CS 570
Fall 2008
HW4 Solution (revised)

1) Suppose you are a consultant for the networking company Clunet, and they have the following problem. The network that they are currently working on is modeled by a connected graph G=(V, E) with n nodes. Each edge e is a fiber-optic cable that is owned by one of two companies- creatively named X and Y- and leased to Clunet.
Their plan is to choose a spanning tree T of G, and upgrade the links corresponding to the edges of T. Their business relations people have already concluded an agreement with companies X and Y stipulating a number k so that in the tree T that is chosen, k of the edges will be owned by X and n-k-1 of the edges will be owned by Y.
Clunet management now faces the following problem: It is not at all clear to them whether there even exists a spanning tree meeting these conditions, and how to find one if it exists. So this is the problem they put to you: give a polynomial time algorithm that takes G, with each edges labeled X or Y, and either (i) returns a spanning tree with exactly k edges labeled X, or (ii) reports correctly that no such tree exists.

The algorithm:

```
Line 1    Remove all edges labeled X from G to get a new graph G’, which
          has k’ disconnected components.
Line 2    If k’ > k + 1
Line 3        return “no such tree exists”
Line 4    Else
Line 5        add exactly k’-1 edges labeled X in G’ to make G’
              connected
Line 6        remove all edges labeled Y in G’
Line 7        add exactly k-k’+1 edges labeled X in G’ subject to the
              constraint that no circle is formed in G’. If cannot find such
              edges, return “no such tree exists”.
Line 8        add exactly n-k-1 edges labeled Y in G’ subject to the
              constraint that no circle is formed in G’
Line 9    return G’, which is a spanning tree with exactly k edges labeled X.
```

2) Let us say that a graph G = (V, E) is a near-tree if it is connected and has at most n + 8 edges, where n = |V|. Give an algorithm with running time O(n) that takes a near-tree G with costs on its edges, and returns the minimum spanning tree of G. You may assume that all edge costs are distinct.

Algorithm:

Line 1: While (G is not a tree)
Line 2: find a circle in G
Line 3: find the largest edge *e* in that circle
Line 4: delete that edge *e* from G

3) A group of network designers at the communications company CluNet find themselves facing the following problem. They have a graph $G = (V, E)$, in which the nodes represent sites that what to communicate. Each edge *e* is a communication link, with a given available bandwidth b_e.
For each pair of nodes $u, v \in V$, they want to select a single *u-v* path *P* on which this pair will communicate. The *bottleneck rate b(P)* of the path *P* is the minimum bandwidth of any edge it contains; that is, $b(P) = \min_{e \in P} b_e$. The *best achievable bottleneck* rate for the pair *u, v* in *G* is simple the maximum, over all *u-v* paths *P* in *G*, of the value *b(P)*.
It's getting to be very complicated to keep track of a path for each pair of nodes, and so one of the network designers makes a bold suggestion: Maybe one can find a spanning tree *T* of *G* so that for every pair of nodes *u, v*, the unique *u-v* path in the tree actually attains the best achievable bottleneck rate for *u, v* in *G*. (In other words, even if you could choose any *u-v* path in the whole graph, you couldn't do better than the *u-v* path in *T*.)
This idea is roundly heckled in the offices of CluNet for a few days, and there's a natural reason for the skepticism: each pair of nodes might want a very different-looking path to maximize its bottleneck rate; why should there be a single tree that simultaneously makes everybody happy? But after some failed attempts to rule out the idea, people begin to suspect it could be possible.
Show that such a tree exists, and give an efficient algorithm to find one. That is: give an algorithm constructing 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*.

Answer: Use a modified Prim's Algorithm in which you grow the spanned tee (until complete) with a node v that maximizes the attachment cost.

1. Initially S = {s} and d(s) = 0
2. While S != V

3. Select a node v not in S with at least one edge from S for which d'(v) = max(l_e) for e(u,v): u in S
4. Add v to S
5. End while

Starting from any node s would result in the same tree being spanned, so spanning tree T always contains the best achievable bottleneck rate paths for u to v in G.

Let the Case |S| = 1; S = {s} and d(s) = 0
Assume it holds when |S| = k for some value of k >= 1. We now grow S to size k+1 by adding the node v. Let (u, v) be the final edge on our s-v path. Call this path $P_{v.}$ Suppose we have another path to v that is shorter. It would have crossed from S to V-S at some point, where the boundary edge would be from u' to v'. However, based on our greedy step in the algorithm, the shorter path u to v was picked instead, so it is contradiction to have a shorter path going through u' and v' to v.

4) In trying to understand the combinatorial structure of spanning trees, we can consider the space of all possible spanning trees of a given graph, and study the properties if this space. This is a strategy that has been applied to many similar problems as well.
Here is one way to do this. Let *G* be a connected graph, and *T* and *T'* two different spanning trees of *G*. We say that *T* and *T'* are *neighbors* if T contains exactly one edge that is not in *T'*, and *T'* contains exactly one edge that is not in *T*.
Now, from any graph *G*, we can build a (large) graph *H* as follows. The nodes of *H* are the spanning trees of *G*, and there is an edge between two nodes *H* if the corresponding spanning trees are neighbors.
Is it true that for any connected graph *G*, the resulting graph *H* is connected? Give a proof that *H* is always connected, or provided an example (with explanation) of a connected graph *G* for which *H* is not connected.

Answers: Yes, if a graph H contains multiple spanning trees, there must be at least 1 way to remove an unique edge e connecting nodes u and v and replacing it with another unique edge e' to connect u' to v, creating another connected graph. This would by definition be 2 neighbors, which results in always having a connected graph H.