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
**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.
**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.
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.

![Diagram of a non-acyclic spanning tree](image-url)
**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.
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.

Minimum spanning tree $T$
(weight = 50 = 4 + 6 + 5 + 8 + 9 + 11 + 7)

**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.

spanning tree $T$ of graph $G$
Network design

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

Reference: Botstein & Brown group
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 \textbf{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
Simplifying assumptions

For simplicity, we assume:

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

Note. Algorithms still work even if duplicate edge weights.

![Diagram of a graph with edge weights](image)
**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.
Minimum spanning trees: quiz 2

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 weight of $e$ is less than the weight of $f$, the weight of $T'$ is less than the weight of $T$.
- Contradiction.

\[\text{the MST } T \text{ does not contain } e\]

\[\text{adding } e \text{ to MST creates a unique cycle}\]
Framework for minimum spanning tree algorithm

**Generic algorithm (to compute MST)**

\[ T = \emptyset. \]
Repeat until \( T \) is a spanning tree: \( V - 1 \) edges
- 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

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
    }
}
```

Idiom for processing an edge e. int v = e.either(), w = e.other(v).
Weighted edge: Java implementation

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.
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<Edge>();
    }

    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.

```
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
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: visualization
Kruskal’s algorithm: correctness proof

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

**Pf.** Let $T$ be tree at some point during execution, and let $e$ be next edge.

- **[Case 1]** 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. Why?
  - Cut property $\Rightarrow$ edge $e$ is in the MST.
Kruskal’s algorithm: correctness proof

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

**Pf.** Let $T$ be tree at some point during execution, and let $e$ be next edge.

[Case 2] Kruskal’s algorithm discards edge $e = v\rightarrow w$.
- From Case 1, all edges in $T$ are in the MST.
- The MST can’t contain a cycle. □
Kruskal’s algorithm: implementation challenge

Challenge. Would adding edge \( v-w \) to tree \( T \) 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 \( v-w \) would create a cycle.
- To add \( v-w \) to \( T \), merge sets containing \( v \) and \( w \).

Case 1: add \( v-w \) to \( T \) and merge sets containing \( v \) and \( w \)

Case 2: adding \( v-w \) creates a cycle
Kruskal’s algorithm: Java implementation

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

    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;
    }
}
```
Kruskal’s algorithm: running time

**Proposition.** In the worst case, Kruskal’s algorithm computes MST in time proportional to $E \log E$.

**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>Sort</td>
<td>1</td>
<td>$E \log E$</td>
</tr>
<tr>
<td>Union</td>
<td>$V - 1$</td>
<td>$\log V^\dagger$</td>
</tr>
<tr>
<td>Find</td>
<td>$2E$</td>
<td>$\log V^\dagger$</td>
</tr>
</tbody>
</table>

$\dagger$ using weighted quick union

- **Total.** $V \log V + E \log V + E \log E$. 
  
  dominated by $E \log E$ 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.

![Graph Image]

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 with $E \log E$ running time.

maximum spanning tree $T$ (weight = 104)
Greed is good

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 \( T \)

an edge-weighted graph

<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>
In which order does Prim’s algorithm select edges in the MST? Assume it starts from vertex s.

A. 1, 2, 3, 4, 5
B. 1, 2, 5, 4, 8
C. 8, 2, 1, 5, 4
D. 8, 5, 4, 2, 1
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 $T$.
- No crossing edge is in $T$.
- No crossing edge has lower weight.
- Cut property $\Rightarrow$ edge $e$ is in the MST. □
Prim’s algorithm: lazy implementation

**Challenge.** Find the 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 $e = v–w$ 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 $w$ (assuming other endpoint not in $T$)

![Diagram](attachment:image.png)

1-7 is min weight edge with exactly one endpoint in $T$

priority queue of crossing edges

1-7 0.19
0-2 0.26
5-7 0.28
2-7 0.34
4-7 0.37
0-4 0.38
6-0 0.58
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

```java
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<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);

        while (!pq.isEmpty() && 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);
        }
    }
}
```

assume graph $G$ is connected

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 $T$
Prim’s algorithm: lazy implementation

```java
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<Edge> mst() {
    return mst;
}
```

- add vertex \(v\) to tree \(T\)
- for each edge \(e = v \rightarrow w\):
  - add \(e\) to PQ if \(w\) not already in \(T\)
Lazy Prim’s algorithm: running time

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

**Pf.**

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

*can be improved to $V$ with “eager” version*

<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; extra space at most $V$; running time $E \log V$. 

Prim's algorithm: eager implementation

![Diagram of a graph with vertices and edges indicating the Prim's algorithm process.](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="image" alt="Kruskal Visualization" /></td>
<td>sorting union–find</td>
<td>$E \log E$</td>
</tr>
<tr>
<td>Prim</td>
<td><img src="image" alt="Prim Visualization" /></td>
<td>priority queue</td>
<td>$E \log V$</td>
</tr>
</tbody>
</table>

- **Kruskal**: An algorithm for finding a minimum spanning tree (MST) in a weighted, connected, undirected graph. It uses a greedy approach and can be implemented in $E \log E$ time.
- **Prim**: Another algorithm for finding an MST, Prim's algorithm also uses a greedy approach and runs in $E \log V$ time.