Minimum Spanning Tree

**MST.** Given connected graph $G$ with positive edge weights, find a min weight set of edges that connects all of the vertices.

**Theorem.** [Cayley, 1889] There are $V^2 - V$ spanning trees on the complete graph on $V$ vertices.

**MST Origin**

Otakar Boruvka (1926).
- Electrical Power Company of Western Moravia in Brno.
- Most economical construction of electrical power network.
- Concrete engineering problem is now a cornerstone problem in combinatorial optimization.
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.

Medical Image Processing

MST describes arrangement of nuclei in the epithelium for cancer research

Two Greedy Algorithms

Kruskal’s algorithm. Consider edges in ascending order of cost. Add the next edge to $T$ unless doing so would create a cycle.

Prim’s algorithm. Start with any vertex $s$ and greedily grow a tree $T$ from $s$. At each step, add the cheapest edge to $T$ that has exactly one endpoint in $T$.

Theorem. Both greedy algorithms compute an MST.

Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.” - Gordon Gecko
Weighted Graphs

Edge Data Type

```java
public class Edge implements Comparable<Edge> {
    public final int v, int w;
    public final double weight;

    public Edge(int v, int w, double weight) {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int other(int vertex) {
        if (vertex == v) return w;
        else return v;
    }

    public int compareTo(Edge e) {  
        if (w < e.weight) return -1;
        else if (w > e.weight) return +1;
        else return 0;
    }
}
```

Weighted Graph Interface

```java
public class WeightedGraph {  
    private int V;  // # vertices
    private Sequence<Edge>[] adj;  // adjacency lists

    public WeightedGraph(int V) {  
        this.V = V;
        adj = (Sequence<Edge>[] ) new Sequence[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Sequence<Edge>();
    }

    public void insert(Edge e) {  
        adj[v].add(e);
        adj[w].add(e);
    }

    public Iterable<Edge> adj(int v) { return adj[v]; }
}
```

Weighted Graph: Java Implementation

Identical to Graph.java but use Edge adjacency lists instead of int.
MST Structure

**MST.** Given connected graph $G$ with positive edge weights, find a min weight set of edges that connects all of the vertices.

**Def.** A spanning tree of a graph $G$ is a subgraph $T$ that is connected and acyclic.

**Property.** MST of $G$ is always a spanning tree.

**Greedy Algorithms**

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

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

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

**Spanning Tree**

**Cycle Property**

**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 $T^*$. Let’s see what happens.
- Deleting $f$ from $T^*$ disconnects $T^*$. Let $S$ be one side of the cut.
- Some other edge in $C$, say $e$, has exactly one endpoint in $S$.
- $T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, cost$(T) <$ cost$(T^*)$.
- This is a contradiction. •
**Cut Property**

**Simplifying assumption.** All edge costs \( c_e \) are distinct.

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

**Pf.** [by contradiction]
- Suppose \( e \) does not belong to \( T^* \). Let’s see what happens.
- Adding \( e \) to \( T^* \) creates a (unique) cycle \( C \) in \( T^* \).
- Some other edge in \( C \), say \( f \), has exactly one endpoint in \( S \).
- \( T = T^* \cup \{ e \} - \{ f \} \) is also a spanning tree.
- Since \( c_e < c_f \), \( \text{cost}(T) < \text{cost}(T^*) \).
- This is a contradiction. 

---

**Kruskal’s Algorithm**

**Kruskal’s algorithm.** [Kruskal, 1956] Consider edges in ascending order of cost. Add the next edge to \( T \) unless doing so would create a cycle.

---

**Kruskal’s Algorithm: Example**

<table>
<thead>
<tr>
<th>3-5</th>
<th>1-7</th>
<th>6-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>0-7</td>
<td>0-1 3-4</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
</table>
Kruskal’s Algorithm: Proof of Correctness

**Theorem.** Kruskal’s algorithm computes the MST.

**Pf.** [case 1] If adding $e$ to $T$ creates a cycle $C$, then $e$ is the max weight edge in $C$. The cycle property asserts that $e$ is not in the MST.

![Diagram of a graph with a cycle](image)

Kruskal’s Algorithm: Implementation

**Q.** How to check if adding an edge to $T$ would create a cycle?

**A1.** Naïve solution: use DFS.
- $O(V)$ time per cycle check.
- $O(EV)$ time overall.

**A2.** Use the union-find data structure.
- Maintain a set for each connected component.
- If $v$ and $w$ are in same component, then adding $v$-$w$ creates a cycle.
- To add $v$-$w$ to $T$, merge sets containing $v$ and $w$.

![Diagram showing two cases](image)
Kruskal’s Algorithm: Java Implementation

```java
public class Kruskal {
    private Sequence<Edge> mst = new Sequence<Edge>();
    public Kruskal(WeightedGraph G) {
        // sort edges in ascending order
        Edge[] edges = G.edges();
        Arrays.sort(edges);
        // greedily add edges to MST
        UnionFind uf = new UnionFind(G.V());
        for (int i = 0; i < E && mst.size() < G.V()-1; i++) {
            int v = edges[i].v;
            int w = edges[i].w;
            if (!uf.find(v, w)) { // safe to stop early if
                uf.unite(v, w); // tree already has V-1 edges
                mst.add(edges[i]);
            }
        }
    }
    public Iterable<Edge> mst() { return mst; }
}
```

Kruskal’s Algorithm: Running Time

Kruskal running time. \(O(E \log V)\).

\(E = V^2\) so \(O(\log E) = O(\log V)\)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Frequency</th>
<th>Time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>1</td>
<td>(E \log V)</td>
</tr>
<tr>
<td>union</td>
<td>(V)</td>
<td>(\log^* V)</td>
</tr>
<tr>
<td>find</td>
<td>(E)</td>
<td>(\log^* V)</td>
</tr>
</tbody>
</table>

Remark. If edges already sorted: \(O(E \log^* V)\) time.

recall: \(\log^* V = 5\) in this universe

Prim’s Algorithm: Example

Prim’s Algorithm. \([\text{Jarník 1930}, \text{Dijkstra 1957}, \text{Prim 1959}]\)

Start with vertex 0 and greedily grow tree \(T\). At each step, add cheapest edge that has exactly one endpoint in \(T\).
Prim’s Algorithm: Example

Prim’s Algorithm: Proof of Correctness

**Theorem.** Prim’s algorithm computes the MST.

**Pf.**
- Let $S$ be the subset of vertices in current tree $T$.
- Prim adds the cheapest edge $e$ with exactly one endpoint in $S$.
- Cut property asserts that $e$ is in the MST.

Prim’s Algorithm: Implementation

**Q.** How to find cheapest edge with exactly one endpoint in $S$?

**A1.** Brute force: try all edges.
- $O(E)$ time per spanning tree edge.
- $O(E V)$ time overall.

**A2.** Maintain edges with (at least) one endpoint in $S$ in a priority queue.
- Delete min to determine next edge $e$ to add to $T$.
- Disregard $e$ if both endpoints are in $S$.
- Upon adding $e$ to $T$, add to PQ the edges incident to one endpoint.

**Running time.**
- $O(\log V)$ time per edge (using a binary heap).
- $O(E \log V)$ time overall.
Removing the Distinct Edge Costs Assumption

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

Fact. Prim and Kruskal don’t actually rely on the assumption. only our proof of correctness does!

Removing the Distinct Edge Costs Assumption

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

One way to remove assumption. Kruskal and Prim only access edge weights through `compareTo()`; suffices to introduce tie-breaking rule.

```java
public class LazyPrim {
    private Sequence<Edge> mst = new Sequence<Edge>();
    
    public LazyPrim(WeightedGraph G) { is v in S?
        boolean[] marked = new boolean[G.V()];
        MinPQ<Edge> pq = new MinPQ<Edge>();
        int s = 0; 
        marked[s] = true; add all edges incident to s
        for (Edge e : G.adj(s)) pq.insert(s);
        
        while (!pq.isEmpty()) { add edge to MST unless both endpoints are already in S
            Edge e = pq.delMin(); int v = e.v, w = e.w;
            if (!marked[v] || !marked[w]) mst.add(e);
            if (!marked[v]) for (Edge f : G.adj(v)) pq.insert(f);
            if (!marked[w]) for (Edge f : G.adj(w)) pq.insert(f);
            marked[v] = marked[w] = true. these edges have exactly one endpoint in S
        }
    }
}
```

Advanced MST Algorithms
Euclidean MST

*Key geometric fact.* Edges of the Euclidean MST are edges of the Delaunay triangulation.

**Euclidean MST algorithm.**
- Compute Voronoi diagram to get Delaunay triangulation.
- Run Kruskal’s MST algorithm on Delaunay edges.

**Running time.** \( O(N \log N) \).
- Fact: \( 3N \) Delaunay edges since it’s planar.
- \( O(N \log N) \) for Voronoi.
- \( O(N \log N) \) for Kruskal.

**Lower bound.** Any comparison-based Euclidean MST algorithm requires \( \Omega(N \log N) \) comparisons.

Brute force. Compute \( \Theta(N^2) \) distances and run Prim’s algorithm.

Ingenuity. Exploit geometry and do it in \( O(N \log N) \).