4.3 Minimum Spanning Trees

- introduction
- cut property
- edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm

https://algs4.cs.princeton.edu
4.3 Minimum Spanning Trees

- introduction
- cut property
- edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm
Def. A spanning tree of $G$ is a subgraph $T$ that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.
Def. A spanning tree of $G$ is a subgraph $T$ that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.
**Spanning tree**

**Def.** A spanning tree of $G$ is a subgraph $T$ that is:
- A tree: connected and acyclic.
- Spanning: includes all of the vertices.
Def. A spanning tree of $G$ is a subgraph $T$ that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.
Minimum spanning tree problem

Input. Connected, undirected graph $G$ with positive edge weights.
Minimum spanning tree problem

**Input.** Connected, undirected graph $G$ with positive edge weights.

**Output.** A spanning tree of minimum weight.

---

**Brute force.** Try all spanning trees?
Let $T$ be any spanning tree of a connected graph $G$ with $V$ vertices. Which of the following properties must hold?

A. $T$ contains exactly $V - 1$ edges.
B. Removing any edge from $T$ disconnects it.
C. Adding any edge to $T$ creates a cycle.
D. All of the above.
Network design

https://www.utdallas.edu/~besp/teaching/mst-applications.pdf
Dendrogram of cancers in human

Reference: Botstein & Brown group
More MST applications

MST describes arrangement of nuclei in the epithelium for cancer research

http://www.bccrc.ca/ci/ta01_archlevel.html

http://ginger.indstate.edu/ge/gfx

https://www.youtube.com/watch?v=GwKuFREOgmo

http://www.flickr.com/photos/quasimondo/2695389651

http://algo.inria.fr/broutin/gallery.html
Applications

MST is fundamental problem with diverse applications.

- Cluster analysis.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Curvilinear feature extraction in computer vision.
- Find road networks in satellite and aerial imagery.
- Handwriting recognition of mathematical expressions.
- Measuring homogeneity of two-dimensional materials.
  Model locality of particle interactions in turbulent fluid flows.
- Reducing data storage in sequencing amino acids in a protein.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Network design (communication, electrical, hydraulic, computer, road).
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).

http://www.utdallas.edu/~besp/teaching/mst-applications.pdf
4.3 Minimum Spanning Trees

- introduction
- cut property
- edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm

https://algs4.cs.princeton.edu
Simplifying assumptions

For simplicity, we assume:

- The graph is connected. \( \Rightarrow \) MST exists.
- The edge weights are distinct. \( \Rightarrow \) MST is unique. \( \text{see Exercise 4.3.3} \)

Note. Algorithms still work even if duplicate edge weights.
**Cut property**

**Def.** A cut in a graph is a partition of its vertices into two (nonempty) sets.

**Def.** A crossing edge is an edge that has one endpoint in each set.

**Cut property.** For any cut, the min-weight crossing edge is in the MST.
Which is the min-weight edge crossing the cut \{2, 3, 5, 6\}? 

A. 0–7 (0.16) 
B. 2–3 (0.17) 
C. 0–2 (0.26) 
D. 5–7 (0.28)
**Cut property**

**Def.** A *cut* in a graph is a partition of its vertices into two (nonempty) sets.

**Def.** A *crossing edge* is an edge that has one endpoint in each set.

**Cut property.** For any cut, the min-weight crossing edge $e$ is in the MST.

**Note.** A cut may have multiple edges in the MST.
**Cut property: correctness proof**

**Def.** A **cut** in a graph is a partition of its vertices into two (nonempty) sets.

**Def.** A **crossing edge** is an edge that has one endpoint in each set.

**Cut property.** For any cut, the min-weight crossing edge $e$ is in the MST.

**Pf.** [by contradiction] Suppose $e$ is not in the MST $T$.

- Adding $e$ to the MST creates a cycle.
- Some other edge $f$ in cycle must be a crossing edge.
- Removing $f$ and adding $e$ yields a different spanning tree $T'$.
- Since $\text{weight}(e) < \text{weight}(f)$, we have $\text{weight}(T') < \text{weight}(T)$.
- Contradiction. □
Framework for minimum spanning tree algorithm

\[
\begin{align*}
\textbf{Generic algorithm (to compute MST)} \\
\quad T &= \emptyset. \\
\text{Repeat until } T \text{ is a spanning tree:} & \quad V-1 \text{ edges} \\
& \quad \quad - \text{Find a cut in } G. \\
& \quad \quad - e \leftarrow \text{min-weight crossing edge.} \\
& \quad \quad - T \leftarrow T \cup \{ e \}. \\
\end{align*}
\]

Efficient implementations.

- Which cut? \(2^{V-2}\) distinct cuts
- How to compute min-weight crossing edge.

\textbf{Ex 1.} Kruskal’s algorithm.
\textbf{Ex 2.} Prim’s algorithm.
\textbf{Ex 3.} Borůvka’s algorithm.
4.3 **Minimum Spanning Trees**

- introduction
- cut property
- edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm
## Weighted edge API

**API.** Edge abstraction for weighted edges.

```java
public class Edge implements Comparable<Edge> {
    public Edge(int v, int w, double weight) {
        // create a weighted edge v–w
    }
    public int either() {
        // either endpoint
    }
    public int other(int v) {
        // the endpoint that’s not v
    }
    public int compareTo(Edge that) {
        // compare edges by weight
    }
    // ...
}
```

Diagram:

```
  v---weight
    ^
    |  edge e = v–w
    |  w
```

**Idiom for processing an edge e.** `int v = e.either(), w = e.other(v);`
public class Edge implements Comparable<Edge> {
    private final int v, 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 compareTo(Edge that) {
        return Double.compare(this.weight, that.weight);
    }
}
Edge-weighted graph API

API. Same as Graph and Digraph, except with explicit Edge objects.

```java
public class EdgeWeightedGraph

    EdgeWeightedGraph(int V)  // create an empty graph with V vertices

    void addEdge(Edge e)  // add weighted edge e to this graph

    Iterable<Edge> adj(int v)  // edges incident to v

    ;  ;
```
Edge-weighted graph: adjacency-lists representation

**Representation.** Maintain vertex-indexed array of Edge lists.
Edge-weighted graph: adjacency-lists implementation

```java
public class EdgeWeightedGraph
{
    private final int V;
    private final Bag<Edge>[] adj;

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

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

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

- **same as Graph (but adjacency lists of Edge objects)**
- **constructor**
- **add same Edge object to both adjacency lists**
**Minimum spanning tree API**

**Q.** How to represent the MST?

**A.** Technically, an MST is an edge-weighted graph. For convenience, we represent it as a set of edges.

```java
public class MST {
    MST(EdgeWeightedGraph G)
    edges()         // edges in MST
    weight()       // weight of MST
}
```
4.3 Minimum Spanning Trees

- introduction
- cut property
- edge-weighted graph API
  - Kruskal’s algorithm
  - Prim’s algorithm

https://algs4.cs.princeton.edu
Kruskal’s algorithm demo

Consider edges in ascending order of weight.

- Add next edge to $T$ unless doing so would create a cycle.

\[ \begin{array}{c|c}
\text{graph edges} & \text{sorted by weight} \\
\hline
0-7 & 0.16 \\
2-3 & 0.17 \\
1-7 & 0.19 \\
0-2 & 0.26 \\
5-7 & 0.28 \\
1-3 & 0.29 \\
1-5 & 0.32 \\
2-7 & 0.34 \\
4-5 & 0.35 \\
1-2 & 0.36 \\
4-7 & 0.37 \\
0-4 & 0.38 \\
6-2 & 0.40 \\
3-6 & 0.52 \\
6-0 & 0.58 \\
6-4 & 0.93 \\
\end{array} \]
In which order does Kruskal’s algorithm select edges in MST?

A. 1, 2, 4, 5, 6
B. 1, 2, 4, 5, 8
C. 1, 2, 5, 4, 8
D. 8, 2, 1, 5, 4
Kruskal’s algorithm: visualization
Kruskal’s algorithm: correctness proof

Proposition. [Kruskal 1956] Kruskal’s algorithm computes the MST.

Pf. When considering edge $e$, Kruskal’s algorithm adds it to $T$ if and only if it is in the MST.

[Case 1 $\Rightarrow$] Kruskal’s algorithm adds edge $e = v$–$w$ to $T$.
- Vertices $v$ and $w$ are in different connected components of $T$.
- Cut = set of vertices connected to $v$ in $T$.
- By construction of cut, no edge crossing cut is currently in $T$.
- No edge crossing cut has lower weight.
- Cut property $\Rightarrow$ edge $e$ is in the MST.
Kruskal’s algorithm: correctness proof

**Proposition.** [Kruskal 1956] Kruskal’s algorithm computes the MST.

**Pf.** When considering edge \( e \), Kruskal’s algorithm adds it to \( T \) if and only if it is in the MST.

[Case 2 \( \iff \)] Kruskal’s algorithm discards edge \( e = v - w \).

- From Case 1, all edges in \( T \) are in the MST.
- The MST can’t contain a cycle, so it can’t contain edge \( e \). □
Kruskal’s algorithm: implementation challenge

**Challenge.** Would adding edge $v\rightarrow w$ to $T$ create a cycle? If not, add it.

**Efficient solution.** Use the union–find data structure.

- Maintain a set for each connected component in $T$.
- If $v$ and $w$ are in same set, then adding $v\rightarrow w$ would create a cycle.
- To add $v\rightarrow w$ to $T$, merge sets containing $v$ and $w$.

Case 2: adding $v\rightarrow w$ creates a cycle  

Case 1: add $v\rightarrow w$ to $T$ and merge sets containing $v$ and $w$
public class KruskalMST
{
    private Queue<Edge> mst = new Queue<>();

    public KruskalMST(EdgeWeightedGraph G)
    {
        DirectedEdge[] edges = G.edges();
        Arrays.sort(edges);
        UF uf = new UF(G.V());

        for (int i = 0; i < G.E(); i++)
        {
            Edge e = edges[i];
            int v = e.either(), w = e.other(v);
            if (uf.find(v) != uf.find(w))
            {
                mst.enqueue(e);
                uf.union(v, w);
            }
        }
    }

    public Iterable<Edge> edges()
    {
        return mst;
    }
}
Proposition. In the worst case, Kruskal's algorithm computes the MST in an edge-weighted graph in \( \Theta(E \log E) \) time.

Pf.

- Bottlenecks are sort and union–find operations.

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SORT</strong></td>
<td>1</td>
<td>( E \log E )</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>( V - 1 )</td>
<td>log ( V )↑</td>
</tr>
<tr>
<td><strong>FIND</strong></td>
<td>( 2E )</td>
<td>log ( V )↑</td>
</tr>
</tbody>
</table>

↑ using weighted quick union

- Total.  \( \Theta(V \log V) + \Theta(E \log V) + \Theta(E \log E) \).

  dominated by \( \Theta(E \log E) \) since graph is connected
Given a graph with positive edge weights, how to find a spanning tree that minimizes the sum of the squares of the edge weights?

A. Run Kruskal’s algorithm using the original edge weights.
B. Run Kruskal's algorithm using the squares of the edge weights.
C. Run Kruskal’s algorithm using the square roots of the edge weights.
D. All of the above.

sum of squares = 4^2 + 6^2 + 5^2 + 10^2 + 11^2 + 7^2 = 347
Problem. Given an undirected graph $G$ with positive edge weights, find a spanning tree that maximizes the sum of the edge weights.

Goal. Design algorithm that takes $\Theta(E \log E)$ time in the worst case.
Gordon Gecko (Michael Douglas) evangelizing the importance of greed (in algorithm design?)
Wall Street (1986)
4.3 Minimum Spanning Trees

- introduction
- cut property
- edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm
Prim’s algorithm demo

- Start with vertex 0 and grow tree $T$.
- Repeat until $V - 1$ edges:
  - add to $T$ the min-weight edge with exactly one endpoint in $T$
In which order does Prim’s algorithm select edges in the MST? Assume it starts from vertex $s$.

A. 8, 2, 1, 4, 5
B. 8, 2, 1, 5, 4
C. 8, 2, 1, 5, 6
D. 8, 2, 3, 4, 5
Prim’s algorithm: visualization
**Proposition.** [Jarník 1930, Dijkstra 1957, Prim 1959]

Prim’s algorithm computes the MST.

**Pf.** Let $e = \text{min-weight edge with exactly one endpoint in } T$.

- Cut = set of vertices in $T$.
- Cut property $\Rightarrow$ edge $e$ is in the MST.

**Challenge.** How to efficiently find min-weight edge with exactly one endpoint in $T$?
Prim’s algorithm: lazy implementation demo

- Start with vertex 0 and grow tree $T$.
- Repeat until $V - 1$ edges:
  - add to $T$ the min-weight edge with exactly one endpoint in $T$
Prim’s algorithm: lazy implementation

**Challenge.** How to efficiently find min-weight edge with exactly one endpoint in \( T \)?

**Lazy solution.** Maintain a PQ of edges with (at least) one endpoint in \( T \).

- Key = edge; priority = weight of edge.
- **DELETE-MIN** to determine next edge \( e = v \rightarrow w \) to add to \( T \).
- If both endpoints \( v \) and \( w \) are marked (both in \( T \)), disregard.
- Otherwise, let \( w \) be the unmarked vertex (not in \( T \)):
  - add \( e \) to \( T \) and mark \( w \)
  - add to PQ any edge incident to \( w \) ← but don’t bother if other endpoint is in \( T \)

![Graph showing Prim's algorithm](image)

1-7 is min weight edge with exactly one endpoint in \( T \)

priority queue of crossing edges

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>0-2</td>
<td>0.26</td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
</tr>
<tr>
<td>0-4</td>
<td>0.38</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
</tbody>
</table>
Prim's algorithm: lazy implementation

public class LazyPrimMST
{
    private boolean[] marked; // MST vertices
    private Queue<Edge> mst; // MST edges
    private MinPQ<Edge> pq; // PQ of edges
    
    public LazyPrimMST(WeightedGraph G)
    {
        pq = new MinPQ<>();
        mst = new Queue<>();
        marked = new boolean[G.V()];
        visit(G, 0); // assume graph G is connected
    }

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

    private void visit(WeightedGraph G, int v)
    {
        marked[v] = true; // add v to tree T
        for (Edge e : G.adj(v))
            if (!marked[e.other(v)])
                pq.insert(e);
    }

    while (mst.size() < G.V() - 1)
    {
        Edge e = pq.delMin();
        int v = e.either(), w = e.other(v);
        if (marked[v] && marked[w]) continue;
        mst.enqueue(e);
        if (!marked[v]) visit(G, v);
        if (!marked[w]) visit(G, w);
    }
}

for each edge e = v–w:
    repeatedly delete the min-weight edge e = v–w from PQ
    add e to PQ if w not already in T
    ignore if both endpoints in tree T
    add edge e to tree T
    add either v or w to tree T
Lazy Prim’s algorithm: running time

**Proposition.** In the worst case, lazy Prim’s algorithm computes the MST in $\Theta(E \log E)$ time and $\Theta(E)$ extra space.

**Pf.**
- Bottlenecks are PQ operations.
- Each edge is added to PQ at most once.
- Each edge is deleted from PQ at most once.

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INSERT</strong></td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
<tr>
<td><strong>DELETE-MIN</strong></td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
</tbody>
</table>
**Prim’s algorithm: eager implementation**

**Challenge.** Find min-weight edge with exactly one endpoint in $T$.

**Observation.** For each vertex $v$, need only min-weight edge connecting $v$ to $T$.
- MST includes at most one edge connecting $v$ to $T$. Why?
- If MST includes such an edge, it must take lightest such edge. Why?

**Impact.** PQ of vertices; $\Theta(V)$ extra space; $\Theta(E \log V)$ running time in worst case.

See textbook for details
MST: algorithms of the day

<table>
<thead>
<tr>
<th>algorithm</th>
<th>visualization</th>
<th>bottleneck</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kruskal</td>
<td></td>
<td>sorting</td>
<td>$E \log E$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>union–find</td>
<td></td>
</tr>
<tr>
<td>Prim</td>
<td></td>
<td>priority queue</td>
<td>$E \log V$</td>
</tr>
</tbody>
</table>