Minimum Spanning Tree

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge weights $c_e$, an MST is a subset of the edges $T \subseteq E$ such that $T$ is a spanning tree whose sum of edge weights is minimized.

$G = (V, E)$

$T, \sum_{e \in T} c_e = 50$
Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road

- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree

- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy
  - learning salient features for real-time face verification
  - reducing data storage in sequencing amino acids in a protein
  - model locality of particle interactions in turbulent fluid flows
  - autoconfig protocol for Ethernet bridging to avoid cycles in a network

- Cluster analysis.
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cut property. Let $S$ be any subset of nodes (called a cut), and let $e$ be the min cost edge with exactly one endpoint in $S$. Then every MST contains $e$.

Cycle property. Let $C$ be any cycle, and let $f$ be the max cost edge belonging to $C$. Then no MST contains $f$.

[e is in the MST]

[f is not in the MST]
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

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

Pf. By contradiction
- Suppose $e = \{u,v\}$ does not belong to $T^*$.
- Adding $e$ to $T^*$ creates a cycle $C$ in $T^*$.
- There is a path from $u$ to $v$ in $T^* \Rightarrow$ there exists another edge, say $f$, that leaves $S$.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, cost($T'$) < cost($T^*$).
- This is a contradiction.
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cycle property. Let $C$ be any cycle in $G$, and let $f$ be the max cost edge belonging to $C$. Then the MST $T^*$ does not contain $f$.

Pf. By contradiction
- Suppose $f$ belongs to $T^*$.
- Deleting $f$ from $T^*$ cuts $T^*$ into two connected components.
- There exists another edge, say $e$, that is in the cycle and connects the components.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. •
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]
- Consider edges in ascending order of weight.
- **Case 1:** If adding \( e \) to \( T \) creates a cycle, discard \( e \) according to cycle property.
- **Case 2:** Otherwise, insert \( e = (u, v) \) into \( T \) according to cut property where \( S = \) set of nodes in \( u \)'s connected component.

![Case 1](image1.png)  
**Case 1**

![Case 2](image2.png)  
**Case 2**
Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.
- Build set $T$ of edges in the MST.
- Maintain set for each connected component.
- $O(m \log n)$ for sorting and $O(m \log n)$ for union-find.

Kruskal(G, c) {
    Sort edges weights so that $c_1 \leq c_2 \leq \ldots \leq c_m$.
    $T = \{\}$

    foreach $(u \in V)$ make a set containing singleton $u$

    for $i = 1$ to $m$ are $u$ and $v$ in different connected components?
        $(u,v) = e_i$
        if $(u$ and $v$ are in different sets) {
            $T = T \cup \{e_i\}$
            merge the sets containing $u$ and $v$
        } merge two components
    return $T$
}
Union Find Data Structure

- Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

\[ \{V,A,B,C\} \]

\[ \{W,P,Q\} \]
Union Find Data Structure

- Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

To check whether $A, Q$ are in same connected component, follow pointers and check if root is the same.
Union Find Data Structure

- Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

- To **merge** sets, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary).

![Diagram of Union Find Data Structure]
Union Find Data Structure

- Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

- To **merge** sets, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary).

![Diagram of Union Find Data Structure]

1. V,2
2. A,1
3. B,0
4. C,0
5. W,2
6. P,1
7. Q,0
8. R,0
Union Find Data Structure

- Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

- To **merge** sets, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary).
Union Find Data Structure

- To merge sets, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary).

- **Claim:** If the label of a root is $k$, there are at least $2^k$ elements in the set. (Therefore the depth of any tree in algorithm is at most $\log n$)
Union Find Data Structure

- **Claim:** If the label of a root is $k$, there are at least $2^k$ elements in the set. (Therefore the depth of any tree in algorithm is at most $\log n$)
- **Pf:** By induction on $k$. When $k = 0$, this is true. If we merge roots with labels $k_1 > k_2$, the number of vertices only increases while the label stays the same. If $k_1 = k_2$, the merged tree has label $k_1+1$, and by induction, at least $2^{k_1} + 2^{k_2} = 2^{k_1+1}$ elements.
Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- \(O(m \log n)\) for sorting and \(O(m \log n)\) for union-find.

```
Kruskal(G, c) {
    Sort edges weights so that \(c_1 \leq c_2 \leq \ldots \leq c_m\).
    T = {}

    foreach (u ∈ V) make a set containing singleton u

    for i = 1 to m
        (u,v) = e_i
        if (u and v are in different sets) {
            T = T ∪ \{e_i\}
            merge the sets containing u and v
        }
    return T
}
```
Removing the assumption that edge weights are distinct

Suppose edge weights are not distinct, and Kruskal’s algorithm sorts edges so
\[ w(e_1) \leq w(e_2) \leq ... \leq w(e_m) \]

Suppose Kruskal finds MST \( T \) of weight \( w(T) \), but the optimal solution \( T^* \) has weight \( w(T^*) < w(T) \).

Perturb each of the weights by a very small amount so that

\[ w'(e_1) < w'(e_2) < ... < w'(e_m) \]

If the perturbation is small enough, \( w'(T^*) < w'(T) \). However, this contradicts the correctness of Kruskal’s algorithm, since the algorithm will still find \( T \)!
Greedy Algorithms

**Kruskal's algorithm.** Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge $e$ in $T$ 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 root node $s$ and greedily grow a tree $T$ from $s$ outward. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$.

**Remark.** All three algorithms produce an MST.