Minimum Spanning Tree

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

Multiple Greedy Algorithms Solve MST!

- **Prim's algorithm**: Greedily grow the tree adding minimum cost edge between \( S \) and \( V-S \).
- **Kruskal's algorithm**: Sort all edges from low to high cost and add an edge as long as it does not create a cycle.
- **Reverse-Delete algorithm**: Start with \( T = G \). Sort edges from high to low cost. Delete an edge as long as it does not disconnect the graph.
Kruskal’s Algorithm

\[ T = \{ \} \]

Sort edges by cost

for each edge \( e \) in order of ascending cost {
    if \( T \cup \{ e \} \) does not contain a cycle {
        \[ T = T \cup \{ e \} \]
    }
}

When does adding an edge create a cycle?

Sorted edges: \( (a,c), (a,b), (b,d), (e,f), (c,d), (c,e), (a,d), (b,e) \)

\[ T = \{ \} \]

Should we add edge \( (a, c) \) to the MST?

When does adding an edge create a cycle?

Sorted edges: \( (a,b), (b,d), (e,f), (c,d), (c,e), (a,d), (b,e) \)

\[ T = \{ (a, c) \} \]

Should we add edge \( (a, b) \) to the MST?
When does adding an edge create a cycle?

Sorted edges:  (b,d), (e,f), (c,d), (c,e), (a,d), (b,e)
T = {(a, c), (a, b)}

Should we add edge (b, d) to the MST?

When does adding an edge create a cycle?

Sorted edges:  (e,f), (c,d), (c,e), (a,d), (b,e)
T = {(a, c), (a, b), (b, d)}

Should we add edge (e, f) to the MST?

When does adding an edge create a cycle?

Sorted edges:  (c,d), (c,e), (a,d), (b,e)
T = {(a, c), (a, b), (b, d), (e, f)}

Should we add edge (c, d) to the MST?
When does adding an edge create a cycle?

Sorted edges: (c,e), (a,d), (b,e)
T = {(a, c), (a, b), (b, d), (e, f)}

Should we add edge (c, e) to the MST?

When does adding an edge create a cycle?

Sorted edges: (a,d), (b,e)
T = {(a, c), (a, b), (b, d), (e, f), (c, e)}

Should we add edge (a, d) or (b, e) to the MST?

Refined Kruskal’s Algorithm

Kruskal(G, c) {
    Sort edges by weights so that c₁ ≤ c₂ ≤ ... ≤ cₘ.
    T ← {};
    foreach (u ∈ V) make a set containing singleton u
    for each edge e = (u, v)
        if (u and v are in different sets) {
            T ← T ∪ {e}
            merge the sets containing u and v
        }
    return T
}
**Union-Find**

- Data structure that allows us to quickly find which set an element is in and to union 2 sets
- Assumptions:
  - Sets are disjoint
  - We are not interested in splitting sets
- Operations:
  - MakeUnionFind(G) - create the initial sets each containing 1 node
  - Find(v) - determine which set a node is in
  - Union(S₁, S₂) - union 2 sets

**Union-Find Data Structure**

// Array maps a node to the name of the set it is in.
String[] components;

- Cost of MakeUnionFind(G)? \(O(n)\)
- Cost of Find(v)? \(O(1)\)
- Cost of Union(S₁, S₂)? \(O(n)\)
Union Operation

Union (S₁, S₂) {
    for i = 1 to n {
        if (components[i] = S₁ || components[i] = S₂) {
            components[i] = S_new;
        }
    }
}

How can we avoid walking the entire array?
How can we avoid renaming all the elements in S₁ and S₂?

Cost of Kruskal's Algorithm?

Kruskal(G, c) {
    Sort edge weights so that c₁ ≤ c₂ ≤ ... ≤ cᵣ.
    T ← {}.

    MakeUnionFind (G);
    for each edge eᵢ = (u, v) {
        S₁ = Find(u);
        S₂ = Find(v);
        if (S₁ ≠ S₂) {
            T ← T ∪ {eᵢ};
            Union (S₁, S₂);
        }
    }
    return T;
}