Minimum Spanning Trees

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.
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.

Brute force. Try all spanning trees?

Network design

MST of bicycle routes in North Seattle

Models of nature

MST of random graph
MINIMUM SPANNING TREES

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

Applications

MST is a 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)

Cut property

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 / in cycle must be a crossing edge.
- Removing / and adding \( e \) is also a spanning tree.
- Since weight of \( e \) is less than the weight of /, that spanning tree is lower weight.
- Contradiction.

Cut property: adding \( e \) to MST creates a cycle

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

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  6-2  0-7  2-3

min-weight crossing edge

crossing edges (sorted by weight)

\[
\begin{align*}
1-7 & 0.19 \\
1-3 & 0.29 \\
1-5 & 0.32 \\
4-5 & 0.35 \\
1-2 & 0.36 \\
4-7 & 0.37 \\
0-4 & 0.38 \\
6-4 & 0.93
\end{align*}
\]

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  5-7  6-2  0-7  2-3  1-7

min-weight crossing edge

crossing edges (sorted by weight)

\[
\begin{align*}
4-5 & 0.35 \\
4-7 & 0.37 \\
0-4 & 0.38 \\
6-4 & 0.93
\end{align*}
\]
Greedy MST algorithm: correctness proof

**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)

![a cut with no black crossing edges](image)

Greedy MST algorithm: efficient implementations

**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)

![weights need not be proportional to distance](image)

**Q.** What if graph is not connected?
- **A.** Compute minimum spanning forest = MST of each component.

![can independently compute MST of components](image)

---

**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.**

```java
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;
    }
}
```

**Idiom for processing an edge e:**

```java
int v = e.either(), w = e.other(v);
```

**Weighted edge: Java implementation**

**Constructor**

Maintain vertex-indexed array of `Edge` objects.

**Edge-weighted graph API**

**Conventions.** Allow self-loops and parallel edges.

```java
public class EdgeWeightedGraph {
    public EdgeWeightedGraph(int V) {
    }
    public EdgeWeightedGraph(In in) {
    }
    void addEdge(Edge e) {
    }
    Iterable<Edge> adj(int v) {
    }
    Iterable<Edge> edges() {
    }
    int V() {
    }
    int E() {
    }
    String toString() {
    }
}
```

**Edge-weighted graph: adjacency-lists representation**

Maintain vertex-indexed array of `Edge` lists.
**Minimum Spanning Tree API**

**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
}
```

```
% 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.

Graph edges sorted by weight:

- $0-7 \ 0.16$
- $2-3 \ 0.17$
- $1-7 \ 0.19$
- $0-2 \ 0.26$
- $5-7 \ 0.28$
- $1-3 \ 0.29$
- $1-5 \ 0.32$
- $2-7 \ 0.34$
- $4-5 \ 0.35$
- $1-2 \ 0.36$
- $4-7 \ 0.37$
- $0-4 \ 0.38$
- $6-2 \ 0.40$
- $3-6 \ 0.52$
- $6-0 \ 0.58$
- $6-4 \ 0.93$

Graph:

- Adding $0-7$ to the MST does not create a cycle.
- Adding $2-3$ to the 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.

Graph:

- Adding $0-7$ to the MST does not create a cycle.
- Adding $2-3$ to the 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.

Graph:

- Adding $0-7$ to the MST does not create a cycle.
- Adding $2-3$ to the 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.

Graph:

- Adding $0-7$ to the MST does not create a cycle.
- Adding $2-3$ to the 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.

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.

```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
1-5  0.32
```

creates a cycle

```
not in MST
```

```
2-7  0.34
```

creates a cycle not in MST

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

```
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
```

in MST

```
4-5  0.35
```

does not create a cycle

```
```

```
```

creates a cycle

```
not in MST
```

```
4-5  0.35
```

creates a cycle

```
```

```
```

creates a cycle

```
not in MST
```

```
4-7  0.37
```
Kruskal’s algorithm

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

0-7 0.16
2-3 0.17
1-7 0.19
0-2 0.26
5-7 0.28
1-3 0.23
1-5 0.32
2-7 0.34
4-5 0.35
1-3 0.36
4-7 0.37
0-4 0.38

creates a cycle
not in MST

0-7 0.16
2-3 0.17
1-7 0.19
0-2 0.26
5-7 0.28
1-3 0.23
1-5 0.32
2-7 0.34
4-5 0.35
1-3 0.36
4-7 0.37
0-4 0.38
6-0 0.58

creates a cycle
not in MST

creates a cycle
not in MST

creates a cycle
not in MST

creates a cycle
not in MST
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: visualization

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

Proof. Kruskal's algorithm is a special case of the greedy MST algorithm.
- Suppose Kruskal's algorithm colors the edge $e = v - w$ black.
- Cut $= \{v\}$ set of vertices connected to $v$ in tree $T$.
- No crossing edge is black.
- No crossing edge has lower weight. Why?

$0-7$ 0.16
$2-3$ 0.17
$1-7$ 0.19
$0-2$ 0.26
$5-7$ 0.28
$1-3$ 0.23
$1-5$ 0.32
$2-7$ 0.34
$4-5$ 0.35
$1-2$ 0.36
$4-7$ 0.37
$0-4$ 0.38
$6-2$ 0.40
$3-6$ 0.52
$6-0$ 0.58
$6-4$ 0.93

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: implementation challenge**

**Challenge.** Would adding edge \( v - w \) to tree \( T \) create a cycle? If not, add it.

**How difficult?**
- \( E + V \)
- \( V \)
- \( \log V \)
- \( \log^* V \)
- \( \frac{1}{61} \)

**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 \( w \).

**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;  }
}
```

**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.

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

```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

![Graph](image)

```
• 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.

![Graph](image)
```
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

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  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

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

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

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
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
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: 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 $= \text{set of vertices connected on tree.}$
- No crossing edge is black.
- No crossing edge has lower weight.

Prim's algorithm: proof of correctness

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

How difficult?
- $E$
- $V$
- $\log E$
- $\log^* E$
- $1$

Prim's algorithm: implementation challenge

edge $e = 7-5$ added to tree

1-7 0.19
0-2 0.26
2-7 0.28
5-7 0.37
4-7 0.39
0-4 0.38
6-0 0.48
**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 $e = v\to w$ 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.

Prim's algorithm - Lazy implementation

- Add to PQ all edges incident to $0$
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 0-7 and add to MST

 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 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 1-7 and add to MST
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.

edges on PQ (sorted by weight):

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

MST edges
0-7  1-7

delete edge 0-2 and add to MST

edges on PQ (sorted by weight):

0-2  0.26
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 1

edges on PQ (sorted by weight):

0-2  0.26
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

edge becomes obsolete (lazy implementation leaves on PQ)

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.

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 \( T \) the min weight edge with exactly one endpoint in \( T \). Repeat until \( V-1 \) edges.

![Diagram](image1)

**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

![Diagram](image2)

**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

![Diagram](image3)

**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

![Diagram](image4)

**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

![Diagram](image5)

**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

![Diagram](image6)

**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

![Diagram](image7)

**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

![Diagram](image8)

**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

![Diagram](image9)

**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

![Diagram](image10)

**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)**

<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>6-7</td>
<td>0.37</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</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

**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>6-7</td>
<td>0.37</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</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

**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>6-7</td>
<td>0.37</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</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

**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>6-7</td>
<td>0.37</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</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 1-5 and discard obsolete edge**

**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>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>4-5</td>
<td>0.35</td>
</tr>
<tr>
<td>3-6</td>
<td>0.38</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>

**delete 2-7 and discard obsolete edge**

**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>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>4-5</td>
<td>0.35</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>

**delete 4-5 and add to MST**

**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>4-5</td>
<td>0.35</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</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>
</tbody>
</table>

**delete 6-5**
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 - 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

**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 1-2 and discard obsolete edge

**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

**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
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

<table>
<thead>
<tr>
<th>MST edges</th>
<th>0-7</th>
<th>1-7</th>
<th>0-2</th>
<th>2-3</th>
<th>5-7</th>
<th>4-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-2</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-4</td>
<td>0.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

edges on PQ (sorted by weight):

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

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

<table>
<thead>
<tr>
<th>MST edges</th>
<th>0-7</th>
<th>1-7</th>
<th>0-2</th>
<th>2-3</th>
<th>5-7</th>
<th>4-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-2</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-4</td>
<td>0.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

edges on PQ (sorted by weight):

- 6-0 0.58
- 6-4 0.93

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

<table>
<thead>
<tr>
<th>MST edges</th>
<th>0-7</th>
<th>1-7</th>
<th>0-2</th>
<th>2-3</th>
<th>5-7</th>
<th>6-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-2</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-4</td>
<td>0.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

edges on PQ (sorted by weight):

- 6-0 0.58
- 6-4 0.93

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.
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);
        }
    }
}

117

Prim's algorithm:  lazy implementation

Prim's algorithm:  lazy implementation

Prim's algorithm:  lazy implementation

Prim's algorithm:  eager implementation

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>

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 $e = v \rightarrow T$ to $T$.
- Update PQ by considering all edges $e = v \rightarrow x$ incident to $v$.
  - Ignore if $x$ is already in $T$.
  - Add $x$ to PQ if not already on it.
  - Decrease priority of $x$ if $v \rightarrow x$ 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.

![Image of an edge-weighted graph]

vertices on PQ (sorted by weight)

- Add vertices 7, 2, 4, and 6 to PQ

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.

![Image of a partially constructed tree]

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.

![Image of a partially constructed tree]

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.

![Image of a partially constructed tree]
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

**v edgeTo[] distTo[]**

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<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 vertex 7 to PQ
add vertex 2 to PQ
already a better connection to 2 (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

**v edgeTo[] distTo[]**

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<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)

decrease key of vertex 4 from 0.38 to 0.37
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.

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

MST edges

0-7 1-7

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

MST edges

0-7 1-7

decrease key of vertex 3 from 0.29 to 0.17
decrease key of vertex 6 from 0.58 to 0.40
now better connections to 0 and 1 (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

---

already a better connection to 6 placed
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.

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

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.

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

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.

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

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.

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

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.

**MST 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>4–5</td>
<td>0.35</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.
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.

```
public class IndexMinPQ<Key extends Comparable<Key>>
{
    IndexMinPQ(int N)
    { create indexed priority queue with indices 0, 1, ..., N-1
    }
    void insert(int k, Key key)
    { associate key with index k
    }
    void decreaseKey(int k, Key key)
    { decrease the key associated with index k
    }
    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)`.

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></td>
<td>1</td>
<td>( V )</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 ( V )</td>
<td>d log ( V )</td>
<td>log ( V )</td>
<td>( E ) log ( V )</td>
</tr>
<tr>
<td>Fibonacci heap (Fredman-Tarjan 1984)</td>
<td>( 1 + ) log ( V )</td>
<td>( 1 + ) log ( V )</td>
<td>( 1 + ) ( E + V \log \log V )</td>
<td></td>
</tr>
</tbody>
</table>

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.
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 \( \sim c N \log N \).

**Euclidean 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.

**“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.
Tumors in similar tissues cluster together.