Why Minimum Spanning Trees (MST)?

• Example 1
  - A town has a set of houses and roads where a road connects two houses and there is a repair cost \(w(u, v)\) based on each edge. Repair only enough roads so everyone is connected at minimum cost.

• Example 2
  - In circuit design a set of \(n\) pins are to be connected with \(n-1\) wires of least cost.

• Example 3
  - Given a set of sensor nodes that need to be connected to an upstream node and ultimately a sink node, what is the least cost tree where the cost is the power required to transmit from a node to the upstream node.

• Example 4
  - Travelling Salesman Problem (TSP) is hard. Can we use MST to develop an approximation algorithm.
What is a minimum spanning tree?

- **Undirected graph** $G = (V, E)$
  - Weight (or cost) $w(u, v)$ on each edge $(u, v) \in E$
- **Minimum spanning tree**: a tree $T \subseteq E$ that connects all vertices and has minimum weight.
  - $T$ is called a spanning tree if it connects all vertices
  - minimize $w(T) = \sum_{(u,v) \in T} w(u, v)$,
    over all spanning trees
An example MST

- Properties of an MST
  - It has \(|V| - 1\) edges
  - It has no cycles
  - It might not be unique

- For the above tree (shaded)
  - The cost is 37

- Question
  - Does breadth first or depth first search give us an MST?
Building a solution using a generic MST algorithm

Generic-MST(G, w)

\[ A = \emptyset \] //A is set of edges building up to an MST

While A is not a spanning tree

1. find an edge (u, v) that is safe for A
2. \[ A = A \cup \{(u,v)\} \]

return A

In this generic algorithm we build a solution by:

- building a set A of edges starting from the empty set
- As we add an edge, we want to maintain the loop invariant:
  - A is a subset of some MST
- Add only edges that maintain this invariant. An edge is safe for A if when added to A the loop invariant is maintained. Such an edge is call a safe edge
- How do we find a safe edge?
Cuts and edges that cross them

- Given $G = (V, E)$, let $A \subseteq E$ and $S \subset V$
- A cut $(S, V - S)$ is a partition of vertices in two disjoint sets $S$ and $V - S$
- Edge $(u, v) \in E$ crosses cut $(S, V - S)$ if one endpoint is in $S$ and the other is in $V - S$
- A cut respects $A$ if and only if no edge in $A$ crosses the cut
- An edge is a light edge crossing a cut if and only if its weight is minimum over all edges crossing the cut. A light edge may not be unique.
Example

(a)

(b)
The main theorem

Theorem
Let $A$ be a subset of some MST, $(S, V - S)$ be a cut that respects $A$ and $(u, v)$ be a light edge that crosses $(S, V - S)$. Then $(u, v)$ is safe for $A$.

Proof
Let $T$ be the MST such that $A = T$. Clearly there exists a simple path $p$ in $T$ from $u$ to $v$. And since $u$ and $v$ are on opposite sides of the cut, there must be an $(x, y)$ that is an edge in $p$ and that crosses the cut. Thus, $(x, y)$ is either $(u, v)$ in case we are done.
Else, we can replace $(x, y)$ by $(u, v)$ and get a new MST $T' = T - \{(x, y)\} \cup \{u, v\}$, that is equal or lesser weight than $T$, since $(u, v)$ is a light edge.
This is a contradiction if it is of lesser weight thus both $(x, y)$ and $(u, v)$ must have been light edges, neither of which are in $A$ since $A$ respects the cut.
So $(u, v)$ is also safe for $A$: since $T'$ is an MST, $A = T'$ and $A \cup \{(u, v)\}$ is in $T'$. 
Why the light edge is safe for $A$

Solid edges shown are in $\text{mst}T$. Shaded edges are the path $p$. $A$ is some subset of $T$ (not shown).

Since the cut respects $A$, $(x, y)$ and $(u, v)$ are not in $A$. In fact none of the edges that cross the cut can be in $A$. Thus removing $(x, y)$ break $T$ into two components. Adding $(u, v)$ back in reconnects the two components.
Important Corollary

Let $C = (V_C, E_C)$ be a connected component in the forest $G_A = (V, A)$ where $A$ is a subset included in some minimal spanning tree of $E$. Let $(u, v)$ be a light edge connecting $C$ to some other component in $G_A$. That is, $(u, v)$ crosses the cut $(V_C, V - V_C)$.

Then $(u, v)$ is safe for $A$.

Proof
Set $S = V_C$ in the previous theorem.

Note that the cut respects $A$, since by definition of the forest $G_A$, we are considering connected components and thus no cross edge between two components can be in $A$. 

Adding a safe edge one edge at a time

• Note that the idea of a greedy algorithm is one that tries to modify / extend one best step at a time

• The above generic algorithm is a greedy algorithm. We have two classical (greedy) algorithms for minimum spanning trees

• Kruskal’s Algorithm
  - Kruskal’s adds edges starting with a forest of trees.
  - It sorts the edges by cost and tries to add edges in order of ascending (non-decreasing) order of cost
  - An edge can be added if it connects two different trees (sets) in the forest. An edge cannot be added if it is added to the same set or same tree. In this case a cycle would be formed. Only safe edges are added.

• Prim’s Algorithm
  - Prim’s starts with an arbitrary node that is the “growing tree” and there is always one tree that is extended.
  - A node is added that is the least cost node that connects to the current tree. Connecting to a tree means to connect to some node in the tree.
  - This results in an extended tree by adding the corresponding edge. Again, the next least cost node (and edge) is added to the extended tree, etc.
Applications of the Corollary

• Kruskal’s algorithm: The edge that is added is a light edge from one tree ($C_1$ component) to another tree ($C_2$ component) that is in the forest $G_A$.

• Prim’s algorithm: Note that the growing tree is always the set of edges $A$. Each step adds a light edge to an isolated vertex in the set $V - A$. 
Pseudo code for Kruskal

\[
\begin{align*}
\text{MST-Kruskal}(G, w) \\
A &= \emptyset \\
\text{for each vertex } v \in G.V: \\
\quad \text{Make-Set}(v) \\
\text{sort the edges of } G.E \text{ into ascending order by cost} \\
\text{for each edge } (u, v) \in G.E, \text{ considered in ascending order:} \\
\quad \text{if } \text{Find-Set}(u) \neq \text{Find-Set}(v) \quad \text{//vertices not in the same set} \\
\quad \quad A = A \cup \{(u, v)\} \quad \text{//it is ok to add the edge to } A \\
\quad \quad \text{Union}(u, v) \quad \text{//union the two trees} \\
\text{return } A
\end{align*}
\]
Kruskal’s Algorithm: an example
Complexity Analysis of Kruskal’s Algorithm

- Initial $A$: $O(1)$
- First for loop: $|V|$ Make-Set operations. Each node is a separate tree in the forest. No edges added to $A$ yet.
- Sort $E$: $O(E \ lg E)$ to sort the edges.
- Second for loop: $O(E)$ Find-Sets and Unions
  - Assume the implementation of disjoint-set Union Find with path compression and union by rank
  - Then we have, for this loop: $O((V + E) \alpha(V))$
- Total time is $O((V + E) \alpha(V)) + O(E \ lg E)$
  - since $G$ is connected, $|E| \geq |V| - 1$
  - $\alpha(V) = O(lg V) = O(lg E)$
- Total time is $O(E \ lg E)$
  - Can also be stated as $O(E \ lg V)$ since for graphs $O(lg E) = O(lg V)$
Pseudo code for Prim

MST-Prim(G, w, r)
for each u ∈ G.V
    u.key = ∞ //every vertex has has distance ∞ to the tree
    u.π = nil //this is the parent of a vertex when in the tree
r.key = 0 //r is root of tree. It has distance 0 to the tree
Q = G.V //put all vertices into the priority queue
while Q ≠ ∅ //while queue is not empty
    u = Extract-Min (Q) //Get min element, initially r
    for each v ∈ G.Adj[u] //check each neighbor of u
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v) //note that this involves an O(lg V)
            //decrease key operation on the min-heap
Prim’s Algorithm: an example
Complexity Analysis of Prim’s Algorithm

• Build Min-heap $O(V)$

• While loop executes $|V|$ times
  - each Extract-Min operation is $\lg V$; total time is $V \lg V$
  - all the for loop calls together in the while loop are $2|E|$
    • within for loop besides constant time the only time cost is the Decrease-Key cost
      $(v.key = w(u,v))$ which is $\lg V$
    • total time for this part is thus $E \lg V$

• Total time is thus $O(V \lg V + E \lg V)$ which is thus $O(E \lg V)$ since for a connected graph $|E| \geq |V| - 1$

• This is asymptotically the same as Kruskal’s algorithm
Summary

- We analyzed two minimum spanning tree algorithms
- We showed proved their correctness by a general argument that applies to both
- We also showed that both algorithms are $O(E \lg V)$
- Both of these graph algorithms can also be classified as greedy algorithms