

CS303 (Spring 2008)— Solutions to Assignment 5

Exercise 23.1-6

Here, we are looking at graphs that may have multiple edges of the same cost (otherwise, of course, the problem is trivial). First, let's assume that for each cut (S, \bar{S}) , there is a unique light edge (i.e., of smallest cost c_e) crossing the cut. Let's call it e_S . By the cut property, each such edge e_S must be part of every minimum spanning tree. Define $E' := \{e_S \mid (S, \bar{S}) \text{ is a cut}\}$. So for every minimum spanning tree T , by the above argument, we have $T \supseteq E'$. On the other hand, E' itself is already connected, because it contains an edge for every cut. Thus, no spanning tree T could possibly contain any edge beyond E' , since they would create a cycle otherwise. Thus, E' itself is the only MST of G .

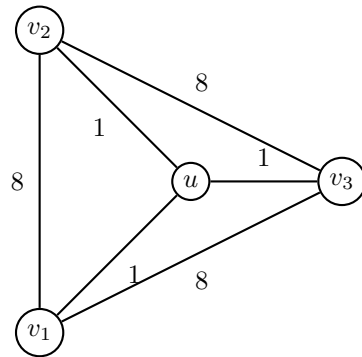
To show that the converse is not true, we have to find a graph G which has a unique MST, but for at least one cut (S, \bar{S}) , there are two or more light edges. The simplest example is a graph with 3 nodes u, v, w , and two edges $(u, v), (v, w)$, each of cost 1. This graph is a tree, so trivially has only itself as a spanning tree, and in particular a unique spanning tree. On the other hand, the cut $(\{v\}, \{u, w\})$ has two edges of cost 1 crossing, which are both light.

Exercise 23.1-11

Let's call the edge whose cost we decreased e , and its new cost is \hat{c} . The only possible change resulting from this decrease is that e may now belong to the new MST, and instead another edge should be thrown out. To test whether this is the case, we can use the cycle property we proved in class. Let's say that u, v are the endpoints of e . Let T be the MST of G before the cost decrease. Then, we run BFS in time $O(n)$ to find the unique path connecting u and v on T ; call it P . Once we have found the path, we can check in time $O(n)$ whether any edge on P has cost more than \hat{c} . If so, we replace the most expensive edge of P with e to obtain the new MST. Otherwise, we know that even after the cost decrease, e should not be part of the MST by the cycle property (because it is most expensive on at least one cycle, namely $P \cup \{e\}$).

Exercise 23.2-8

This algorithm fails. Take the following simple graph:



Never mind how the algorithm first divides the graph, the MST in one subgraph (the one containing u) will be one edge of cost 1, and in the other subgraph, it will be one edge of cost 8. Adding another edge of cost 1 will give a tree of cost 10. But clearly, it would be better to connect everyone to u via edges of cost 1.

Problem 23-3

- (a) The simplest proof is to look at Kruskal's algorithm. Consider the time right before Kruskal's algorithm adds its last (and most expensive) edge e . At that point, there are exactly two components S_1, S_2 . And because these two components are still not connected, no edge cheaper than e went between them (otherwise, it would have been added already). Thus, any spanning tree must contain an edge at least as expensive as e . But then, the minimum bottleneck tree has bottleneck value no lower than c_e , and the MST (found by Kruskal's algorithm) has bottleneck value c_e , so it is a minimum bottleneck tree.
- (b) The algorithm is really quite simple. Given b , go through all edges (in time $\Theta(m)$) and throw out all edges e of cost $c_e > b$, since they should never be used. Call the resulting graph G' . Now run a simple BFS in time $\Theta(m+n)$ to test if G' is connected. If it is, then any spanning tree of G' has a bottleneck of at most b , proving that the minimum bottleneck tree has bottleneck at most b (it could be better). Otherwise (i.e., if G' is not connected any more), then it is impossible to connect the graph using edges only of cost at most b . In particular, therefore, the minimum bottleneck tree must have bottleneck more than b .

Exercise 24.3-4

Here is one way. Run Dijkstra's algorithm, with two modifications: instead of taking sums of edge costs, take products of reliabilities. And instead of choosing the node with the *smallest* value, choose the biggest (using a heap, again). Now, a virtually identical induction proof shows that this algorithm works (though one would have to redo the work).

An even simpler way is to reduce it to the shortest path problem. Here is how: if we take a path P , then the reliability of that path is $\prod_{e \in P} r(e)$. We can rewrite that as $\exp(\sum_{e \in P} \ln(r(e)))$. So, because $\exp(\cdot)$ is a monotone increasing function, maximizing the reliability is the same as maximizing $\sum_{e \in P} \ln(r(e))$, which is the same as minimizing $\sum_{e \in P} (-\ln(r(e)))$. But that can be rewritten as $\sum_{e \in P} \ln(\frac{1}{r(e)})$. In other words, the highest reliability path is the shortest path with respect to the edge costs $c_e := \ln(\frac{1}{r(e)})$. And notice that all these edge costs are actually non-negative, because $r(e) \leq 1$. Hence, our algorithm is: Compute these c_e for all edges (in time $O(m)$), and then run Dijkstra's Algorithm.