1 The Story So Far

- Last time, we started analyzing two MST algorithms – Prim’s and Kruskal’s.
- Each algorithm makes a certain greedy choice at each step of which edge to add to the edge set that will become the final spanning tree.
- It was fairly easy to show that this greedy choice preserves the feasibility of the final solution, i.e., that the final edge set $T$ is a spanning tree.
- Moreover, we proved that there always exists an MST that includes the greedily chosen edge.
- More generally, we proved that there exists an optimal solution to any problem instance that includes the greedy choice.
- We will refer to this property as the “greedy choice property” of the algorithm.
- We proved the greedy choice property for Prim and Kruskal by way of an exchange argument: given an optimal solution that does not make the greedy choice, we can modify the solution so that it does make the greedy choice without making its objective value worse.

We’re now going to show how proving the greedy choice property for our algorithms contributes to a proof of their optimality.

2 A Full Optimality Proof

- As with DP, we need to check three key properties to prove optimality.
- The first is the aforementioned greedy choice property, which shows that the algorithm’s first choice is consistent with optimality.
- (Compare this to the complete choice property for DP – in the greedy case, we are proving that our single choice is in fact a complete choice set all by itself!)
- Our second and third properties are pretty much identical to those defined for DP.
- We need the inductive structure property: after making the greedy choice, we are left with a smaller version of the same problem, such that any feasible solution to this subproblem may feasibly be combined with the greedy choice.
• We can see this for our MST algorithms by treating each subgraph connected by edges in $T$ as a supervertex.

• If a subgraph $H$ of $V$ is contracted into a supervertex $h$, then $\forall u \not\in H$, $c(h, u) = \min_{v \in H} c(v, u)$.

• In other words, a supervertex preserves the costs associated with connecting other vertices to its associated subgraph.

• In this view, after we make the greedy choice and contract the resulting connected subgraph into a supervertex, we clearly want to find a spanning tree on a strictly smaller graph.

• Moreover, If $V'$ is the graph obtained from $V$ after one contraction, and we find a spanning tree $T'$ on $V' \times V'$, then “undoing” the contraction results in a spanning tree for $V$.

• This proves that a feasible solution to $V'$ yields a feasible solution to $V$.

Two properties down, one to go.

• The third property to check for optimality is the optimal substructure property: if we optimally solve the subproblem $V'$ remaining after making the greedy choice $e$, resulting in a sub-solution $T'$, then $T = T' \cup \{e\}$ is an optimal solution for $V$.

• As for DP, it suffices to observe that the cost of a solution is separable into a sum of terms, one depending only on the sub-solution $T'$, and one depending only on the greedy choice $e$.

• In this case, the cost of the final MST $T$ is given by

$$\sum_{(u,v) \in T} c(u, v) = c(e) + \sum_{(u,v) \in T'} c(u, v),$$

and so the cost is indeed separable.

• As for DP, we need to prove inductive structure and optimal substructure for each possible first choice – but this is easy, because greedy makes just one first choice!

As for DP, we can show that these three properties are enough to prove optimality, using a standard inductive argument template.
• **Thm:** Prim’s and Kruskal’s algorithms return an MST.

• **Pf:** by induction on the size of problem $V$.

• **Bas:** if $|V| \leq 2$ (i.e. $V$ has at most one edge), then the algorithm returns the single edge in $V$ if it exists, or $\emptyset$ otherwise. In all cases, this is clearly an MST.

• (Note: you should always justify the correctness of your algorithm’s base case even if you are not writing the full inductive proof!)

• **Ind:** suppose that the algorithm returns an MST for graphs with $< |V|$ vertices.

• Given graph $V$, the greedy choice $e$ induces a subproblem $V'$ after contracting the new connected component to a supervertex.

• By the IH, our algorithm computes an MST $T'$ on $V'$.

• Moreover, by inductive structure, $T = T' \cup \{e\}$ is a valid spanning tree for $V$.

• Suppose that $T$ is not a minimum spanning tree, and let $T'$ be an optimal solution (i.e., an MST) for $V'$. By our greedy choice property, there exists an MST $T^*$ for $V$ that makes the greedy choice $e$.

• We then have

\[
\begin{align*}
\text{cost}(T) & > \text{cost}(T^*) \\
\text{cost}(T') + c(e) & > \text{cost}(T^* - \{e\}) + c(e) \\
\text{cost}(T') & > \text{cost}(T^* - \{e\}).
\end{align*}
\]

• But the last inequality is impossible because $T'$ and $T^* - \{e\}$ are spanning trees for the same (contracted) graph $V'$, and $T'$ is optimal. $\leftrightarrow$

• Conclude that $T$ must be an MST for $V$, as claimed. QED

As for DP, if you can prove the three key correctness properties and validate directly that your algorithm works in the base case, you don’t need to reproduce the entire inductive argument. The hard part is usually proving the greedy choice property, which can be done via an exchange argument.

### 3 Efficient Implementation

• Because greedy algorithms, unlike DP, don’t require branching on each choice, they are generally quite easy to implement in polynomial time.

• The usual question is, how low a polynomial can we achieve?

Let’s focus on Prim’s algorithm.

• At each step of Prim, we need to find a lowest-cost edge connecting some vertex in $T$ to some vertex not in $T$. 

3
• If we maintain a list $L$ of all edges connecting $T$ to not-$T$, then at each step of the algorithm, we
  
  1. Select a lowest-cost edge $e$ from $L$.
  2. Remove $e$ from $L$ and add it to $T$.
  3. Add to $L$ any edges from the “new” endpoint of $e$ to the rest of the graph.

• The list $L$ can be as long as $|E|$, the number of edges in the graph.
• We select/remove edges from $L$ up to $|V|$ times during the algorithm.
• We add each edge of the graph to $L$ at most once.
• If we maintain $L$ as an unordered linked list...
  
  1. Selection of a lowest-cost edge $e$ takes time $\Theta(|E|)$;
  2. Removal of $e$ takes time $O(1)$;
  3. Addition of each new edge takes time $O(1)$.
• Hence, the worst-case cost of the algorithm is at most
  
  $|V| \times \Theta(|E|) + |E| \times O(1) = \Theta(|V||E|)$.

• If instead we maintain $L$ as a min-first priority queue by edge weight...
  
  1. Selection of a lowest-cost edge $e$ takes time $\Theta(\log |E|)$;
  2. Removal of $e$ takes time $\Theta(\log |E|)$;
  3. Addition of each new edge takes time $\Theta(\log |E|)$.
• Hence, the cost of the algorithm is at most
  
  $|V| \times \Theta(\log |E|) + |E| \times \Theta(\log |E|) = \Theta(|E| \log |E|)$,

  which is quite a bit faster.

We can actually implement Kruskal’s algorithm just as fast as Prim’s, but it requires a different, slightly fancier data structure that we will study later.