University of Toronto at Scarborough
CSCC73 - Algorithm Design and Analysis

Assignment #2: Greedy Algorithms Solutions

1. (10 marks) Consider a communications graph. Each edge of the connected graph $G = (V, E)$ represents a communication link between sites (represented as nodes). Each edge e has a bandwidth b_e.

   Each pair of nodes $u, v \in V$ needs to be able to communicate. For any $u, v - path$ P the *bottleneck transmission rate* $b(P)$ of P is the minimum bandwidth of any edge it contains. In other words, $b(P) = min_{e \in P} b_e$. The *best achievable bottleneck rate* for a pair $u, v \in V$ is the maximum, over all $u - v$ paths P in G, of the value $b(P)$. Our goal is to determine a set of $u - v$ paths for each pair $u, v \in V$ with best achievable bottleneck rate.

   Fortunately, we can construct a spanning tree T of G such that for every pair of nodes $u, v \in V$ the unique $u - v$ path in the tree T actually achieves the best achievable bottleneck rate for u, v in G.

   Give an efficient algorithm to construct such a spanning tree. Your algorithm should construct a spanning tree T in which, for each $u, v \in V$, the bottleneck rate of the $u - v$ path in T is equal to the best achievable bottleneck rate for the pair u, v in G. Prove the correctness of your algorithm.

   Sample Solutions.

   Claim. A minimum spanning tree, computed with each edge cost equal to the negative of its bandwidth is a spanning tree such that for each $u, v \in V$, the bottleneck rate of the $u - v$ path in T is equal to the best achievable bottleneck rate for the pair u, v in G.

   Proof. Let's first assume all edge weights are unique. Assume for a contradiction that the claim is not true. Then there exists some pair for nodes u, v for which the path P in the minimum spanning tree does not have a bottleneck rate as high as some other $u - v$ path P'. Let $e = (x, y)$ be an edge of minium bandwidth on the path P; note that $e \notin P'$. Furthermore, e has the smallest bandwidth of any edge in $P \cup P'$. Now, using the edges in $P \cup P'$ other than e it is possible to travel from x to y (for example, by going from x back to u via P, then to v via P', then to y via P). Thus, there is a simple path from x to y using these edges and so there is a cycle C on which e has the minimum bandwidth.

   This means that in our minimum spanning tree instance, e has the max. cost on the cycle C; but then the minimum spanning tree algorithm would not have included e but rather some other edge on the cycle C contradicting that we have a minimum spanning tree.

   Now, if the edge costs are not all distinct we need to find a way to slightly alter the edge weights without compromising the solution. Consider perturbing all edge bandwidths by extremely small amounts so they become distinct, and then defining a minimum spanning tree. We therefore refer, for each edge e to a *real bandwidth* b_e and a *perturbed bandwidth* b'_e. In particular, we choose perturbations small enough so that if $b_e > b_f$ for edges e and f, then also $b'_e > b'_f$. Our tree has the best bottleneck rate for all pairs, under the perturbed bandwidths. But suppose that the $u - v$ path P in this tree did not have the best bottleneck rate if we consider the original, real bandwidths; say there is a better path P'. Then there is an edge e on P for which $b_e > b_f$ for all edges f on P'. But the perturbations were so small that they did not cause any edges with distinct bandwidths to change the relative order of their bandwidth values, so it would follow that $b'_e > b'_f$ for all edges f on P', contradicting our conclusion that P had the best bottleneck rate with respect to the perturbed bandwidths.

   Complexity. Same as Prims or Dijkstra's or $\mathcal{O}(n + m \log n)$.

2. (10 marks) Given a set P of n people. Suppose the i^{th} person claims to know d_i other people in P. Determine in polynomial time if P is a feasible set by constructing a greedy algorithm. By feasible, we mean that it's possible for each person to know the number of people they claim to know. Prove that

your algorithm correctly determines if such a set up is possible. HINT: Think of the i^{th} person being represented by a vertex v_i and the number of people that the person knows as the degree d_i of v_i.

Sample Solutions

Clearly if any of the degrees d_i is equal to 0, then this must be an isolated node in the graph, so we can delete d_i from the list and continue by recursion on the smaller instance.

Otherwise, all d_i are positive. We sort the numbers, relabeling as necessary so that $d_1 \geq d_2 \geq \ldots \geq d_n$. We now consider the list of numbers:

$$L = \{d_1 - 1, d_2 - 1, \ldots, d_{d_n} - 1, d_{d_{n+1}}, \ldots, d_{n-2}, d_{n-1}\}$$

So we subtract 1 from each of the first d_n numbers and drop the last number. We claim that

> "*there exists a graph whose degrees are equal to the list $d_1, d_2, \ldots, d_n$ if and only if there exists a graph whose degrees form the list L.* "

Assuming this claim, we can proceed recursively.

Proof of claim.

First if there is a graph with degree sequence L, then we can add an n^{th} node with neighbours equal to the nodes $v_1, v_2, \ldots, v_{d_n}$ thereby obtaining a graph with degree sequence $d_1, \ldots, d_n$. Conversely, suppose there is graph with degree sequence $d_1, d_2, \ldots, d_n$ where we have $d_1 \geq d_2 \geq \cdots \geq d_n$. We must show that in this case, there is in fact such a graph where node v_n is joined to precisely the nodes $v_1, v_2, \ldots, v_{d_n}$. This will allow us to delete node n and obtain list L.

Consider any graph G with degree sequence $d_1, \ldots, d_n$; we show how to transform G into a graph where v_n is joined to $v_1, v_2, \ldots, v_{d_n}$. If this property does not already hold, then there exists $i < j$ so that v_n is joined to v_j but not v_i. Since $d_i \geq d_j$, it follows that here must be some v_k not equal to any of v_i, v_j, v_n with the property that (v_i, v_k) is an edge but (v_j, v_k) is not. We now replace these two edges by (v_i, v_n) and (v_j, v_k). This keeps all degrees the same; and repeating this transformation will convert G into a graph with the desired property.

Complexity

We take $\Theta(n \log n)$ time to sort the n degrees. For each vertex v_i we need to do update d_i vertices degrees. This works out to $\Theta(m)$ time for a total of $\Theta(n \log n + m)$ time.