that this algorithm finds an MST, or give a counterexample. **(5 marks)**

(c) The cut property allows us to determine when we can include an edge e in our solution. The following property, called the *cycle property*, allows us to determine when an edge can be rejected.

> **Cycle Property:** Assuming distinct edge costs, if e is an edge that is the (unique) most expensive edge in some cycle then e does not belong to *any* MST.

Prove the cycle property. **(5 marks)**

(d) Prove that the following algorithm finds an MST:

> Initialize $T \leftarrow \emptyset$. Consider edges of G in *arbitrary* order. For each edge $e = (u, v)$, if $T \cup \{e\}$ is acyclic, add e to T. Otherwise, let e' be the most expensive edge on the path in T between u and v; if $c_e < c_{e'}$, set $T \leftarrow T \cup \{e\} \setminus \{e'\}$. Return T.

(Do not forget to prove that T is a tree.) **(5 marks)**

Q3: Pick your favorite algorithm — Prim or Kruskal — and prove that it returns an MST even when the edge costs need not be all distinct. **(5 marks)**

(Hint: Try to show that the set T of edges maintained by Prim or Kruskal is a subset of an MST at all times.)

Q4: Having solved their problem of finding a route with smallest travel time, your friends return to you with renewed faith for help with the following problem. They are planning a hiking trip in the Canadian Rockies as part of their expedition. They have drawn up a graph of all their destinations of interest, and for every edge $e = (u, v)$, they have obtained from the same website AskGod.com, the average altitude a_e of the hike trail between locations u and v. Assume that the graph is connected and that all a_es are distinct.

(a) Your friends decide that a hike trail e is *difficult* if $a_e \geq \alpha$, where α is a given number that quantifies their hiking skill. In order to not make the hiking trip overly strenuous, they ask you to design a tree that connects all locations and which has the fewest number of difficult edges. **(5 marks)**

(b) (*) It turns out that your friends have a broad range of skill values, which is not captured by a single α-value. Instead, they have now come up with k different skill levels $\alpha_1 < \alpha_2 < \ldots < \alpha_k$ to describe their hiking skills: people with skill level α_1 are beginners, those with skill level α_k are

experts, etc. It would be wonderful if they could obtain a tree that *simultaneously* minimizes the number of edges with altitude at least α_i for every $i = 1, \ldots, k$. A further complication is that not all hike trails are cleared of snow. The rangers choose an altitude β and clear all hike trails having altitude $a_e \leq \beta$, but they choose β to guarantee that for every two locations u and v there is some u-v path such that all edges on this path are cleared of snow.

Now your friends approach you with the following problem. They would like you to design a tree $T = (V, F)$ that *simultaneously* minimizes the number of edges with $a_e \geq \alpha_i$ for every $i = 1, \ldots, k$. Furthermore, in order to ensure that all edges of T will be cleared of snow, they want that for every $u, v \in V$, if a is the maximum altitude of an edge on the unique path in T between u and v, then $a \leq \max_{e \in P} a_e$, for *every* path P between u and v in G. (Observe that this means that all edges on the u-v path in T will be cleared of snow.)

It is not at all clear if such a tree even exists for a given instance. Either give a polynomial time algorithm that constructs the desired tree, or give an example showing such a tree may not exist.

(10 marks)

You may assume that Prim's algorithm or Kruskal's algorithm work also for non-distinct edge costs, if necessary.