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.

![Graph with weights](image)

**MST of bicycle routes in North Seattle**

![MST of bicycle routes](http://www.flickr.com/photos/ewedistrict/21980840)

**Network design**

**Models of nature**

**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.

![Graph with weights](image)

**MST of random graph**

![MST of random graph](http://algo.inria.fr/broutin/gallery.html)

**Models of nature**

**Brute force.** Try all spanning trees?

**Network design**

MST of bicycle routes in North Seattle

![MST of bicycle routes](http://www.flickr.com/photos/ewedistrict/21980840)

**Models of nature**

MST of random graph

![MST of random graph](http://algo.inria.fr/broutin/gallery.html)
Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research

http://www.bccrc.ca/ci/ta01_archlevel.html

Applications

MST is fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Reducing data storage in sequencing amino acids in a protein.
- Model locality of particle interactions in turbulent fluid flows.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, cable, computer, road).


Minimum Spanning Trees

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

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 $f$ 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.

---

**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
```

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
```

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**

<table>
<thead>
<tr>
<th></th>
<th>0-2</th>
<th>5-7</th>
<th>6-2</th>
<th>0-7</th>
<th>2-3</th>
<th>1-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.7</td>
<td>0.19</td>
<td>0.29</td>
<td>0.32</td>
<td>0.35</td>
<td>0.36</td>
</tr>
<tr>
<td>3</td>
<td>0.37</td>
<td>0.38</td>
<td>0.38</td>
<td>0.93</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Crossing edges** (sorted by weight)

- 1-7
- 1-3
- 1-5
- 4-5
- 1-2
- 4-7
- 0-4
- 6-4
**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)

**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)

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

**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;
    }
    // Other methods...
}
```

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

Weighted edge: Java implementation

```java
public class Edge implements Comparable<Edge> {
    // Fields...
    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;
    }
    // Other methods...
}
```

Edge-weighted graph API

```java
public class EdgeWeightedGraph {
    // Fields...
    public EdgeWeightedGraph(int V) {
    }
    public EdgeWeightedGraph(In in) {
    }
    void addEdge(Edge e) {
    }
    Iterable<Edge> adj(int v) {
        // Return iterable...
    }
    Iterable<Edge> edges() {
        // Return iterable...
    }
    int V() {
        // Return integer...
    }
    int E() {
        // Return integer...
    }
    String toString() {
        // Return string...
    }
    // Other methods...
}
```

Conventions. Allow self-loops and parallel edges.

Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of edge lists.

```
0: 1 2 6 7
1: 0 2 6 7
2: 0 1 5
3: 6
4: 3 5
5: 2 4
6: 0 1 2 3 5
7: 0 1 2
```

**Minimum spanning tree API**

Q. How to represent the MST?

```java
public class MST
{
    private final EdgeWeightedGraph G;

    MST(EdgeWeightedGraph G) // constructor
    Iterable<Edge> edges() // edges in MST
    double weight() // weight of MST
}
```

```java
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());
}
```

**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

**an edge–weighted graph**

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

**in MST**

0-7 0.16
2-3 0.17
1-7 0.19

**does not create a cycle**

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

**in MST**

0-7 0.16
2-3 0.17
1-7 0.19

**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.

In MST does not create a cycle

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

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.

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
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

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
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
### Kruskal's algorithm

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

#### Correctness Proof

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

**Pf.** Kruskal’s algorithm is a special case of the greedy MST algorithm.

- Suppose Kruskal’s algorithm colors the edge $e = v \rightarrow w$ black.
- Cut = set of vertices connected to $v$ in tree $T$.
- No crossing edge is black.
- No crossing edge has lower weight. Why!

---

<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>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>

---

**add edge to tree**
**Kruskal's algorithm: implementation challenge**

**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

![Diagram of a tree and adding an edge](image)

**Efficient solution.** Use the union-find data structure.
- Maintain a set for each connected component in $T$.
- If $v$ and $w$ are in same set, then adding $v \rightarrow w$ would create a cycle.
- To add $v \rightarrow w$ to $T$, merge sets containing $v$ and $w$.

**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 $E \log^* V$. 

recall: $\log^* V = 5$ 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.

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.
**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 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
- 6-2

**Proposition.** [Jarník 1930, Dijkstra 1957, Prim 1959]

Prim's algorithm computes the MST.

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

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

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

**Priority queue**

- 1-7 is min weight edge with exactly one endpoint in $T$
**Prim’s algorithm: lazy implementation**

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

**Lazy solution.** Maintain a PQ of edges with (at least) one endpoint in $T$.

- Key = edge; priority = weight of edge.
- Delete-min to determine next edge $e = v \rightarrow w$ 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.

![Diagram](image.png)

**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.

![Diagram](image.png)
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.

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.

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.

```
MST edges
0-7 1-7
```

```
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
```

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.

```
edge becomes obsolete
(lazy implementation leaves on PQ)
```

```
MST edges
0-7 1-7 0-2
```

```
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
```
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**

- $0-7$
- $1-7$
- $0-2$

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

<table>
<thead>
<tr>
<th></th>
<th>0-7</th>
<th>1-7</th>
<th>0-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-3</td>
<td>0.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-5</td>
<td>0.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-2</td>
<td>0.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0-4</td>
<td>0.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Edges on PQ**

<table>
<thead>
<tr>
<th></th>
<th>2-3</th>
<th>5-7</th>
<th>1-3</th>
<th>1-5</th>
<th>2-7</th>
<th>1-2</th>
<th>4-7</th>
<th>0-4</th>
<th>6-2</th>
<th>6-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-7</td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0-2</td>
<td>0.38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-3</td>
<td>0.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-7</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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**

- $0-7$
- $1-7$
- $0-2$

**Add to PQ all edges incident to 3**

**MST edges**

- $0-7$
- $1-7$
- $0-2$
- $2-3$

**Edges on PQ**

- $0-7$
- $1-7$
- $0-2$
- $2-3$

**Add to PQ all edges incident to 3**

**MST edges**

- $0-7$
- $1-7$
- $0-2$
- $2-3$

**Edges on PQ**

- $0-7$
- $1-7$
- $0-2$
- $2-3$
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)

| 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 |
| 3-6  | 0.52 |
| 6-0  | 0.58 |

### Edges on PQ sorted by weight

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

delete 5-7 and add to MST

### Edges on PQ

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

add to PQ all edges incident to 5

### 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 1–5 and discard obsolete edge

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
- 0–7
- 1–7
- 0–2
- 2–3
- 5–7

Edges on PQ (sorted by weight)
- 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

Delete 4–5 and add to MST

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
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)
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
```

MST edges
0-7  1-7  0-2  2-3  5-7  4-5

delete 1-2 and discard obsolete edge

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

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

```
edges on PQ (sorted by weight)
0-7 0.73
0-4 0.38
6-2 0.40
3-6 0.52
6-0 0.58
*6-4 0.93
```

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 3-6 and discard obsolete edge

```
edges on PQ (sorted by weight)
0-7 0.73
0-4 0.38
6-2 0.40
*6-4 0.93
```

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.

MST edges
- Stop since $V-1$ edges

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**

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

    public LazyPrimMST(WeightedGraph G) {
        pq = new MinPQ<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);
        while (!pq.isEmpty()) {
            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);
        }
    }
}
```

**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>

**Prim's algorithm: eager implementation**

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

**Eager solution.** Maintain a PQ of vertices connected by an edge to $T$, where priority of vertex $v$ = weight of shortest edge connecting $v$ to $T$.
- Delete min vertex $v$ and add its associated edge $e = v \rightarrow w$ 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.

![Diagram of an edge-weighted graph](image)

vertices on PQ (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

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.

<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>vertex</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<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.

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

add vertex 3 to PQ

MST edges
0-7 1-7

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.

<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>1</td>
<td>7–0</td>
<td>0.16</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>7–5</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>7–4</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>2–6</td>
<td>0.40</td>
</tr>
</tbody>
</table>

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.

<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>1</td>
<td>7–0</td>
<td>0.16</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>7–5</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>7–4</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>2–6</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.

\[ \begin{array}{c|c|c} v & \text{edgeTo[]} & \text{distTo[]} \\ \hline 0 & - & - \\ 7 & 0\rightarrow7 & 0.16 \\ 1 & 1\rightarrow7 & 0.19 \\ 2 & 0\rightarrow2 & 0.26 \\ 3 & 2\rightarrow3 & 0.17 \\ 5 & 5\rightarrow7 & 0.28 \\ 4 & 4\rightarrow7 & 0.37 \\ 6 & 6\rightarrow2 & 0.40 \\ \end{array} \]

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.

\[ \begin{array}{c|c|c} v & \text{edgeTo[]} & \text{distTo[]} \\ \hline 0 & - & - \\ 7 & 0\rightarrow7 & 0.16 \\ 1 & 1\rightarrow7 & 0.19 \\ 2 & 0\rightarrow2 & 0.26 \\ 3 & 2\rightarrow3 & 0.17 \\ 5 & 5\rightarrow7 & 0.28 \\ 4 & 4\rightarrow5 & 0.35 \\ 6 & 6\rightarrow2 & 0.40 \\ \end{array} \]

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.

\[ \begin{array}{c|c|c} v & \text{edgeTo[]} & \text{distTo[]} \\ \hline 7 & 0\rightarrow7 & 0.16 \\ 1 & 1\rightarrow7 & 0.19 \\ 2 & 0\rightarrow2 & 0.26 \\ 3 & 2\rightarrow3 & 0.17 \\ 5 & 5\rightarrow7 & 0.28 \\ 4 & 4\rightarrow5 & 0.35 \\ 6 & 6\rightarrow2 & 0.40 \\ \end{array} \]

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.

## 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>4–5</td>
<td>0.35</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)

- 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>4–5</td>
<td>0.35</td>
</tr>
</tbody>
</table>

MST edges
0–7 1–7 0–2 2–3 5–7 4–5
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
    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[i]`, `pq[i]`, and `qp[i]` 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)`. 

```
0  1  2  3  4  5  6  7  8
keys[i]  A  S  O  R  T  I  N  G  -
pq[i]   -  0  6  7  2  1  5  4  3
qp[i]   1  5  4  8  7  6  2  3  -
```

- 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>$E \log_d V$</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>1 $\dagger$</td>
<td>log V $\dagger$</td>
<td>1 $\dagger$</td>
<td>$E = V \log V$</td>
</tr>
</tbody>
</table>

$\dagger$ 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
**Euclidean MST**

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

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

**Ingenuity.** Exploit geometry and do it in $\sim c N \log N$.

---

**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.

---

**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 algorithm**

“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.

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html
Dendrogram

Tree diagram that illustrates arrangement of clusters.

Dendrogram of cancers in human

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group