600.363 Introduction to Algorithms / 600.463 Algorithms I **Lecturer:** Michael Dinitz
Topic: Minimum Spanning Trees **Date:** 10/21/14

14.1 Introduction

Today we're talking about Minimum Spanning Trees (MSTs). For this problem we are given a graph $G = (V, E)$ and edge weight $w : E \to \mathbb{R}_{\geq 0}$ (we will assume that edge weights are positive for convenience, but it doesn't actually matter – everything we'll do works fine with negative edge weights). The goal is to find a tree T which is a spanning subgraph of G, i.e. every node is incident to at least one edge of T, and has minimum total weight $\sum_{e \in T} w(e)$.

This is a classic, fundamental, and important problem in graph algorithms. It has a number of obvious applications, e.g. building a network of minimum cost that allows every pair of nodes to communicate. But it is also a surprisingly useful primitive, which comes up in a variety of somewhat surprising places. As we'll see, it's also one of the simplest examples of some standard proof techniques and algorithmic techniques. In particular, it is one of the most famous places where *greedy algorithms* work well.

Today we'll discuss the two most famous MST algorithms, as well as some extensions. Both of them are greedy algorithms, but they work in slightly different ways.

14.2 Prim's Algorithm

The first algorithm we'll talk about is *Prim's Algorithm.* In Prim's algorithm we start with a node and grow an MST out of it. Because of this approach, the algorithm actually looks a lot like Dijkstra's shortest-path algorithm, but instead of computing a shortest-path tree it computes an MST.

We start with an arbitrary node u and include it in T. We then find the lowest-weight edge incident on u, and add this to T. We then repeat, always adding the minimum-weight edge that has exactly one endpoint in T to T. Slightly more formally, Prim's algorithm is the following:

1. Pick some arbitrary start node u. Initialize $T = \{u\}$.
2. Repeatedly the lowest-weight edge incident to T (the lowest-weight edge that has exactly one vertex in T and one vertex not in T) until T spans all the nodes.

As always, we'll want to do two things: prove correctness and running time. The running time is pretty obvious – it's just the same as Dijkstra. That is, we can maintain a heap of all of the edges with at least one endpoint in T, and in each iteration we do Extract-Mins until we see an edge that has one endpoint in T and one endpoint not in T. So this takes time $O(m \log n)$ using a binary heap, or $O(m + n \log n)$ using a Fibonacci heap. It's actually even faster (though not asymptotically) than Dijkstra, since we never need to do any Decrease-Key operations.

Correctness is more interesting. To prove it, let's first note a simple property about spanning trees. If we take a spanning tree T (not necessarily an MST), and add any edge not in T to it, this creates exactly one cycle. Now we can remove any edge from this cycle, and we're left with a new spanning tree (since removing an edge from a cycle does not kill connectivity but does kill the cycle).

Theorem 14.2.1 *Prim's algorithm correctly computes an MST.*

Proof: We'll prove this by induction. The induction hypothesis will be that after each iteration, the tree T is a subgraph of some minimum spanning tree M. This is trivially true at the start, since initially T is just a single node and no edges. Now suppose that at some point in the algorithm we have T which a subgraph of M, and Prim's algorithm tells us to add the edge e. So we need to prove that $T \cup \{e\}$ is also a subtree of some MST. If $e \in M$ then this clearly true, since by induction T is a subtree of M and $e \in$M and thus $T \cup \{e\}$ is a subtree of M.

So suppose that $e \notin M$. Consider what happens when we add e to M. This creates a cycle. Since e has one endpoint in T and one endpoint not in T (since Prim's algorithm is adding it), there has to be some other edge e' in this cycle that has exactly one endpoint in T. So Prim's algorithm could have added e' but instead chose to add e, which means that $w(e) \leq w(e')$. So if we add e to M and remove e' we are left with a new tree M' whose total weight is at most the weight of M, and which contains $T \cup \{e\}$. This maintains the induction, so proves the theorem. ■

One interesting thing to note about the above proof is that in fact it must be the case that $w(e) = w(e')$, since if $w(e) < w(e')$ then M' would have weight lass than M, contradicting the assumption that M is an MST. But that fact wasn't necessary for the proof to go through.

14.3 Kruskal's Algorithm

The next algorithm we'll consider is a bit different, despite also being a greedy algorithm. Prim's algorithm maintains a graph which is always a tree, and stops when it is spanning. Kruskal's algorithm does not maintain a tree, but instead maintains a forest (a collection of trees) that is always spanning, and stops when the forest becomes a single tree (note though that some "trees" in the forest might just be single nodes). So it is in some ways the opposite of Prim's algorithm.

Kruskal's algorithm is the following:

1. Sort the edges by their weights.
2. Initialize $F = \emptyset$
3. Consider the edges from lowest to highest weight. For each edge e, add it to F if it does not create any cycles in F.

Let's start by proving correctness.

Theorem 14.3.1 *Kruskal's algorithm correctly computes an MST.*

Proof: The argument is actually quite similar to the one we used fro Prim's algorithm: we will prove by induction that F is always a subgraph of some MST. This is obviously true at the beginning, since F is empty. So suppose it is true at some point in our algorithm: F is currently a subgraph of some MST M, and Kruskal's algorithm adds an edge e. We want to show that there is some MST M' such that $F \cup \{e\}$ is a subgraph of M'.

If $e \in M$ then this is trivially true: we set $M' = M$, and since $F \subseteq M$ and $e \in M$ we know that $F \cup \{e\} \subseteq M$. Otherwise, consider what happens when we ad e to M. Since Kruskal's algorithm added e, this means that adding e did not create any cycles, so the two endpoints of e must be in different trees in F. So if we follow this cycle, there must be some other edge e' whose two endpoints also lie in different trees of F (not necessarily the same two trees as e). This means that Kruskal's algorithm could have added e', but instead chose to add e. Thus $w(e) \leq w(e')$. So if we remove e' from M and add e to get a new tree M', we know that the weight of M' is at most the weight of M and is thus an MST, and $F \cup \{e\}$ is a subgraph of M'. This maintains the induction, and proves the theorem. ∎

What about the running time? Sorting takes $O(n \log n)$. Then m times, we have to check whether inserting an edge would cause a cycle to be created. In other words, we have to check whether the two endpoints of an edge are in the same connected component. Instead of a heap like we used for Prim, the important data structure here is Union-Find! We'll have a set for each tree in F, when we add an edge we'll need to do a Union, and when we test whether to add an edge we'll have to do two Finds. So we need to do n initial Make-Sets, $2m$ Finds, and $n-1$ Unions. If we use the simple list-based data structure, this gives running time of $O(m + n \log n)$. If we use the fancier tree-based data structures with path compression and union by rank, we get running time of $O(m \cdot \alpha(n))$, where $\alpha(n)$ is the inverse-Ackermann function applied to n. This is an improvement if m is extremely sparse, but for large m the simple list-based data structure actually performs better since there are so many more Finds.

14.4 Some Unifying Structure

Despite being somewhat "opposite" algorithms, it is striking that the proofs of correctness of Prim's algorithm and Kruskal's algorithm are so similar. There are actually a few reasons for this, but let's highlight one of them.

We'll talk far more about them later in the class, but a *cut* in a graph is simply a partition of the vertices into two parts. That is, given a graph $G = (V, E)$, every subset $S \subseteq V$ defines a cut $(S, \bar{S})$. Equivalently, we can think of the edge set that does between S and $\bar{S}$. Cuts are very related to many other concepts in graph theory and graph algorithms, particularly flows (which we'll talk a lot about later), but they also turn out to be related to MSTs in an interesting way.

Consider an arbitrary cut S, and let T be an arbitrary MST. Can we say anything about the relationship between T and S? To start, the fact that T is spanning tells us something: there is at least one edge from T that goes across the cut. This is easy to see, since if there are no edges from T across the cut the T must have all edges in one side or the other (since it cannot be disconnected), and thus does not span the graph.

But what about the fact that T is a *minimum* spanning tree? This actually gives us more information, and is the basic idea that we used in the analysis of both Prim's and Kruskal's algorithm.

Theorem 14.4.1 *Let $(S, \bar{S})$ be an arbitrary cut, and let e be an edge across the cut (one endpoint in S, the other in $\bar{S}$) that has the smallest weight of any edge across the cut. Then there is an MST T which contains e.*

Proof: The proof works just like our analysis of the algorithms. Let T be an arbitrary MST. If $e \in T$ then we are done. Otherwise, consider adding e to T. This causes a cycle, and there must be at least one other edge e' in this cycle which crosses the cut. By definition $w(e) \leq w(e')$, so we can remove e' and add e to get an MST which contains e. ■

Moreover, it is easy to see that any MST must contain some minimum-weight edge across the cut.

Theorem 14.4.2 *Let $(S, \bar{S})$ be an arbitrary cut, and let E' be the set of edges across the cut of minimum weight (so $w(e) = w(e')$ for an two edges $e, e' \in E'$ and $w(e) < w(e')$ for any $e \in E'$ and $e' \notin E'$). Let T be an arbitrary MST. Then T contains some edge in E'.*

Proof: Suppose that T does not intersect E'. Then pick an arbitrary $e \in E'$, and consider what happens when we add it to T. Some cycle is created, and there must be a different edge e' in the cycle across the cut, which by assumption is not in E'. Thus $w(e) < w(e')$, so if we remove e' from T and add e' we get a new spanning tree T' with $w(T') < w(T)$. This contradicts our assumption that T is an MST. ■