The Minimum Spanning Tree Problem
(plagiarized from Kleinberg and Tardos, *Algorithm design*, pp 142–149)

Recall that a minimum spanning tree (V,T) of a graph $G = (V,E)$ with weighted links is a spanning tree with minimum total weight. This solves, for example, the problem of constructing the lowest cost network connecting a set of sites, where the weight on the link represents the cost. Such a tree can be found many greedy algorithms, including these three:

1) **Kruskal's algorithm:** build a spanning tree by successively inserting edges in order of increasing cost, as long as each new added edge does not create a cycle.

2) **Prim's algorithm:** start at a root node and grow a spanning tree by attaching successive least cost edges directly to the partial tree being created.

3) **"Reverse Delete" algorithm:** start with the full graph (V,E) and delete edges in order of decreasing cost, as long as doing so does not disconnect the graph.

[Note that Kruskal's algorithm was slightly misstated in the previous notes. For *Kruskal*'s algorithm, step 1 should have read "If $T \cup e$ is a *forest*, add e to T", permitting disconnected subgraphs at intermediate stages of the algorithm. The step as stated: "If $T \cup e$ is a *tree*, add e to T" results instead in Prim's algorithm, in which successive edges are always added to a growing tree. Either algorithm turns out to give a minimum spanning tree.]

To demonstrate that algorithms (1,2) produce minimum spanning trees, it is useful to characterize when it is safe to include an edge in the minimum spanning tree, as follows:

Cut property: Assume the edge costs are all distinct. Let S be any subset of nodes that is neither empty nor all of V, and let edge $e = (v,w)$ be the minimum cost edge with one end in S and the other in $V - S$. Then every minimum spanning tree contains the edge e.

Proof: A spanning tree that does not contain the edge e has some other path P from v to w. Let v' be the last node on that path in S, let w' be the first node on that path in $V-S$, and let $e' = (v', w')$ be the edge joining them. Exchanging e' for e reduces the total cost since by assumption edge e has the lowest cost of edges between S and $V - S$. But this exchange still leaves a spanning tree: the result is connected since any path that formerly used e' can now use e by rerouting from v' to v, then to w via edge e, then from w to w'. The exchange also creates no cycles since the only cycle created by adding e to the original tree is the above path P together with e, and that cycle is eliminated by removing e'.

To use the above property to show that Kruskal's algorithm (1) produces a minimum spanning tree (V,T) of G, first note that the algorithm produces a spanning tree: it contains no cycles by construction, and must connect all the vertices of G since if (V,T) is not connected, then there would be some nonempty set $S \subset V$ such that no edge from S to $V - S$ is in T. But since G is connected, the algorithm would add the first such edge encountered. To show that the spanning tree is minimal, consider an edge $e = (v,w)$ added by Kruskal's algorithm, and let S be the set of all nodes to which v is connected just before e is added (which could be just v). Then $v \in S$, but $w \notin S$, since adding e doesn't create a cycle. No edge from S to $V-S$ has already been encountered, since adding it would not have created a cycle, and hence it would have already been added by the algorithm. Thus e is the least expensive such edge, and so according to the cut property belongs to every minimum spanning tree. Hence Kruskal's algorithm produces a minimum spanning tree, because it adds only edges that must be in every minimum spanning tree.

To use the cut property to show that Prim's algorithm (2) also produces a minimum spanning tree is similar. As above, the algorithm produces a spanning tree because it contains no cycles by construction, and connects all the vertices of the original graph. To show that it produces a minimum spanning tree, note that in each iteration there's a set $S \subseteq V$ on which a partial spanning tree has been constructed. The next edge e added is the least expensive between S and $V-S$, and so by the cut property must be in every minimum spanning tree. Hence Prim's algorithm produces a minimum spanning tree, because it adds only edges that must be in every minimum spanning tree.

To demonstrate that the third algorithm above as well produces a minimum spanning tree, it is useful to characterize when an edge is guaranteed *not* to be in any minimum spanning tree, as follows:

Cycle Property: Assume the edge costs are all distinct. Let C be any cycle in G, and let the edge $e=(v,w)$ be the most expensive edge in C. Then e does not belong to any minimum spanning tree of G.

Proof: Let T be a spanning tree that contains e. Deleting e partitions the vertices into two components: a part S containing v, and $V-S$ containing w. The edges of C other than e form a path P with one end at v and the other at w, so there is some edge e' on P that crosses from S to $V-S$. Adding the edge e' gives a graph (V,T') that is connected and has no cycles, so is a spanning tree of G, and is less expensive than T.

To use the above property to show that the "Reverse Delete" algorithm (3) produces a minimum spanning tree, first note that the the algorithm produces a spanning tree (V,T): it's connected by construction, and contains no cycles because the most expensive edge in any cycle is deleted, since that does not disconnect the graph (but *does* eliminate the cycle). To show that the spanning tree is minimum, consider any edge $e=(v,w)$ removed by the algorithm. Just before it's removed, it must lie on some cycle C (otherwise removing it would disconnect the graph). As the first encountered on that cycle, it must be the most expensive on it, so by the cycle property doesn't belong to any minimum spanning tree. Hence the "Reverse Delete" algorithm produces a minimum spanning tree, because it removes only edges that cannot be in any minimum spanning tree.

The assumption of distinct edge costs in the cut/cycle property proofs is easily lifted by permitting perturbations of edge costs, small enough to serve only as "tie-breakers" of formerly equal cost edges. As a final overall observation, note that the combination of the cut and cycle properties implies that any algorithm that builds a spanning tree by repeatedly including edges when justified by the cut property and deleting edges when justified by the cycle property — in any order at all — will produce a mininum spanning tree. This extra flexibility can provide better optimized algorithms for some purposes.

INFO 295, 4 Oct 07