Minimum Spanning Trees

- weighted graph API
- cycles and cuts
- Kruskal’s algorithm
- Prim’s algorithm
- advanced topics

References:
Algorithms in Java, Chapter 20
http://www.cs.princeton.edu/introalgsds/54mst
**Minimum Spanning Tree**

*Given.* Undirected graph $G$ with positive edge weights (connected).

*Goal.* Find a min weight set of edges that connects all of the vertices.
Minimum Spanning Tree

**Given.** Undirected graph $G$ with positive edge weights (connected).

**Goal.** Find a min weight set of edges that connects all of the vertices.

Brute force: Try all possible spanning trees
- problem 1: not so easy to implement
- problem 2: far too many of them

Ex: [Cayley, 1889]: $V^2$ spanning trees on the complete graph on $V$ vertices.

weight($T$) = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7
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-solving model 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

http://www.bccrc.ca/ci/ta01_archlevel.html
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 \( T \).

**Proposition.** 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 graph API
- cycles and cuts
- Kruskal’s algorithm
- Prim’s algorithm
- advanced topics
**Weighted Graph API**

```java
public class WeightedGraph
{
    WeightedGraph(int V) // create an empty graph with V vertices
    void insert(Edge e) // insert edge e
    Iterable<Edge> adj(int v) // return an iterator over edges incident to v
    int V() // return the number of vertices
    String toString() // return a string representation
}
```

iterate through all edges (once in each direction)
Identical to `Graph.java` but use `Edge` adjacency sets instead of `int`.

```java
public class WeightedGraph {
    private int V;
    private SET<Edge>[] adj;

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

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

    public Iterable<Edge> adj(int v) {
        return adj[v];
    }
}
```
public class Edge implements Comparable<Edge> {
    private final int v, int w;
    private final double weight;

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

    public int either() { return v; }

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

    public int weight() { return weight; }

    // See next slide for edge compare methods.
}

Edge abstraction needed for weights

slightly tricky accessor methods (enables client code like this)

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

Two different compare methods for edges

- `compareTo()` so that edges are `Comparable` (for use in `SET`)
- `compare()` so that clients can compare edges by weight.

```java
public final static Comparator<Edge> BY_WEIGHT = new ByWeightComparator();

private static class ByWeightComparator implements Comparator<Edge> {
    public int compare(Edge e, Edge f) {
        if (e.weight < f.weight) return -1;
        if (e.weight > f.weight) return +1;
        return 0;
    }
}

public int compareTo(Edge that) {
    if (this.weight < that.weight) return -1;
    else if (this.weight > that.weight) return +1;
    else return 0;
}
```
weighted graph API
- cycles and cuts
- Kruskal’s algorithm
- Prim’s algorithm
- advanced topics
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 weights $w_e$ 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 $e$.

![Diagram showing f is not in the MST and e is in the MST](image)
**Cycle Property**

**Simplifying assumption.** All edge weights \( w_e \) are distinct.

**Cycle property.** Let \( C \) be any cycle, 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(\( T^* \)).
- Contradicts minimality of \( T^* \).  

![Diagram](image)
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 $c_e < c_f$, cost($T$) < cost($T^*$).
- Contradicts minimality of $T^*$.

\[ \text{MST } T^* \]

\[ \text{cycle } C \]

\[ S \]

\[ e \]

\[ f \]
- weighted graph API
- cycles and cuts
- **Kruskal’s algorithm**
- Prim’s algorithm
- advanced algorithms
- clustering
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
Proposition. Kruskal’s algorithm computes the MST.

Pf. [case 1] Suppose that adding e to T creates a cycle C
• e is the max weight edge in C (weights come in increasing order)
• e is not in the MST (cycle property)
**Proposition.** Kruskal's algorithm computes the MST.

**Pf.** [case 2] Suppose that adding $e = (v, w)$ to $T$ does not create a cycle

- let $S$ be the vertices in $v$'s connected component
- $w$ is not in $S$
- $e$ is the min weight edge with exactly one endpoint in $S$
- $e$ is in the MST (cut property)
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.
Kruskal's algorithm implementation

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

**A2.** Use the union-find data structure from lecture 1 (!).
- 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.

\[
\begin{align*}
\text{Case 1: adding v-w creates a cycle} & \quad \text{Case 2: add v-w to T and merge sets}
\end{align*}
\]
Kruskal's algorithm: Java implementation

```java
public class Kruskal
{
    private SET<Edge> mst = new SET<Edge>();

    public Kruskal(WeightedGraph G)
    {
        Edge[] edges = G.edges();
        Arrays.sort(edges, Edge.BY_WEIGHT);

        UnionFind uf = new UnionFind(G.V());
        for (Edge e : edges)
            if (!uf.find(e.either(), e.other()))
            {
                uf.unite(e.either(), e.other());
                mst.add(edge);
            }
    }

    public Iterable<Edge> mst()
    { return mst; } }
```

Easy speedup: Stop as soon as there are V-1 edges in MST.
Kruskal's algorithm running time

Kruskal running time: Dominated by the cost of the sort.

<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>$E \log E$</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>

† amortized bound using weighted quick union with path compression

recall: $\log^* V \leq 5$ in this universe

Remark 1. If edges are already sorted, time is proportional to $E \log^* V$

Remark 2. Linear in practice with PQ or quicksort partitioning
(see book: don't need full sort)
weight graph API
cycles and cuts
Kruskal’s algorithm
Prim’s algorithm
advanced topics
Prim's algorithm example

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 example
Proposition. 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 $S$.
- $e$ is in the MST (cut property)
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.
Prim's algorithm implementation

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

**A2.** Maintain a priority queue of vertices connected by an edge to $S$
- Delete min to determine next vertex $v$ to add to $S$.
- Disregard $v$ if already in $S$.
- Add to PQ any vertex brought closer to $S$ by $v$.

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

**Note:** This is a lazy version of implementation in Algs in Java

- **Lazy:** put all adjacent vertices (that are not already in MST) on PQ
- **Eager:** first check whether vertex is already on PQ and decrease its key
Key-value priority queue

Associate a value with each key in a priority queue.

API:

```java
public class MinPQplus<Key extends Comparable<Key>, Value>

    MinPQplus() // create a key-value priority queue
    void put(Key key, Value val) // put key-value pair into the priority queue
    Value delMin() // return value paired with minimal key
    Key min() // return minimal key
```

Implementation:

- start with same code as standard heap-based priority queue
- use a parallel array `vals[]` (value associated with `keys[i]` is `vals[i]`)
- modify `exch()` to maintain parallel arrays (do `exch` in `vals[]`)
- modify `delMin()` to return `value`
- add `min()` (just returns `keys[1]`)
Lazy implementation of Prim's algorithm

```java
public class LazyPrim
{
    Edge[] pred = new Edge[G.V()];

    public LazyPrim(WeightedGraph G)
    {
        boolean[] marked = new boolean[G.V()];
        double[] dist = new double[G.V()];
        MinPQplus<Double, Integer> pq;
        pq = new MinPQplus<Double, Integer>();
        dist[s] = 0.0;
        marked[s] = true;
        pq.put(dist[s], s);
        while (!pq.isEmpty())
        {
            int v = pq.delMin();
            if (marked[v]) continue;
            marked(v) = true;
            for (Edge e : G.adj(v))
            {
                int w = e.other(v);
                if (!done[w] && (dist[w] > e.weight()))
                {
                    dist[w] = e.weight(); pred[w] = e;
                    pq.insert(dist[w], w);
                }
            }
        }
    }
}
```

- `pred[v]` is edge attaching `v` to MST
- Marks vertices in MST
- Distance to MST
- Key-value PQ
- Get next vertex
- Ignore if already in MST
- Add to PQ any vertices brought closer to S by v
Prim's algorithm (lazy) example

Priority queue **key** is distance (edge weight); **value** is vertex

Lazy version leaves obsolete entries in the PQ therefore may have multiple entries with same value

```
0-1 0.32
0-2 0.29
0-5 0.60
0-6 0.51
0-7 0.31
1-7 0.21
3-4 0.34
3-5 0.18
4-5 0.40
4-6 0.51
4-7 0.46
6-7 0.25
```
Eager implementation of Prim’s algorithm

Use indexed priority queue that supports
- contains: is there a key associated with value v in the priority queue?
- decrease key: decrease the key associated with value v

[more complicated data structure, see text]

Putative “benefit”: reduces PQ size guarantee from E to V
- not important for the huge sparse graphs found in practice
- PQ size is far smaller in practice
- widely used, but practical utility is debatable
Removing the distinct edge costs assumption

**Simplifying assumption.** All edge weights $w_e$ are distinct.

**Fact.** Prim and Kruskal don't actually rely on the assumption (our proof of correctness does)

Suffices to introduce tie-breaking rule for `compare()`.

**Approach 1:**

```java
public int compare(Edge e, Edge f) {
    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;
}
```

**Approach 2:** add tiny random perturbation.
• weighted graph API
• cycles and cuts
• Kruskal’s algorithm
• Prim’s algorithm

• advanced topics
**Advanced MST theorems: does an algorithm with a linear-time guarantee exist?**

<table>
<thead>
<tr>
<th>Year</th>
<th>Worst Case</th>
<th>Discovered By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1975</td>
<td>$E \log \log V$</td>
<td>Yao</td>
</tr>
<tr>
<td>1976</td>
<td>$E \log \log V$</td>
<td>Cheriton-Tarjan</td>
</tr>
<tr>
<td>1984</td>
<td>$E \log^* V$, $E + V \log V$</td>
<td>Fredman-Tarjan</td>
</tr>
<tr>
<td>1986</td>
<td>$E \log (\log^* V)$</td>
<td>Gabow-Galil-Spencer-Tarjan</td>
</tr>
<tr>
<td>1997</td>
<td>$E \alpha(V) \log \alpha(V)$</td>
<td>Chazelle</td>
</tr>
<tr>
<td>2000</td>
<td>$E \alpha(V)$</td>
<td>Chazelle</td>
</tr>
<tr>
<td>2002</td>
<td>optimal</td>
<td>Pettie-Ramachandran</td>
</tr>
<tr>
<td>20xx</td>
<td>$E$</td>
<td>???</td>
</tr>
</tbody>
</table>

**deterministic comparison based MST algorithms**

<table>
<thead>
<tr>
<th>Year</th>
<th>Problem</th>
<th>Time</th>
<th>Discovered By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1976</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 $N^2 / 2$ distances and run Prim's algorithm.
Ingenuity. Exploit geometry and do it in $O(N \log N)$
[stay tuned for geometric algorithms]
Scientific application: clustering

**k-clustering.** Divide a set of objects classify into k 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.

**Applications.**
- 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.

Outbreak of cholera deaths in London in 1850s. Reference: Nina Mishra, HP Labs
k-clustering of maximum spacing

**k-clustering.** Divide a set of objects classify into k coherent groups.

**distance function.** Numeric value specifying "closeness" of two objects.

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

**k-clustering of maximum spacing.**

Given an integer k, find a k-clustering such that spacing is maximizing.
Single-link clustering algorithm

“Well-known” algorithm for single-link clustering:
• Form $V$ clusters of one object each.
• Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
• Repeat until there are exactly $k$ clusters.

Observation. This procedure is precisely Kruskal's algorithm (stop when there are $k$ connected components).

Property. Kruskal's algorithm finds a $k$-clustering of maximum spacing.
Clustering application: dendrograms

Dendrogram.
Scientific visualization of hypothetical sequence of evolutionary events.

- Leaves = genes.
- Internal nodes = hypothetical ancestors.

Dendrogram of cancers in human

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group