Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.
Minimum Spanning Trees
- Greedy algorithm
- Edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- Context
Minimum spanning tree

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

**Def.** A **spanning tree** of $G$ is a subgraph $T$ that is connected and acyclic.

**Goal.** Find a min weight spanning tree.

A subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.
Minimum spanning tree

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

**Def.** A spanning tree of $G$ is a subgraph $T$ that is connected and acyclic.

**Goal.** Find a min weight spanning tree.
**Minimum spanning tree**

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

**Def.** A spanning tree of $G$ is a subgraph $T$ that is connected and acyclic.

**Goal.** Find a min weight spanning tree.
Given. Undirected graph $G$ with positive edge weights (connected).

Def. A spanning tree of $G$ is a subgraph $T$ that is connected and acyclic.

Goal. Find a min weight spanning tree.

Brute force. Try all spanning trees?

spanning tree $T$: cost $= 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$
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, cable, computer, road).

Minimum Spanning Trees

- Greedy algorithm
- Edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- Context
**Simplifying assumptions.** Edge weights are distinct; graph is connected.

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

Simplifying assumptions. Edge weights are distinct; graph is connected.

Def. A cut in a graph is a partition of its vertices into two (nonempty) sets. 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. Let $e$ be the min-weight crossing edge in cut.
- Suppose $e$ 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 is lower weight.
- Contradiction. □
Greedy MST algorithm

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

![Diagram of an edge-weighted graph with weights](image)
Greedy MST algorithm

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

![Diagram of a graph with labeled edges and vertices. The edges in red represent the crossing edges, and the vertices in gray form one side of the cut. The weights of the edges in the Minimum Spanning Tree (MST) are also shown.]
Greedy MST algorithm

- Start with all edges colored gray.
- Find a cut with no black crossing edges, and color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.
Greedy MST algorithm

- Start with all edges colored gray.
- Find a cut with no black crossing edges, and color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.
Greedy MST algorithm

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

MST edges

0–2  5–7
Greedy MST algorithm

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

**MST edges**

0–2  5–7

crossing edges (sorted by weight)

in MST

6–2  0.40
3–6  0.52
6–0  0.58
6–4  0.93

min-weight crossing edge
Greedy MST algorithm

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

MST edges

0–2  5–7  6–2
Greedy MST algorithm

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

**MST edges**

- 0–2
- 5–7
- 6–2

**Crossing edges (sorted by weight)**

- 0–7 with weight 0.16
- 2–3 with weight 0.17
- 2–7 with weight 0.34
- 4–5 with weight 0.35
- 1–2 with weight 0.36
- 4–7 with weight 0.37
- 3–6 with weight 0.52
Greedy MST algorithm

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

MST edges

0-2  5-7  6-2  0-7
Greedy MST algorithm

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

**MST edges**

0–2  5–7  6–2  0–7

**crossing edges (sorted by weight)**

- 2–3  0.17
- 1–7  0.19
- 1–5  0.32
- 1–2  0.36

**Min-weight crossing edge**
Greedy MST algorithm

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

MST edges

0–2  5–7  6–2  0–7  2–3
Greedy MST algorithm

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

MST edges

0–2  5–7  6–2  0–7  2–3
Greedy MST algorithm

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

MST edges

0–2  5–7  6–2  0–7  2–3  1–7
Greedy MST algorithm

- Start with all edges colored gray.
- Find a cut with no black crossing edges, and color its min-weight edge black.
- Repeat until \( V - 1 \) edges are colored black.

MST edges

\[
0-2 \quad 5-7 \quad 6-2 \quad 0-7 \quad 2-3 \quad 1-7
\]
Greedy MST algorithm

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

**MST edges**

$0-2$, $5-7$, $6-2$, $0-7$, $2-3$, $1-7$, $4-5$
Proposition. The greedy algorithm computes the MST.

Pf.
- Any edge colored black is in the MST (via cut property).
- If fewer than \( V - 1 \) black edges, there exists a cut with no black crossing edges. (consider cut whose vertices are one connected component)
Proposition. The greedy algorithm computes the MST:

Efficient implementations. Choose cut? Find min-weight edge?
Ex 1. Kruskal's algorithm. [stay tuned]
Ex 2. Prim's algorithm. [stay tuned]
Ex 3. Borůvka's algorithm.
**Removing two simplifying assumptions**

**Q.** What if edge weights are not all distinct?

**A.** Greedy MST algorithm still correct if equal weights are present! (our correctness proof fails, but that can be fixed)

**Q.** What if graph is not connected?

**A.** Compute minimum spanning forest = MST of each component.

---

**Various MST anomalies**

- Weights can be 0 or negative.
- MST may not be unique when weights have equal values.
- MST may not exist when graph is not connected.
- Weights need not be proportional to distance.
Minimum Spanning Trees

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

Edge abstraction needed for weighted edges.

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;
    }
}

Weighted edge: Java implementation

- Constructor
- Either endpoint
- Other endpoint
- Compare edges by weight
## Edge-weighted graph API

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class <code>EdgeWeightedGraph</code></td>
<td></td>
</tr>
<tr>
<td><code>EdgeWeightedGraph(int V)</code></td>
<td>create an empty graph with V vertices</td>
</tr>
<tr>
<td><code>EdgeWeightedGraph(In in)</code></td>
<td>create a graph from input stream</td>
</tr>
<tr>
<td><code>void addEdge(Edge e)</code></td>
<td>add weighted edge e to this graph</td>
</tr>
<tr>
<td><code>Iterable&lt;Edge&gt; adj(int v)</code></td>
<td>edges incident to v</td>
</tr>
<tr>
<td><code>Iterable&lt;Edge&gt; edges()</code></td>
<td>all edges in this graph</td>
</tr>
<tr>
<td><code>int V()</code></td>
<td>number of vertices</td>
</tr>
<tr>
<td><code>int E()</code></td>
<td>number of edges</td>
</tr>
<tr>
<td><code>String toString()</code></td>
<td>string representation</td>
</tr>
</tbody>
</table>

### Conventions
Allow self-loops and parallel edges.
Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of Edge lists.

```
adj[] = 8
 0 1 2 3 4 5 6 7
0 5 0.35 4 7 0.37 5 7 0.28 0 7 0.16 1 5 0.32 0 4 0.38 2 3 0.17 1 7 0.19 0 2 0.26 1 2 0.36 1 3 0.29 2 7 0.34 6 2 0.40 3 6 0.52 6 0 0.58 6 4 0.93
1 3 0.29 1 2 0.36 1 7 0.19 1 5 0.32 6 2 0.40 2 7 0.34 1 2 0.36 0 2 0.26 2 3 0.17 3 6 0.52 1 3 0.29 2 3 0.17 6 4 0.93 0 4 0.38 4 7 0.37 4 5 0.35 1 5 0.32 5 7 0.28 4 5 0.35 6 4 0.93 6 0 0.58 3 6 0.52 6 2 0.40 2 7 0.34 1 7 0.19 0 7 0.16 5 7 0.28 5 7 0.28
```
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];
    }
}
Q. How to represent the MST?

```java
public class MST {
    MST(EdgeWeightedGraph G) { /* constructor */
    }
    Iterable<Edge> edges() { /* edges in MST */
    }
    double weight() { /* weight of MST */
    }
}
```

tinyEWG.txt

```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```
Q. How to represent the MST?

```java
public class MST
{
  MST(EdgeWeightedGraph G) constructor
  Iterable<Edge> edges() edges in MST
  double weight() weight of MST
}

class Main
{
  public static void main(String[] args)
  {
    In in = new In(args[0]);
    EdgeWeightedGraph G = new EdgeWeightedGraph(in);
    MST mst = new MST(G);
    for (Edge e : mst.edges())
      StdOut.println(e);
    StdOut.printf("%.2f\n", mst.weight());
  }
}

% java MST tinyEWG.txt
  0-7 0.16
  1-7 0.19
  0-2 0.26
  2-3 0.17
  5-7 0.28
  4-5 0.35
  6-2 0.40
  1.81
```
Minimum Spanning Trees

- Greedy algorithm
- Edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- Context
Kruskal's algorithm

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

an edge-weighted graph

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>
**Kruskal's algorithm**

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

Does not create a cycle
Kruskal's algorithm

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

![Graph with edges and weights]

- Edge 0-7 in MST with weight 0.16
- Edge 2-3 in MST with weight 0.17

does not create a cycle
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

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

![Graph with edges and weights](image)

- 0-7 0.16
- 2-3 0.17
- 1-7 0.19
- 0-2 0.26
- 5-7 0.28

in MST

does not create a cycle
Kruskal's algorithm

• Consider edges in ascending order of weight.
• Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

- Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm

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

![Graph with edges and weights]
Kruskal's algorithm: visualization
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 $e = v \rightarrow w$ black.
- Cut = set of vertices connected to $v$ in tree $T$.
- No crossing edge is black.
- No crossing edge has lower weight. Why?

![Diagram](image-url)
**Challenge.** Would adding edge $v \rightarrow w$ to tree $T$ create a cycle? If not, add it.

**How difficult?**

- $E + V$
- $V$
- $\log V$
- $\log^* V$
- 1

---

**Kruskal's algorithm: implementation challenge**

run DFS from $v$, check if $w$ is reachable

(T has at most $V - 1$ edges)

use the union-find data structure!

($\log^*$ function: number of times needed to take the lg of a number until reaching 1)
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: adding $v-w$ creates a cycle

Case 2: add $v-w$ to $T$ and merge sets containing $v$ and $w$
Kruskal's algorithm: Java implementation

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

    public KruskalMST(EdgeWeightedGraph G) {
        MinPQ<Edge> pq = new MinPQ<Edge>();
        for (Edge e : G.edges())
            pq.insert(e);

        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;
    }
}
```

- build priority queue
- greedily add edges to MST
- edge v–w does not create cycle
- merge sets
- add edge to MST
**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

**Remark.** If edges are already sorted, order of growth is $E \log^* V$. 

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

- Greedy algorithm
- Edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- Context
Prim's algorithm

- 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.
• 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.
Prim's algorithm

• 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.
Prim's algorithm

• 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.

MST edges

0–7
Prim's algorithm

- 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.

**MST edges**

0–7
Prim's algorithm

- 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.

**MST edges**

$0-7, 1-7$
**Prim's algorithm**

- 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 Prim's algorithm](attachment:prim_diagram.png)

**MST edges**

- 0–7
- 1–7
Prim's algorithm

- 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.
Prim's algorithm

- 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.

MST edges

0–7  1–7  0–2
Prim's algorithm

- 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.

MST edges

0–7  1–7  0–2  2–3
Prim's algorithm

- 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.

MST edges

0–7  1–7  0–2  2–3

edges with exactly one endpoint in T (sorted by weight)

5–7  0.28
1–5  0.32
4–7  0.37
0–4  0.38
6–2  0.40
3–6  0.52
6–0  0.58
Prim's algorithm

- 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.

**MST edges**

0–7  1–7  0–2  2–3  5–7
Prim's algorithm

- 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.

MST edges

0–7  1–7  0–2  2–3  5–7
Prim's algorithm

- 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.

MST edges

- 0-7
- 1-7
- 0-2
- 2-3
- 5-7
- 4-5
Priming's algorithm

- 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.

MST edges

$0-7$  $1-7$  $0-2$  $2-3$  $5-7$  $4-5$
• 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.
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 $e = \text{min weight edge connecting a vertex on the tree to a vertex not on the tree}$. 
- 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?**
- \( E \)  
- \( V \)
- \( \log E \)
- \( \log^* E \)
- \( l \)

```
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
```

1-7 is min weight edge with exactly one endpoint in \( T \)
**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 $e = v\rightarrow w$ to add to $T$.
- Disregard if both endpoints $v$ and $w$ are in $T$.
- Otherwise, let $v$ be vertex not in $T$:
  - add to PQ any edge incident to $v$ (assuming other endpoint not in $T$)
  - add $v$ to $T$
Prim's algorithm - Lazy implementation

- 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

<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

- 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.
Prim's algorithm - Lazy implementation

- 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.

add to PQ all edges incident to 0

edges on PQ
(sorted by weight)

* 0–7 0.16
* 0–2 0.26
* 0–4 0.38
* 6–0 0.58
• 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.

delete 0–7 and add to MST

<table>
<thead>
<tr>
<th>edges on PQ</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>0–2</td>
<td>0.26</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

- 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.

MST edges

0–7

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–2</td>
<td>0.26</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

- 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.

add to PQ all edges incident to 7

MST edges

0–7

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 1–7  0.19</td>
</tr>
<tr>
<td>0–2  0.26</td>
</tr>
<tr>
<td>* 5–7  0.28</td>
</tr>
<tr>
<td>* 2–7  0.34</td>
</tr>
<tr>
<td>* 4–7  0.37</td>
</tr>
<tr>
<td>0–4  0.38</td>
</tr>
<tr>
<td>6–0  0.58</td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

delete 1-7 and add to MST

MST edges
0–7

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</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>
• 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.

**MST edges**

0–7, 1–7

**edges on PQ (sorted by weight)**

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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

- 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.

add to PQ all edges incident to 1

MST edges

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–2 0.26</td>
</tr>
<tr>
<td>5–7 0.28</td>
</tr>
<tr>
<td>* 1–3 0.29</td>
</tr>
<tr>
<td>* 1–5 0.32</td>
</tr>
<tr>
<td>* 2–7 0.34</td>
</tr>
<tr>
<td>* 1–2 0.36</td>
</tr>
<tr>
<td>4–7 0.37</td>
</tr>
<tr>
<td>0–4 0.38</td>
</tr>
<tr>
<td>6–0 0.58</td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

delete edge 0–2 and add to MST

**MST edges**

0–7  1–7

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<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>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>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>
• 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.

Prim's algorithm - Lazy implementation

edges on PQ (sorted by weight)

- 5–7 0.28
- 1–3 0.29
- 1–5 0.32
- 2–7 0.34
- 1–2 0.36
- 4–7 0.37
- 0–4 0.38
- 6–0 0.58

MST edges

0–7 1–7 0–2
Prim's algorithm - Lazy implementation

- 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.

add to PQ all edges incident to 2

no need to add edge 1-2 or 2-7 because it's already obsolete

MST edges

0-7  1-7  0-2

Edges on PQ (sorted by weight)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>2-3</td>
<td>0.17</td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>1-3</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>1-5</td>
<td>0.32</td>
<td></td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
<td></td>
</tr>
<tr>
<td>1-2</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>0-4</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>6-2</td>
<td>0.40</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
<td></td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

delete 2–3 and add to MST

MST edges

0–7 1–7 0–2

edges on PQ
(sorted by weight)

* 2–3 0.17
5–7 0.28
1–3 0.29
1–5 0.32
2–7 0.34
1–2 0.36
4–7 0.37
0–4 0.38
* 6–2 0.40
6–0 0.58
Prim's algorithm - Lazy implementation

- 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.

MST edges

0–7   1–7   0–2   2–3

edges on PQ (sorted by weight)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>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>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

add to PQ all edges incident to 3

MST edges

0–7  1–7  0–2  2–3

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5–7  0.28</td>
</tr>
<tr>
<td>1–3  0.29</td>
</tr>
<tr>
<td>1–5  0.32</td>
</tr>
<tr>
<td>2–7  0.34</td>
</tr>
<tr>
<td>1–2  0.36</td>
</tr>
<tr>
<td>4–7  0.37</td>
</tr>
<tr>
<td>0–4  0.38</td>
</tr>
<tr>
<td>6–2  0.40</td>
</tr>
<tr>
<td>*  3–6  0.52</td>
</tr>
<tr>
<td>6–0  0.58</td>
</tr>
</tbody>
</table>
• 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.

delete 5–7 and add to MST

MST edges

0–7  1–7  0–2  2–3

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>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>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

MST edges

0–7  1–7  0–2  2–3  5–7

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>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>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

add to PQ all edges incident to 5

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

MST edges

0–7  1–7  0–2  2–3  5–7
Prim's algorithm - Lazy implementation

- 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.

delete 1–3 and discard obsolete edge

MST edges

0–7 1–7 0–2 2–3 5–7
Prim's algorithm - Lazy implementation

- 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.

MST edges

0–7  1–7  0–2  2–3  5–7

del 1–5 and discard obsolete edge

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–5  0.32</td>
</tr>
<tr>
<td>2–7  0.34</td>
</tr>
<tr>
<td>4–5  0.35</td>
</tr>
<tr>
<td>1–2  0.36</td>
</tr>
<tr>
<td>4–7  0.37</td>
</tr>
<tr>
<td>0–4  0.38</td>
</tr>
<tr>
<td>6–2  0.40</td>
</tr>
<tr>
<td>3–6  0.52</td>
</tr>
<tr>
<td>6–0  0.58</td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

Delete 2–7 and discard obsolete edge

MST edges

<table>
<thead>
<tr>
<th></th>
<th>MST edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–7</td>
<td></td>
</tr>
<tr>
<td>1–7</td>
<td></td>
</tr>
<tr>
<td>0–2</td>
<td></td>
</tr>
<tr>
<td>2–3</td>
<td></td>
</tr>
<tr>
<td>5–7</td>
<td></td>
</tr>
</tbody>
</table>

Edges on PQ (sorted by weight):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2–7</td>
<td>0.34</td>
<td></td>
</tr>
<tr>
<td>4–5</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>1–2</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>4–7</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>0–4</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>6–2</td>
<td>0.40</td>
<td></td>
</tr>
<tr>
<td>3–6</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>6–0</td>
<td>0.58</td>
<td></td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

delete 4–5 and add to MST

MST edges

0–7 1–7 0–2 2–3 5–7
Prim's algorithm - Lazy implementation

- 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.

MST edges

0–7  1–7  0–2  2–3  5–7  4–5
Prim's algorithm - Lazy implementation

- 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.

add to PQ all edges incident to 4

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>

MST edges

0−7  1−7  0−2  2−3  5−7  4−5
Prim's algorithm - Lazy implementation

- 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.

\[ \begin{align*}
\text{MST edges} & \quad 0-7 \quad 1-7 \quad 0-2 \quad 2-3 \quad 5-7 \quad 4-5 \\
\text{edges on PQ} & \quad \text{(sorted by weight)} \\
& \quad 1-2 \quad 0.36 \\
& \quad 4-7 \quad 0.37 \\
& \quad 0-4 \quad 0.38 \\
& \quad 6-2 \quad 0.40 \\
& \quad 3-6 \quad 0.52 \\
& \quad 6-0 \quad 0.58 \\
& \quad 6-4 \quad 0.93 \\
\end{align*} \]
Prim's algorithm - Lazy implementation

- 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.

delete 4–7 and discard obsolete edge

edges on PQ
(sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>

MST edges

0–7  1–7  0–2  2–3  5–7  4–5
Prim's algorithm - Lazy implementation

- 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.

delete 0–4 and discard obsolete edge

**MST edges**

- 0–7
- 1–7
- 0–2
- 2–3
- 5–7
- 4–5

<table>
<thead>
<tr>
<th>edges on PQ (sorted by weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–4 0.38</td>
</tr>
<tr>
<td>6–2 0.40</td>
</tr>
<tr>
<td>3–6 0.52</td>
</tr>
<tr>
<td>6–0 0.58</td>
</tr>
<tr>
<td>6–4 0.93</td>
</tr>
</tbody>
</table>
Prim's algorithm - Lazy implementation

- 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.

Delete 6–2 and add to MST

**MST edges**

0–7  1–7  0–2  2–3  5–7  4–5

**edges on PQ**

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>
• 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.

delete 6–2 and add to MST

MST edges

0–7  1–7  0–2  2–3  5–7  4–5  6–2

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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

- 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.

stop since $V-1$ edges

edges on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<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>

MST edges

0–7 1–7 0–2 2–3 5–7 4–5 6–2
Prim's algorithm - Lazy implementation

- 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.

MST edges

0–7  1–7  0–2  2–3  5–7  4–5  6–2
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())
        {
            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);
        }
    }
}

Prim's algorithm: lazy implementation

- 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 v or w to tree
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 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 $T$.

Eager solution. Maintain a PQ of vertices connected by an edge to $T$, where priority of vertex $v$ = weight of shortest 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 shortest edge connecting $x$ to $T$
Prim's algorithm - Eager implementation

- 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

<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 - Eager implementation

- 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.
Prim's algorithm - Eager implementation

- 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.

<table>
<thead>
<tr>
<th>vertex</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>4</td>
<td>0–4</td>
<td>0.38</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

vertices on PQ (sorted by weight)

add vertices 7, 2, 4, and 6 to PQ
• 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.
Prim's algorithm - Eager implementation

- 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.

### Prim's Table

<table>
<thead>
<tr>
<th>vertex</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>4</td>
<td>0–4</td>
<td>0.38</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

vertices on PQ (sorted by weight)

MST edges

0–7
Prim's algorithm - Eager implementation

- 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.

<table>
<thead>
<tr>
<th>vertices on PQ (sorted by weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>0–4</td>
<td>0.38</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

MST edges

0–7

decrease key of vertex 4 from 0.38 to 0.37
add vertex 1 to PQ
add vertex 5 to PQ
already a better connection to 2 (discard)
• 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.

Prim's algorithm - Eager implementation

MST edges
0–7  1–7

vertices on PQ (sorted by weight)

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>
Prim's algorithm - Eager implementation

- 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.

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

vertices on PQ (sorted by weight)

MST edges

0–7  1–7
Prim's algorithm - Eager implementation

- 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.

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>3</td>
<td>1–3</td>
<td>0.29</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

- add vertex 3 to PQ
- already a better connection to 5 and 7 (discard)

MST edges

0–7  1–7
Prim's algorithm - Eager implementation

• 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.

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>3</td>
<td>1–3</td>
<td>0.29</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

MST edges

0–7   1–7
• 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.

Prim's algorithm - Eager implementation

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>3</td>
<td>1–3</td>
<td>0.29</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–0</td>
<td>0.58</td>
</tr>
</tbody>
</table>

MST edges

0–7  1–7  0–2
Prim's algorithm - Eager implementation

- 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.

**MST edges**

- $0-7$
- $1-7$
- $0-2$
Prim's algorithm - Eager implementation

- 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.

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>

MST edges

0–7  1–7  0–2  2–3
Prim's algorithm - Eager implementation

- 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.

### MST edges

0–7 1–7 0–2 2–3

```
<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>
```
Prim's algorithm - Eager implementation

• 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.

MST edges

0–7 1–7 0–2 2–3

already a better connection to 6 (discard)
Prim's algorithm - Eager implementation

- 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.

## Prim's algorithm

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>

### MST edges

- 0–7
- 1–7
- 0–2
- 2–3
• 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.

**Prim's algorithm - Eager implementation**

<table>
<thead>
<tr>
<th>$v$</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–7</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>

**MST edges**

0–7  1–7  0–2  2–3  5–7
- 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.

**MST edges**

0–7  1–7  0–2  2–3  5–7
Prim’s algorithm - Eager implementation

- 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.

**MST edges**

0–7  1–7  0–2  2–3  5–7
Prim's algorithm - Eager implementation

- 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.

MST edges

0–7 1–7 0–2 2–3 5–7 4–5
Prim's algorithm - Eager implementation

- 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 a graph with vertices and edges labeled with weights. The MST edges are highlighted in black.]

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>0.35</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>

MST edges:

- 0–7
- 1–7
- 0–2
- 2–3
- 5–7
- 4–5

already a better connection to 6 (discard)
Prim's algorithm - Eager implementation

- 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.

### MST edges

- 0–7
- 1–7
- 0–2
- 2–3
- 5–7
- 4–5

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>0.35</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>
Prim's algorithm - Eager implementation

- 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.

MST edges

0–7  1–7  0–2  2–3  5–7  4–5  6–2
Prim's algorithm - Eager implementation

• 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.

MST edges

0–7  1–7  0–2  2–3  5–7  4–5  6–2

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0–7</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1–7</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>0–2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>2–3</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>5–7</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>0.35</td>
</tr>
<tr>
<td>6</td>
<td>6–2</td>
<td>0.40</td>
</tr>
</tbody>
</table>
Indexed priority queue

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

- Client can insert and delete-the-minimum.
- Client can change the key by specifying the index.

```java
public class IndexMinPQ<Key extends Comparable<Key>> {
  IndexMinPQ(int N) {
    // create indexed priority queue with indices 0, 1, ..., N-1
    associate key with index k
  }
  void insert(int k, Key key) {
    // decrease the key associated with index k
  }
  void decreaseKey(int k, Key key) {
  }
  boolean contains() {
    // is k an index on the priority queue?
  }
  int delMin() {
    // remove a minimal key and return its associated index
  }
  boolean isEmpty() {
    // is the priority queue empty?
  }
  int size() {
    // number of entries in the priority queue
  }
}
```
Indexed priority queue implementation

Implementation.

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

![Priority Queue Diagram](image)

<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>pq[i]</td>
<td>-</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</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>
</tbody>
</table>
Prim's algorithm: running time

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>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 (Johnson 1975)</td>
<td>$d \log_d V$</td>
<td>$d \log_d V$</td>
<td>$\log_d V$</td>
<td>$E \log_{E/V} V$</td>
</tr>
<tr>
<td>Fibonacci heap (Fredman-Tarjan 1984)</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.
Minimum Spanning Trees

- Greedy algorithm
- Edge-weighted graph API
- Kruskal's algorithm
- Prim's algorithm
- Context
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 $\sim c N \log N$. 
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.

outbreak of cholera deaths in London in 1850s (Nina Mishra)

**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.
“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 merge the two clusters.
- Repeat until there are exactly k clusters.

Observation. This is Kruskal's algorithm (stop when k connected components).

Alternate solution. Run Prim's algorithm and delete k-1 max weight edges.
Dendrogram. Tree diagram that illustrates arrangement of clusters.
Dendrogram. Tree diagram that illustrates arrangement of clusters.

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html
**Dendrogram.** Tree diagram that illustrates arrangement of clusters.

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html
Dendrogram. Tree diagram that illustrates arrangement of clusters.

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html
Dendrogram. Tree diagram that illustrates arrangement of clusters.

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html
**Dendrogram.** Tree diagram that illustrates arrangement of clusters.

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html
Tumors in similar tissues cluster together.

Reference: Botstein & Brown group