Minimum Spanning Trees
Shortest Paths

Cormen et. al. VI 23,24
Minimum Spanning Tree

Given a set of locations, with **positive** distances to each other, we want to create a sub-graph that connects all nodes to each other with the minimum sum of distances.

Then that sub-graph is a tree, i.e., has no cycles. **WHY?**

If there is a cycle, we can take one edge out of the cycle and still connect all nodes. (Repeat if there are more cycles.)
Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-complete problems.
  - TSP
- Cluster analysis.

**Minimal or Minimum Spanning Tree?**

Minimum is the smallest possible or allowable amount. Minimal implies that the amount is (relatively) small. Hence **Minimum Spanning Tree**.
Three Greedy Algorithms for MST

Kruskal's algorithm. Start with $T = \emptyset$. Consider edges in ascending order of cost. Add edge $e$ to $T$ unless doing so would create a cycle.

Reverse-Delete algorithm. Start with $T = E$. Consider edges in descending order of cost. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

Prim's algorithm. Start with some node $s$ and greedily grow a tree $T$ from $s$. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$, i.e., without creating a cycle.
The cut property

Simplifying assumption. All edge costs are distinct. In this case the MST is unique. In general it is not, but then all MSTs have the same minimum sum of edges.

Cut property. Let $S$ be a subset of nodes, $S$ neither empty nor equal $V$, and let $e$ be the minimum cost edge with exactly one endpoint in $S$. Then the MST contains $e$. The cut property establishes the correctness of Prim's algorithm.

If there more equal minimum cost edges, just pick one.
The cut property

Cut property. Let $S$ be a subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the MST $T$ contains $e$.

If $e = (v,w)$ is the only edge connecting $S$ and $V-S$ it must be in $T$.

Else, there is another edge $e' = (v',w')$ with $c_{e'} > c_e$ connecting $S$ and $V-S$. Assume $e'$ is in the MST, and not $e$. Adding $e$ to the spanning tree creates a cycle, then taking out $e'$ out removes the cycle creating a new spanning tree with lower cost. Contradiction.

Remember CS220: if we add an edge to a tree we get a cycle, if we take any edge out of that cycle we get a tree again.
Prim's Algorithm

Prim's algorithm. [Jarník 1930, Prim 1957, Dijkstra 1959]

- Initialize $S = \text{any node}$.
- Apply cut property to $S$: add min cost edge $(v, w)$ where $v$ is in $S$ and $w$ is in $V - S$, and add $w$ to $S$.
- Repeat until $S = V$, i.e., greedily growing the MST.
Prim’s algorithm: Implementation

- Maintain set of explored nodes $S$.
- For each unexplored node $v$, maintain attachment cost $a[v] = \text{cost of cheapest edge } v \text{ to a node in } S$.

```
Prim(G,s)
    foreach (v ∈ V)
        priority a[v] ← ∞
    a[s]= 0
    priority queue Q = {}
    foreach (v ∈ V) insert v onto Q (key: a[v] )
    set S ← {}
    while (Q is not empty) {
        u ← delete min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (c_e < a[v]))
                a[v] = c_e
```
Let's do the Prim again, starting at d

{ (d,c), (c,b), (b,i), (b,e), (e,f), (f,g), (g,h), (h,a) }
Kruskal's algorithm [Kruskal, 1956]

Kruskal:
Consider edges in ascending order of weight. Add edge unless doing so would create a cycle.

Cannot add this edge
Kruskal works

1. **Spanning Tree:** Kruskal keeps adding edges until all nodes are connected, and does not create cycles, so produces a spanning tree.

2. **Minimum Spanning Tree:** Consider $e=(v, w)$ added by Kruskal. $S$ is the set of nodes connected to $v$ just before $e$ is added; $v$ is in $S$ and $w$ is not (otherwise we created a cycle). Therefore $e$ is the cheapest edge connecting $S$ to a node in $V-S$, and hence, $e$ is in any MST (cut property).
Reverse-Delete algorithm

Start with $T = E$. Consider edges in descending order of cost. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

Is it always safe to remove $e$, i.e. could $e$ be in an MST?
Cycle property. Let $C$ be any cycle in $G$, and let $e$ be the max cost edge belonging to $C$. Then $e$ doesn’t belong to any MST of $G$.

Let $T$ be a spanning tree that contains max edge $e=(v,w)$. Remove $e$; this will disconnect $T$, creating $S$ containing $v$, and $V-S$ containing $w$.

$C\{-e\}$ is a path $P$. Following $P$ from $v$ will at some stage cross $S$ into $V-S$ by edge $e'$ with lower cost than $e$, so $T - \{e\} + \{e'\}$ is again a spanning tree and its cost is lower than $T$, so $T$ is not an MST.
Shortest Paths Problems

Given a **weighted** graph $G=(V,E)$ find the shortest path
- path length is the sum of its edge weights.

The shortest path from $u$ to $v$ is $\infty$ if there is no path from $u$ to $v$.

Variations of the shortest path problem:
1) **SSSP** (Single source SP): find the SP from some node $s$ to all nodes in the graph.

2) **SPSP** (single pair SP): find the SP from some $u$ to some $v$.
   We can use 1) to solve 2), also there is no more efficient algorithm for 2) than that for 1).

3) **SDSP** (single destination SP) can use 1) by reversing its edges.

4) **APSP** (all pair SPs) could be solved by $|V|$ applications of 1), but there are other approaches (cs420).
Dijkstra's (Greedy) SSSP algorithm only works for graphs with only positive edge weights.

\( S \) is the set of explored nodes. For each \( u \) in \( S \), \( d[u] \) is a distance. Initialize: \( S = \{s\} \) the source, and \( d[s] = 0 \), for all other nodes \( v \) in \( V-S \), \( d[v] = \infty \)

while \( S \neq V \):

   select a node \( v \) in \( V-S \) with at least one edge from \( S \), for which \( d'[v] = \min_{e=(u,v), u \in S} d[u] + w_e \)

   add \( v \) to \( S \) (\( S = S + v \))

   \( d[v] = d'[v] \)

What does \( d' \) represent?

the minimum path length extending with one edge out of \( S \)

To compute the actual minimum paths, maintain an array \( p[v] \) of predecessors. When \( d[v] \) is set to \( d'[v] \), set \( p[v] \) to \( u \).

Notice: Dijkstra is very similar to Prim, but where Dijkstra minimizes path lengths, Prim minimizes edge lengths.
Let's do Dijkstra, starting at d

In this case yes, because all path lengths are different. But in general, multiple path lengths can be equal, and thus can lead to different choices.
Does Dijkstra’s algorithm lead to a Minimum Spanning Tree?

Can you create a counter example?

Shortest paths from A?
Minimum Spanning Tree?

Formulate the difference between Prim and Dijkstra
Dijkstra works

For each \( u \) in \( S \), the path \( P_{s,u} \) is the shortest \((s,u)\) path.

**Proof:** by induction on the size of \( S \)

**Base:** \(|S| = 1\)  \(d[s]=0\)  OK

**Step:** Suppose it holds for \(|S| = k \geq 1\), then grow \( S \) by 1 adding node \( v \) using edge \((u,v)\) \((u \text{ already in } S)\) to create the next \( S \). Then path \( P_{s,u,v} \) is path \( P_{s,u} + (u,v) \), and is the shortest path to \( v \).

**WHY?** What are the "ingredients" of an exchange argument? What are the inequalities?
Greedy exchange argument

Assume there is another path P from s to v. P leaves s with edge (x,y). Then the path P goes from s to x to y to v.

What can you say about $P: s \rightarrow^* x \rightarrow y$ compared to $P_{s,u,v}$? How does the algorithm pick $P_{s,u,v}$? Why does it not work for negative edges?

$P$ from $s$ to $y$ is at least as long as $P_{s,u,v}$ because the algorithm picks the shortest extension out of $S$.

Hence the path

$P: s \rightarrow^* x \rightarrow y \rightarrow^* v$ is at least as long as

$P_{s,u,v}: s \rightarrow^* u \rightarrow v$

This would not work if $w(y,v) < 0$

Corner case?

$s \rightarrow x \rightarrow y = s \rightarrow u \rightarrow v$ AND $y = v$