4.3 Minimum Spanning Trees

- introduction
- cut property
- edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm
4.3 Minimum Spanning Trees

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

https://algs4.cs.princeton.edu
A motivating example

Install minimum number of paving stones to connect all of the houses.
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. Removing any edge from $T$ disconnects it.
B. Adding any edge to $T$ creates a cycle.
C. $T$ contains exactly $V - 1$ edges.
D. All of the above.
Network design

Network. Vertex = network component; edge = potential connection; edge weight = cost.

electrical, computer, telecommunication, transportation

3 Other practical applications

Other practical applications based on minimal spanning trees include:

• Taxonomy.
• Cluster analysis: clustering points in the plane, single-linkage clustering, graph-theoretic clustering, and clustering gene expression data.


Hierarchical clustering

Microarray graph. Vertex = cancer tissue; edge = all pairs; edge weight = dissimilarity.

Reference: Botstein & Brown group
More MST applications

- Image segmentation
- MST dithering
- Slime mold vs. rail network

**Figure 2:** MST dithering

**Figure 3:** Slime mold vs. rail network

**Figure 4:** Phylogeny tree reconstruction
4.3 Minimum Spanning Trees

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

For simplicity, we assume:

- The graph is connected. \(\Rightarrow\) MST exists.
- The edge weights are distinct. \(\Rightarrow\) MST is unique.

Note. Today's algorithms all work with duplicate edge weights.

but assumption simplifies the analysis

no two edge weights are equal

see Exercise 4.3.3 (solved on booksite)
**Cut property**

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

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

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

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

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

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

**Note.** A cut may have multiple edges 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)
Def. A **cut** in a graph is a partition of its vertices into two nonempty sets.

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

**Cut property.** For any cut, its min-weight crossing edge \( e \) is in the MST.

**Pf.** [by contradiction] Suppose \( e \) is not in the MST \( T \).

- Adding \( e \) to \( T \) creates a unique cycle.
- Some other edge \( f \) in cycle must also be a crossing edge.
- Removing \( f \) and adding \( e \) to \( T \) yields a different spanning tree \( T' \).
- Since \( \text{weight}(e) < \text{weight}(f) \), we have \( \text{weight}(T') < \text{weight}(T) \).
- Contradiction.  

![Diagram](attachment:image.png)

---

19
Generic algorithm (to compute MST in G)

\[ T = \emptyset. \]
Repeat until \( T \) is a spanning tree:
- Find a cut in \( G \).
- \( e \leftarrow \text{min-weight crossing edge.} \)
- \( T \leftarrow T \cup \{ e \} \).

Efficient implementations.

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

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

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

https://algs4.cs.princeton.edu
Weighted edge API

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

```java
public class Edge implements Comparable<Edge>

    Edge(int v, int w, double weight)  // create a weighted edge v–w
    int either()                        // either endpoint
    int other(int v)                    // the endpoint that's not v
    int compareTo(Edge that)           // compare edges by weight

edge e = v–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);
    }
}
**API.** Same as Graph and Digraph, except with explicit Edge objects.

```java
public class EdgeWeightedGraph

    EdgeWeightedGraph(int V)  // edge-weighted graph with V vertices (and no edges)

    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.

```
adj[]
V
adj[0] <- 6 0 .58 -> 0 1 .26 -> 0 4 .38 -> 0 7 .16
adj[1] <- 1 3 .29 -> 1 2 .36 -> 1 7 .19 -> 1 5 .32
adj[2] <- 6 2 .40 -> 2 7 .34 -> 1 2 .36 -> 0 2 .26 -> 2 3 .17
adj[3] <- 3 6 .52 -> 1 3 .29 -> 2 3 .17
adj[5] <- 1 5 .32 -> 5 7 .28 -> 4 5 .35
adj[6] <- 6 4 .93 -> 6 0 .58 -> 3 6 .52 -> 6 2 .40
adj[7] <- 2 7 .34 -> 1 7 .19 -> 0 7 .16 -> 5 7 .28 -> 4 7 .37
```
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];
    }
}
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) constructor

Iterable<Edge> edges() edges in MST

double 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.

graph edges sorted by weight

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.16</td>
</tr>
<tr>
<td>2-3</td>
<td>0.17</td>
</tr>
<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>1-3</td>
<td>0.29</td>
</tr>
<tr>
<td>1-5</td>
<td>0.32</td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>4-5</td>
<td>0.35</td>
</tr>
<tr>
<td>1-2</td>
<td>0.36</td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
</tr>
<tr>
<td>0-4</td>
<td>0.38</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>6-4</td>
<td>0.93</td>
</tr>
</tbody>
</table>

an edge–weighted graph
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: correctness proof

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

**Pf.** Kruskal’s algorithm adds edge $e$ to $T$ if and only if $e$ 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, $e$ is a crossing edge and no crossing edge
  - is currently in $T$
  - was considered by Kruskal before $e$
- Thus, $e$ is a min weight crossing edge.
- Cut property $\Rightarrow e$ is in the MST.
Kruskal’s algorithm: correctness proof

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

**Pf.** Kruskal’s algorithm adds edge $e$ to $T$ if and only if $e$ is in the MST.

[Case 2 ⇐] Kruskal’s algorithm discards edge $e = v – w$.

- From Case 1, all edges currently in $T$ are in the MST.
- The MST can’t contain a cycle, so it can’t also contain $e$.  

```
adding edge to tree would create a cycle
```
Challenge. Would adding edge $v-w$ 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 $T$, initially each vertex in its own set.
- If $v$ and $w$ are in same set, then adding $v-w$ to $T$ would create a cycle. [Case 2]
- Otherwise, add $v-w$ to $T$ and merge sets containing $v$ and $w$. [Case 1]
Kruskal’s algorithm: Java implementation

```java
public class KruskalMST {
    private Queue<Edge> mst = new Queue<>();

    public KruskalMST(EdgeWeightedGraph G) {
        Edge[] 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;
    }
}
```

- **edges in the MST**
- **sort edges by weight**
- **maintain connected components**
- **optimization: stop as soon as V–1 edges in T**
- **greedily add edges to MST**
- **edge v–w does not create cycle**
- **add edge e to MST**
- **merge connected components**
**Proposition.** In the worst case, Kruskal’s algorithm computes the MST in an edge-weighted graph in $\Theta(E \log E)$ time and $\Theta(E)$ extra space.

**Pf.**

- Bottlenecks are sorting 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>$\log V ^\dagger$</td>
</tr>
<tr>
<td><strong>FIND</strong></td>
<td>$2E$</td>
<td>$\log V ^\dagger$</td>
</tr>
</tbody>
</table>

† using weighted quick union

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

\[ \text{dominated by } \Theta(E \log E) \]

\[ \text{since graph is connected} \]
Minimum spanning trees: quiz 4

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.
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 $T$

```
  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
```
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
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 $T$
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 $w$  

\[\begin{array}{c|c}
1 & 5 \\
2 & 6 \\
3 & 7 \\
4 & 0 \\
5 & 1 \\
6 & 7 \\
7 & 0 \\
\end{array}\]

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>0-2</td>
<td>0.26</td>
</tr>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>3-7</td>
<td>0.28</td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

*but don’t bother if other endpoint is already in $T$*
Prim’s algorithm: lazy implementation

```java
class LazyPrimMST {
  private boolean[] marked; // MST vertices
  private Queue<WeightedGraph.Edge> mst; // MST edges
  private MinPQ<WeightedGraph.Edge> pq; // PQ of edges

  LazyPrimMST(WeightedGraph G) {
    pq = new MinPQ<>();
    mst = new Queue<>();
    marked = new boolean[G.V()];
    visit(G, 0); // assume graph G is connected
  }

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

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

Repeatedly delete the min-weight edge \(e = v \rightarrow w\) from \(PQ\)
- ignore if both endpoints in tree \(T\)
- add edge \(e\) to tree \(T\)
- add either \(v\) or \(w\) to tree \(T\)
- repeatedly delete the min-weight edge \(e = v \rightarrow w\) from \(PQ\)

For each edge \(e = v \rightarrow w\):
- add \(e\) to \(PQ\) if \(w\) not already in \(T\)
- add \(v\) to tree \(T\)
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>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>$E$</td>
<td>$\log E$†</td>
</tr>
<tr>
<td>DELETE–MIN</td>
<td>$E$</td>
<td>$\log E$†</td>
</tr>
</tbody>
</table>

† using binary heap
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><img src="image1.png" alt="Kruskal visualization" /></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><img src="image2.png" alt="Prim visualization" /></td>
<td>priority queue</td>
<td>$E \log V$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>source</td>
<td>license</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>Muddy City Problem</td>
<td>CS Unplugged</td>
<td>CC BY-NC-SA 4.0</td>
<td></td>
</tr>
<tr>
<td>Microarrays and Clustering</td>
<td>Botstein and Brown</td>
<td>by author</td>
<td></td>
</tr>
<tr>
<td>Image Segmentation</td>
<td>Felzenszwalb and Huttenlocher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phylogeny Tree</td>
<td>Derzelle et al.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MST Dithering</td>
<td>Mario Klingemann</td>
<td>CC BY-NC 2.0</td>
<td></td>
</tr>
<tr>
<td>Slime Mold vs. Rail Network</td>
<td>Harvard Magazine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mona Singh</td>
<td>Princeton University</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A final thought

“The algorithms we write are only as good as the questions we ask. And the best questions come from collaboration and creative thinking.” — Mona Singh