4.3 Minimum Spanning Trees

- introduction
- greedy algorithm
- edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- context
4.3 Minimum Spanning Trees

- introduction
- greedy algorithm
- edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- context
Spanning tree

**Def.** A **spanning tree** of an undirected graph $G$ is a subgraph $T$ that is:

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

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

Def. A spanning tree of an undirected graph $G$ is a subgraph $T$ that is:

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

not a tree (cyclic)
Def. A spanning tree of an undirected graph $G$ is a subgraph $T$ that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.
Quiz 1: spanning trees

Let $T$ be a spanning tree of a connected graph $G$ with $V$ vertices. Which of the following statements are true?

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.

E. I don't know.
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? (Impractical.)
Network design

MST of bicycle routes in North Seattle

http://www.flickr.com/photos/ewedistrict/21980840
Models of nature

MST of random graph

http://algo.inria.fr/broutin/gallery.html
Rules for Biologically Inspired Adaptive Network Design

Atsushi Tero, Seiji Takagi, Tetsu Saigusa, Kentaro Ito, Dan P. Bebber, Mark D. Fricker, Kenji Yumiki, Ryo Kobayashi, Toshiyuki Nakagaki

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

MST dithering

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

MST is fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- 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.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).

4.3 Minimum Spanning Trees

- introduction
- greedy algorithm
- edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- context
Simplifying assumptions

For simplicity, we assume

- The graph is connected. ⇒ MST exists.
- The edge weights are distinct. ⇒ MST is unique.
Cut property

**Def.** A cut in a graph is a partition of its vertices into two (nonempty) sets. **Def.** A crossing edge connects a vertex in one set with a vertex in the other.
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)
E. I don't know.
Cut property

Def. A cut in a graph is a partition of its vertices into two (nonempty) sets. 
Def. A crossing edge connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is 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 connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.
Pf. Suppose min-weight crossing edge \( e \) is not in the MST.
   • Adding \( e \) to the MST creates a cycle.
   • Some other edge \( f \) in cycle must be a crossing edge.
   • Removing \( f \) and adding \( e \) is also a spanning tree.
   • Since weight of \( e \) is less than the weight of \( f \), that spanning tree has lower weight.
   • Contradiction. □
Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V-1$ edges are colored black.

an edge-weighted graph

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0.19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0.26</td>
<td></td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.19</td>
<td></td>
<td>0.28</td>
<td></td>
<td>0.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0.26</td>
<td>0.34</td>
<td>0.35</td>
<td></td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.16</td>
<td>0.28</td>
<td>0.34</td>
<td>0.35</td>
<td>0.38</td>
<td>0.40</td>
<td>0.58</td>
</tr>
<tr>
<td>7</td>
<td>0.17</td>
<td>0.26</td>
<td>0.34</td>
<td>0.35</td>
<td>0.38</td>
<td>0.40</td>
<td>0.58</td>
</tr>
</tbody>
</table>
Removing two simplifying assumptions

Q. What if edge weights are not all distinct?
A. Greedy MST algorithm correct even if equal weights are present!

Q. What if graph is not connected?
A. Compute minimum spanning forest = one MST per component.
Greedy MST algorithm: efficient implementations

**In practice:** How to find cut? How to find min-weight edge?

**Ex 1.** Kruskal's algorithm. [stay tuned]

**Ex 2.** Prim's algorithm. [stay tuned]

**Ex 3.** Borůvka's algorithm.
4.3 Minimum Spanning Trees

- Introduction
- Greedy algorithm
- Edge-weighted graph API
- Kruskal’s algorithm
- Prim’s algorithm
- Context
Weighted edge API

Edge abstraction needed 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 this edge to that edge

    double weight()  // the weight

    String toString()  // string representation
```

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) {
        if (this.weight < that.weight) return -1;
        else if (this.weight > that.weight) return +1;
        else return 0;
    }
}
## Edge-weighted graph API

```java
public class EdgeWeightedGraph
{
    EdgeWeightedGraph(int V) {
        // create an empty graph with V vertices
    }
    EdgeWeightedGraph(In in) {
        // create a graph from input stream
    }
    void addEdge(Edge e) {
        // add weighted edge e to this graph
    }
    Iterable<Edge> adj(int v) {
        // edges incident to v
    }
    Iterable<Edge> edges() {
        // all edges in this graph
    }
    int V() {
        // number of vertices
    }
    int E() {
        // number of edges
    }
    String toString() {
        // string representation
    }
}
```

### Conventions.
Allow self-loops and parallel edges.
Edge-weighted graph: adjacency-lists 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<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Bag<Ba...
Minimum spanning tree API

Q. How to represent the MST?

```
public class MST

MST(EdgeWeightedGraph G) constructor

Iterable<Edge> edges() edges in MST

double weight() weight of MST
```
4.3 Minimum Spanning Trees

- introduction
- greedy algorithm
- edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- context
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

![an edge-weighted graph]

<table>
<thead>
<tr>
<th>Weight</th>
<th>Value</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>
Kruskal's algorithm: visualization
Kruskal's algorithm: correctness proof

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

Pf. Kruskal's algorithm is a special case of the greedy MST algorithm.
   - Suppose Kruskal's algorithm colors the edge $e = v-w$ black.
   - Cut = set of vertices connected to $v$ in tree $T$.
   - No crossing edge is black.
   - No crossing edge has lower weight. Why?
Challenge. Would adding edge $v \rightarrow w$ to tree $T$ create a cycle? If not, add it.

How difficult to implement?

A. $E + V$

B. $V$

C. $\log V$ (or $\log^* V$)

D. $\log E$ (or $\log^* E$)

E. 1
Kruskal's algorithm: implementation challenge

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

Case 1: adding \( v \rightarrow w \) creates a cycle
Case 2: add \( v \rightarrow w \) to \( T \) and merge sets containing \( v \) and \( w \)
public class KruskalMST
{
    private Queue<Edge> mst = new Queue<Edge>();

    public KruskalMST(EdgeWeightedGraph G)
    {
        MinPQ<Edge> pq = new MinPQ<Edge>(G.edges());
        UF uf = new UF(G.V());
        while (!pq.isEmpty() && mst.size() < G.V()-1)
        {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (!uf.connected(v, w))
            {
                uf.union(v, w);
                mst.enqueue(e);
            }
        }
    }

    public Iterable<Edge> edges()
    {
        return mst;
    }
}
Kruskal's algorithm: running time

**Proposition.** Kruskal's algorithm computes MST in time proportional to $E \log E$ (in the worst case).

**Pf.**

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>build pq</td>
<td>1</td>
<td>$E$</td>
</tr>
<tr>
<td>delete–min</td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
<tr>
<td>union</td>
<td>$V$</td>
<td>$\log^* V$</td>
</tr>
<tr>
<td>connected</td>
<td>$E$</td>
<td>$\log^* V$</td>
</tr>
</tbody>
</table>

† amortized bound using weighted quick union with path compression

often called fewer than $E$ times
4.3 Minimum Spanning Trees

- introduction
- greedy algorithm
- edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- context
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

**an edge-weighted graph**
Prim’s algorithm: visualization
Proposition. [Jarník 1930, Dijkstra 1957, Prim 1959]
Prim's algorithm computes the MST.

**Pf.** Prim's algorithm is a special case of the greedy MST algorithm.
- Suppose edge $e = \min$ weight edge connecting a vertex on the tree to a vertex not on the tree.
- Cut = set of vertices connected on tree.
- No crossing edge is black.
- No crossing edge has lower weight.

`edge e = 7-5 added to tree`
Prim's algorithm: implementation challenge

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

How difficult?

A. $E$

B. $V$

C. $\log E$

D. 1

E. I don't know.
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 \( T \).
- Disregard if both endpoints \( v \) and \( w \) are marked (both in \( T \)).
- Otherwise, let \( w \) be the unmarked vertex (not in \( T \)):
  - add \( e \) to \( T \) and mark \( w \)
  - add to PQ all edges incident to \( w \) (assuming other endpoint not in \( T \))
Prim's algorithm: lazy implementation demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

![Diagram of an edge-weighted graph](attachment:image.png)

<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>
Prim's algorithm: lazy implementation

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

    public LazyPrimMST(WeightedGraph G) {
        pq = new MinPQ<WeightedEdge>();
        mst = new Queue<WeightedEdge>();
        marked = new boolean[G.V()];
        visit(G, 0);

        while (!pq.isEmpty() && mst.size() < G.V() - 1) {
            WeightedEdge 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 G is connected
- Repeatedly delete the min weight edge $e = v-w$ from PQ
- Ignore if both endpoints in T
- Add edge $e$ to tree
- Add either $v$ or $w$ to tree
Prim's algorithm: lazy implementation

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 v to T
for each edge e = v–w, add to PQ if w not already in T
Lazy Prim's algorithm: running time

**Proposition.** Lazy Prim's algorithm computes the MST in time proportional to \( E \log E \) and extra space proportional to \( E \) (in the worst case).

**Pf.**

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete min</td>
<td>( E )</td>
<td>( \log E )</td>
</tr>
<tr>
<td>insert</td>
<td>( E )</td>
<td>( \log E )</td>
</tr>
</tbody>
</table>
Challenge. Find min weight edge with exactly one endpoint in $T$.

Observation. For each vertex $v$, need only lightest 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?
Prim's algorithm: eager implementation demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

an edge-weighted graph
Prim's algorithm: eager implementation demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

\[
\begin{array}{c|c|c}
\text{v} & \text{edgeTo[]} & \text{distTo[]} \\
\hline
0 & - & - \\
7 & 0-7 & 0.16 \\
1 & 1-7 & 0.19 \\
2 & 0-2 & 0.26 \\
3 & 2-3 & 0.17 \\
5 & 5-7 & 0.28 \\
4 & 4-5 & 0.35 \\
6 & 6-2 & 0.40 \\
\end{array}
\]

**MST edges**

0–7  1–7  0–2  2–3  5–7  4–5  6–2
Prim's algorithm: eager implementation

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

**Eager solution.** Maintain a PQ of vertices connected by an edge to $T$, where priority of vertex $v =$ weight of lightest edge connecting $v$ to $T$.

- Delete min vertex $v$ and add its associated edge $e = v-w$ to $T$.
- Update PQ by considering all edges $e = v-x$ incident to $v$
  - ignore if $x$ is already in $T$
  - add $x$ to PQ if not already on it
  - **decrease priority** of $x$ if $v-x$ becomes lightest edge connecting $x$ to $T$

```
black: on MST
red: on PQ
```

PQ has at most one entry per vertex
Indexed priority queue

Associate an index between 0 and $N - 1$ with each key in a priority queue.

- Insert a key associated with a given index.
- Delete a minimum key and return associated index.
- Decrease the key associated with a given index.

```java
public class IndexMinPQ<Key extends Comparable<Key>> {
    IndexMinPQ(int N)
    void insert(int i, Key key)
    void decreaseKey(int i, Key key)
    int delMin()
    boolean contains(int i)
    boolean isEmpty()
    int size()
}
```

create indexed priority queue with indices 0, 1, ..., $N - 1$
associate key with index $i$
remove a minimal key and return its associated index
decrease the key associated with index $i$
is $i$ an index on the priority queue?
is the priority queue empty?
number of keys in the priority queue

for Prim's algorithm, $N = V$ and index = vertex.
Indexed priority queue: implementation

Binary heap implementation. [see Section 2.4 of textbook]

- Start with same code as MinPQ.
- Maintain parallel arrays so that:
  - keys[i] is the priority of vertex i
  - qp[i] is the heap position of vertex i
  - pq[i] is the index of the key in heap position i
- Use swim(qp[i]) to implement decreaseKey(i, key).

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys[i]</td>
<td>A</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>I</td>
<td>N</td>
<td>G</td>
<td>-</td>
</tr>
<tr>
<td>qp[i]</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>pq[i]</td>
<td>-</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>-</td>
</tr>
</tbody>
</table>

Decrease key of vertex 2 to C

Vertex 2 is at heap index 4
Prim's algorithm: which priority queue?

Depends on PQ implementation: \( V \) insert, \( V \) delete-min, \( E \) decrease-key.

<table>
<thead>
<tr>
<th>PQ implementation</th>
<th>insert</th>
<th>delete-min</th>
<th>decrease-key</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>1</td>
<td>( V )</td>
<td>1</td>
<td>( V^2 )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( \log V )</td>
<td>( \log V )</td>
<td>( \log V )</td>
<td>( E \log V )</td>
</tr>
<tr>
<td>d-way heap</td>
<td>( \log_d V )</td>
<td>( d \log_d V )</td>
<td>( \log_d V )</td>
<td>( E \log_{E/V} V )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( 1^† )</td>
<td>( \log V^† )</td>
<td>( 1^† )</td>
<td>( E + V \log V )</td>
</tr>
</tbody>
</table>

† amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.
4.3 Minimum Spanning Trees

- introduction
- greedy algorithm
- edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- context
Does a linear-time MST algorithm exist?

deterministic compare-based MST algorithms

<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>$E \log V$</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>

**Remark.** Linear-time randomized MST algorithm (Karger-Klein-Tarjan 1995).
Euclidean MST

Given $N$ points in the plane, find MST connecting them, where the distances between point pairs are their Euclidean distances.

**Brute force.** Compute $\sim N^2/2$ distances and run Prim's algorithm.

**Ingenuity.** Exploit geometry and do it in $N \log N$ time.
Problem. Given an edge-weighted graph $G$, find a spanning tree that maximizes the sum of the edge weights.

Running time. $E \log E$ (or better).
Problem. Given an edge-weighted graph $G$, find a spanning tree that minimizes the sum of the squares of its edge weights.

Running time. $E \log E$ (or better).
**Minimum Bottleneck Spanning Tree**

**Problem.** Given an edge-weighted graph $G$, find a spanning tree that minimizes the maximum weight of its edges.

**Running time.** $E \log E$ (or better).

Note: need to be a MST

minimum bottleneck spanning tree $T$ (bottleneck = 9)
Solution. Compute a MST; it is a MBST.

Pf. Suppose MST is not a MBST.

• Let $e$ = edge in MST with weight strictly larger than bottleneck weight.
• Consider cut formed by deleting $e$ from MST.
• MBST contains at least one edge $f$ crossing cut.
• Adding $f$ to MST and deleting $e$ yields better MST.
Scientific application: clustering

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

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

**Goal.** Divide into clusters so that objects in different clusters are far apart.

Applications.

- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases.
- Skycat: cluster $10^9$ sky objects into stars, quasars, galaxies.
**Single-link clustering**

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

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

**Single link.** Distance between two clusters equals the distance between the two closest objects (one in each cluster).

**Single-link clustering.** Given an integer $k$, find a $k$-clustering that maximizes the distance between two closest clusters.
Single-link clustering algorithm

“Well-known” algorithm in science literature 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 merge the two clusters.
- Repeat until there are exactly \( k \) clusters.

**Observation.** This is Kruskal's algorithm. (stopping when \( k \) connected components)

**Alternate solution.** Run Prim; then delete \( k - 1 \) max weight edges.
Dendrogram of cancers in human

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group