Kruskal's MST Algorithm

In Wednesday's class we looked at the Disjoint Set data structure that splits a bunch of data values into sets in such a way that each datum is in exactly one set. There are 2 primary methods for this structure: Find(x), which gives the set containing x, and Union(x, y), which merges the sets containing x and y into one set.

Clicker Q: What does the Find(x) method actually return?

A. A string with the name of the set containing x.
B. An array list of the elements of the set containing x.
C. A HashSet of the elements of the set containing x.
D. One element of the set containing x, called the "root" of the set.

Clicker Q: How does Union(x, y) join the sets containing x and y?

A. It finds all of the elemIents of the set containing x and adds them to the set containing y.
B. It makes x point at y.
C. It makes the root of the set containing x point at y.
D. It makes the root of the smaller of the two sets point a the root of the larger of the two sets.

Last Clicker Q: How long do Find(x) and Union(x,y) take if our sets contain a total of n objects?

A. Find: O(n) Union: O(n*log(n))
B. Find: O(n) Union: O(n)
C. Find: O(log(n)) Union: O(n)
D. Find: O(log(n)) Union: O(n)

Joseph Kruskal, a mathematician/computer scientist/statistician at Bell Labs, developed another minimum spanning tree algorithm that uses our Disjoint Set data structure. Here is his algorithm:

Kruskal's Algorithm: Order the edges of the graph from smallest to largest. Call MakeSets() on the nodes of the graph, consider each of them to be a singleton set. Now take the edges in order from smallest to largest. Let u and v be the nodes connected by an edge. If Find(u) == Find(v), which means that u and v are already connected, then discard the edge. If Find(u) != Find(v) include this edge in our spanning tree and merge the sets containing u and v.

Kruskal's algorithm will certainly form a graph with no cycles; when it gets n-1 edges it will connect all of the nodes of the graph and so will be a spanning tree. How do we know it is a minimum spanning tree?

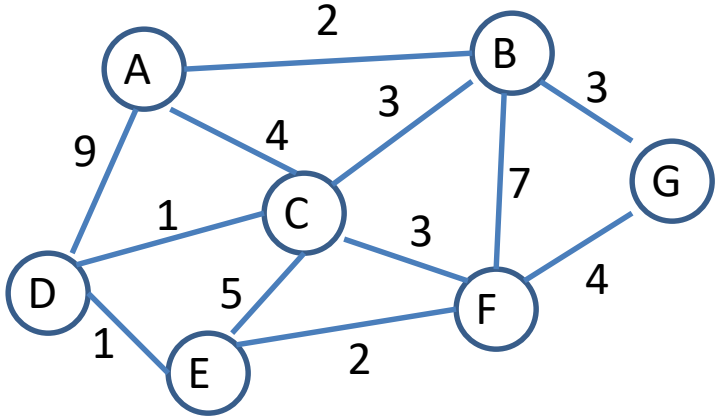
Recall the theorem we proved for Prim's algorithm:

Theorem: Let S be any set of nodes in an undirected weighted graph, all of whose edge weights are distinct. Let e, the edge from node v to node w, be the cheapest edge that connects some node in S to a node not in S. Then edge e must be included in any minimum spanning tree for the graph.

Let S be the set of nodes connected to v at the time we choose this edge. w cannot be in S, or else this edge would form a cycle. So e is an edge connecting S to the complement of S. If there was a cheaper edge connecting S to its complement, we would have considered it previously (since we are taking the edges from smallest to largest) and we would have used it since it doesn't form a cycle. This means that edge e is the cheapest edge connecting S to its complement; according to the theorem any minimum cost spanning tree must include e. Even if the edge weights aren't all distinct, no cheaper tree can exist that does not include e.

Thus, we see that the edges chosen by Kruskal must form a minimum cost spanning tree.

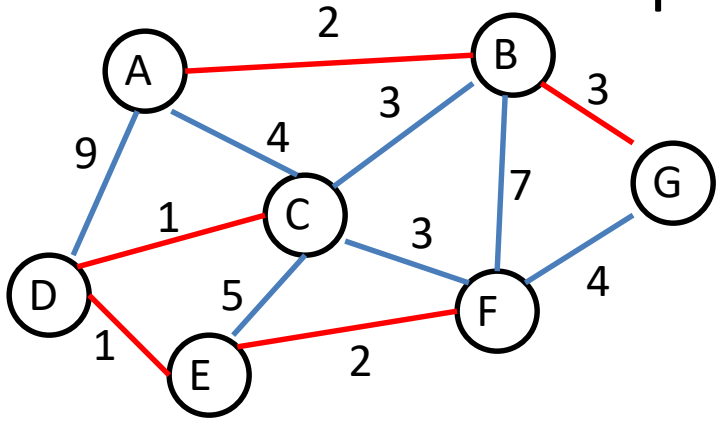
Kruskal's algorithm considers the edges in increasing order and uses those that connect nodes that are in different connected components. For example:



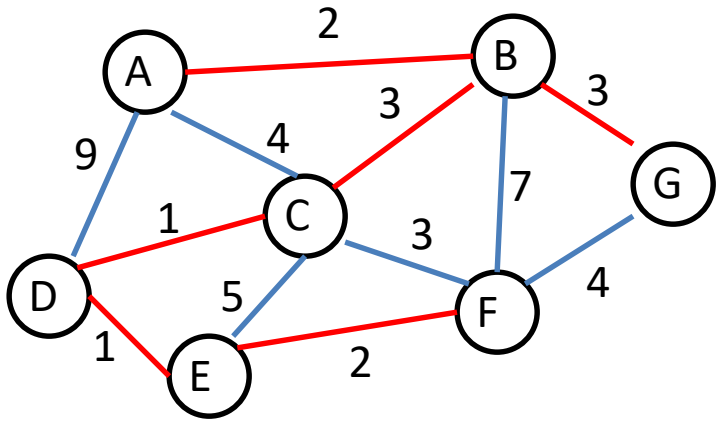
We consider the edges in the following order:

1. D-C
2. D-E
3. A-B
4. F-E
5. B-G
6. C-F
7. C-B
8. A-C
9. F-G
10. E-C
11. B-F
12. A-D

The first 5 edges we consider cause no problems:



So far we have selected 5 edges to connect our 7 nodes. The nodes fall into 2 connected components: {C,D,E,F} and {A,B,G}. We need one more edge. The next edge in order is C-F but C and F are in the same component. The next edge is C-B and those vertices are in different components, so that completes our spanning tree.



Kruskal produces the same minimum spanning tree as Prim, which is no surprise. Note that we only considered about half of all of the edges when we were choosing edges for the spanning tree

What is our time analysis? We could form a priority queue with the edges and their weights; it takes time O(|E|) to make the queue and time O (log(|E|)) to poll it. We would make disjoint sets containing the nodes of the graph; this takes time O(|V|) to make the singleton sets and time O(log(|V|) each time we do a find. Each step does a poll of the priority queue, 2 find operations, and possibly a union, making it O(log(|V|) + log(|E|)) = O(log(|V|) (remember that |E| <= $|V|^2$, so O(log(|E|) = O(log(|V|)). If we are really unlucky we need to do |E| steps, so the entire analysis is O(|E|*log(|V|)). This is the same running time as Prim's algorithm.

Kruskal's algorithm has the advantage that it doesn't change priorities of the edges once they are in the priority queue.