**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.

![Graph](image)

$\text{cost}(T) = 50$

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

**MST Origin**

Otakar Borůvka (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.

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.

Medical Image Processing

MST describes arrangement of nuclei in the epithelium for cancer research

Weighted Graphs

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

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>WeightedGraph(int V)</td>
<td>create empty graph</td>
</tr>
<tr>
<td></td>
<td>insert(Edge e)</td>
<td>add edge e</td>
</tr>
<tr>
<td>Iterable&lt;Edge&gt;</td>
<td>adj(int v)</td>
<td>return iterator over edges incident to v</td>
</tr>
<tr>
<td>int</td>
<td>V()</td>
<td>return number of vertices</td>
</tr>
<tr>
<td>String</td>
<td>toString()</td>
<td>return string representation</td>
</tr>
</tbody>
</table>

for (int v = 0; v < G.V(); v++) {
    for (Edge e : G.adj(v)) {
        int w = e.other(v);
        // edge v-w
    }
}

Iterate through all edges (once in each direction)

Weighted Graph: Java Implementation

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

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

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

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

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

Edge Data Type

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 f) {
        Edge e = this;
        if (e.weight < f.weight) return -1;
        else if (e.weight > f.weight) return +1;
        else return 0;
    }
}

MST Structure
Spanning Tree

**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 $e$.

**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.  

**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 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 $e$.

**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 $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.
- 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

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 $C$, so the cycle property asserts that $e$ is not in the MST.
Theorem. Kruskal’s algorithm computes the MST.

Pf (case 2). If adding $e = (v, w)$ to $T$ does not create a cycle, then $e$ is the min weight edge with exactly one endpoint in $S$, so the cut property asserts that $e$ is in the MST. If adding $e = (v, w)$ to $T$ does not create a cycle, then

Case 1: adding $v$-$w$ creates a cycle

Case 2: add $v$-$w$ to $T$ and merge sets

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 $w$.

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)) {
                uf.unite(v, w);
                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 \( O(\log E) = O(\log V) \)

### Operation | Frequency | Cost
--- | --- | ---
sort | 1 | \( E \log V \)
union | \( V-1 \) | \( \log^* V \uparrow \)
find | \( E \) | \( \log^* V \uparrow \)

\( \uparrow \) amortized bound using weighted quick union with path compression

**Remark.** If edges already sorted: \( O(E \log^* V) \) time.
recall: \( \log^* V \approx 5 \) in this universe

Prim's Algorithm

Prim's algorithm. [Jarník 1930, Dijkstra 1957, 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: 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.

Running time.
- $O(\log V)$ time per edge (using a binary heap).
- $O(E \log V)$ time overall.

Prim's Algorithm: Implementation

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

- $O(E)$ time per spanning tree edge.
- $O(EV)$ 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 edges incident to one endpoint to PQ.

Running time.
- $O(\log V)$ time per edge (using a binary heap).
- $O(E \log V)$ time overall.

Prim's Algorithm: Java Implementation

```java
public class LazyPrim {
    private Sequence<Edge> mst = new Sequence<Edge>();

    public LazyPrim(WeightedGraph G) {
        boolean[] marked = new boolean[G.V()];
        MinPQ<Edge> pq = new MinPQ<Edge>();
        marked[0] = true;
        for (Edge e : G.adj(0)) pq.insert(e);

        while (!pq.isEmpty()) {
            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;
        }
    }
}
```
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 $\text{compareTo}$; suffices to introduce tie-breaking rule.

```java
public int compareTo(Edge f) {
    Edge e = this;
    if (e.weight < f.weight) return -1;
    if (e.weight > f.weight) return +1;
    if (e.v < f.v) return -1;
    if (e.v > f.v) return +1;
    if (e.w < f.w) return -1;
    if (e.w > f.w) return +1;
    return 0;
}
```

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

Advanced MST Algorithms

<table>
<thead>
<tr>
<th>Year</th>
<th>Problem</th>
<th>Time</th>
<th>Discovered By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1975</td>
<td>Planar MST</td>
<td>$E$</td>
<td>Cheriton-Tarjan</td>
</tr>
<tr>
<td>1992</td>
<td>MST Verification</td>
<td>$E$</td>
<td>Dixon-Rauch-Tarjan</td>
</tr>
<tr>
<td>1995</td>
<td>Randomized MST</td>
<td>$E$</td>
<td>Karger-Klein-Tarjan</td>
</tr>
</tbody>
</table>

Related problems
Euclidean MST

**Euclidean MST.** Given N points in the plane, find MST connecting them.
- Distances between point pairs are **Euclidean** distances.

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

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

---

Clustering

**Clustering.** Given a set of objects classify into coherent groups.

- Distance function. Numeric value specifying "closeness" of two objects.
- Fundamental problem. Divide into clusters so that points in different clusters are far apart.
  - Routing in mobile ad hoc networks.
  - Identify patterns in gene expression.
  - Document categorization for web search.
  - Similarity searching in medical image databases.
  - Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

---

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: $\leq 3N$ 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.

---

Clustering

**Clustering.** Given a set of objects classify into coherent groups.

- Distance function. Numeric value specifying "closeness" of two objects.
- Fundamental problem. Divide into clusters so that points in different clusters are far apart.
  - Routing in mobile ad hoc networks.
  - Identify patterns in gene expression.
  - Document categorization for web search.
  - Similarity searching in medical image databases.
  - Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.
**Clustering of Maximum Spacing**

**k-clustering.** Divide objects into k non-empty groups.

**Distance function.** Assume it satisfies several natural properties.
- \( c(v, w) = 0 \) iff \( v = w \) (identity of indiscernibles)
- \( c(v, w) \neq 0 \) (nonnegativity)
- \( c(v, w) = c(w, v) \) (symmetry)

**Spacing.** Min distance between any pair of points in different clusters.

**Clustering of maximum spacing.** Given an integer \( k \), find a \( k \)-clustering of maximum spacing.

**Dendrogram.** Scientific visualization of hypothetical sequence of evolutionary events.
- Leaves = genes.
- Internal nodes = hypothetical ancestors.

**Tumors in similar tissues cluster together.**

**Reference:** Botstein & Brown group