CSE 331: Introduction to Algorithm Analysis and Design
Greedy Algorithms

1 Minimum Spanning Tree

1.1 Problem

Input: A weighted undirected connected graph $G = (V, E)$ where each edge $e \in E$ has weight $l_e > 0$.

Output: A minimum spanning tree (MST) of G. Specifically, the MST is defined by (V, T) where $T \subseteq E$, all nodes are connected by the edges in T, and the sum of the edge weights in T is minimum ($\sum_{e \in T} l_e$ in minimized).

1.2 Prim's Algorithm

1. Pick a starting node s
2. Initialize $S = \{s\}$
3. Iteratively add the node with the least cost crossing edge $e = (u, v)$ to S such that $u \in S$ and $v \notin S$ with minimum l_e.

1.3 Kruskals's Algorithm

1. Sort the edges in E by increasing weight.
2. Initialize $T = \emptyset$
3. Iteratively add the the edges in E to T unless adding it would create a cycle.

1.4 Cut Property Lemma

Lemma 1. *Given a connected undirected graph $G = (V, E)$ with distinct edge weights and a cut $S \subset V$ such that $S \neq \emptyset$ and $V \setminus S \neq \emptyset$, then the crossing edge $e = (u, v)$ with $u \in S$ and $v \in V \setminus S$ with the smallest weight is in every MST of G.*

Proof. We will prove the cut property lemma by contradiction. For the contraction, we will assume that we have an MST of G defined by the edges T and for some cut S such that $S \neq \emptyset$ and $V \setminus S \neq \emptyset$, that the minimum weight crossing edge $e = (u, v)$ with $u \in S$ and $v \in V \setminus S$ is not in T ($e \notin T$). Since the MST is connected by T, we know that there must be at least one crossing edge between S and $V \setminus S$ in Tthat connects these disjoint sets of nodes[1].

[1] This proof will differ slightly from the one in class in that we will not label this crossing edge so we can avoid using a double prime later. Suffice to know that such an edge exists.

We will then add the minimum crossing edge e to the set T. Since T defined a tree, we know that it is connected, has $n-1$ edges, and no cycles. By adding e to this set, we must have introduced a cycle that contains e. If we follow this cycle starting with e moving from u to v, it must end back at u. This implies that at some edge e', the cycle crosses S and $V \setminus S$ again. We then remove e' from T to remove the cycle. Since we removed one edge in a cycle, the graph will still be connected since any path connecting two nodes containing e' can instead go around the remaining edges in the cycle instead.

We'll call the resulting set of edges after this swap T' and argue that it has less weight than T. We can see that $w(T') = w(T) + w(e) - w(e')$ where we overload $w()$ to mean either the weight of an edge or the sum of weights for a set of edges. Since we know that $w_e < w_{e'}$, we have $w(e) - w(e') < 0$ so it follows that $w(T') < w(T)$. Since have found a tree defined by T' that has less total weight than T, we have a contradiction of the fact that T defined a minimum spanning tree. Therefore, our assumption that there exists an MST that does not contain e is false since we can always decrease it's weight by swapping e for some e' with larger weight. Thus, the minimum crossing edge must be in all MST's.

□

To utilize this lemma in the following proofs of correctness, we will use a trick to address property of distinct edge weights which is required in the cut property lemma. Specifically, if there are edges in E with identical weights there can be multiple MST's for Gthat an algorithm can output[2]. This is a problem since the proofs depend on the argument that there are edges that must be in all MST's. If there is an arbitrary choice between two edges, either can be in an MST, but neither must be in all MST's.

The trick we use to address this is to adjust the edge weights in E to artificially make the weights distinct, but not by enough to change the order of edge weights in the graph. This effectively removes the ambiguous decisions that an algorithm would have to make to the point that there is only one MST for G and we will prove that the algorithm returns this exact MST. This is an artificial construct that we use to address the unique edge weights constrain in the correctness proofs and is not used in the algorithms themselves.

1.5 Correctness of Prim's Algorithm

Theorem 1. *Prim's algorithm run on a connected undirected graph $G = (V, E)$ will always output a Minimum Spanning Tree (MST) of G.*

Proof. To prove the correctness of Prim's algorithm, we will utilize the cut property lemma. To do this, we must make our argument fit into the conditions of the lemma. We first use the trick described above to address the condition of unique edge weights. We now define the set S from the lemma to be the conveniently labeled S from the algorithm. Since we initialize $S = \{s\}$ and never remove any element from S, we have $S \neq \emptyset$. The algorithm runs until $S = V$, which implies $V \setminus S \neq \emptyset$ whenever an edge is being added to T. This satisfies all the condition of the lemma.

By the definition of the algorithm, it is always adding the smallest weight crossing edge between S and $V \setminus S$. By the cut property lemma, we know that this edge must be in all MST's. Since the algorithm is only adding edges that must be in every MST, it can't make a mistake by adding an

[2]If there exists a cycle in G with two or more edges having identical weight, the choice of which of these edges to add to the MST is arbitrary

edge that doesn't lead to minimum weight. Since the algorithm explores all edges starting at s, it will explore the connected component of s. The graph is connected, so it will explore the entire graph while only adding edges that must be in every MST. The result must be an MST of G. $\square$

1.6 Correctness of Kruskal's Algorithm

Theorem 2. *Kruskal's algorithm run on a connected undirected graph $G = (V, E)$ will always output a Minimum Spanning Tree (MST) of G.*

Proof. This proof is similar to the proof of Prim's algorithm in the sense that we will set up the proof to utilize the cut property lemma for the crucial part of the proof. Specifically, when an edge $e = (u, v)$ is added to T, we will show that it fits the conditions for the cut property and therefore must be in any MST. We will define S to be the current connected component of u in T at this point of the algorithm. Since $u \in S$, we have $S \neq \emptyset$. We know that the algorithm is adding e to T, so it doesn't create a cycle. This implies that $v \in V \setminus S$ since if $v \in S$ it would create a cycle because S is connected. Thus we have $V \setminus S \neq \emptyset$.

By contradiction, e must be the minimum crossing edge. If e were not the minimum crossing edge, then the minimum crossing edge would have already been considered by the algorithm. Since S and $V \setminus S$ are not connected, this crossing edge would not create a cycle and it would have been added to T which is a contradiction of the sets being disconnected. Therefore, e is the minimum crossing edge and must be in every MST. The algorithm only adds edges that must be in all MST's and considers all edges in the graph adding the as long as they don't create a cycle. The result must be a minimum spanning tree.

$\square$

1.7 Runtime

Both Prim's and Kruskal's algorithms can be implemented in $O(m \cdot \log n)$ time by using efficient data structures. Prim's can be implemented using a priority queue (not the one that comes with java) and Kruskal's can reach this bound by using a union-find data structure. The sorting step in Kruskal's takes $O(m \cdot \log m)$, but since m is $O(n^2)$ we have $\log m$ is $O(\log n^2) = O(2 \log n)$ which is $O(\log n)$