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.

Brute force. Try all spanning trees?

Network design

MST of bicycle routes in North Seattle

http://www.flickr.com/photos/eedistrict/219808480

Models of nature

MST of random graph

http://algo.inria.fr/~broutin/gallery.html
Medical image processing

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 $f$ 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.

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.

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.

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.

21

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.

22

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.

23

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.

24
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.

```

```
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: 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;
    }
}
```

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

Edge-weighted graph API

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

Conventions. Allow self-loops and parallel edges.
public class EdgeWeightedGraph {
    private final int V;
    private final Bag<Edge>[] adj;

    public EdgeWeightedGraph(int V) {
        this.V = V;
        adj = (Bag<Edge>[])(new Bag[V]);
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Edge>();
    }

    public void addEdge(Edge e) {
        int v = e.either(), w = e.other(v);
        adj[v].add(e);
        adj[w].add(e);
    }

    public Iterable<Edge> adj(int v) {
        return adj[v];
    }
}

Minimum spanning tree API

Q. How to represent the MST?

public class MST {
    public MST(EdgeWeightedGraph G) {
        // MST constructor
    }

    public Iterable<Edge> edges() {
        // edges in MST
    }

    public double weight() {
        // weight of MST
    }
}

Minimum spanning tree API:

- 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

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 → 0-7 0.16

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 → 2-3 0.17

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 → 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 → 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.

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

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.

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

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
1-3  0.29

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.

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

not in MST

creates 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.

creates a cycle

not in MST

creates a cycle

not in MST

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.
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

Kruskal's algorithm: correctness proof

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

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

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>

\( \dagger \) amortized bound using weighted union with path compression

**Remark.** If edges are already sorted, order of growth is \( E \log^* V \).
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 0.35
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
```

Context

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

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.

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.

### Prim's algorithm: visualization

![Graph with MST edges: 0-7, 1-7, 0-2, 2-3, 5-7, 4-5, 6-2]

### Prim's algorithm: proof of correctness

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

Prim's algorithm computes the MST.

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

Edge $e = 7-5$ added to tree

### Prim's algorithm: implementation challenge

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

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

Try all edges

Use a priority queue!
**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.

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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.

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 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.

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.

add to PQ all edges incident to 1

MST edges
0-7 1-7

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

delete edge 0-2 and add to MST

MST edges
0-7 1-7

edge becomes obsolete (lazy implementation leaves on PQ)

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.

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)

| 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 |

add to PQ all edges incident to 3

MST edges

| 0-7 | 1-7 | 0-2 | 2-3 |

edges on PQ (sorted by weight)

| 5-7 | 0.28 |
| 1-3 | 0.32 |
| 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 |
• 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)
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

• 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)
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

add to PQ all edges incident to 5

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.

delete 1–5 and discard obsolete edge

```
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
```

MST edges
0–7  1–7  0–2  2–3  5–7

delete 2–7 and discard obsolete edge

```
edges on PQ (sorted by weight)
0-7  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
```

MST edges
0–7  1–7  0–2  2–3  5–7

delete 4–5 and add to MST

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

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 $T$ 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.

• 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.

• 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.

delete 6-2 and add to MST

MST edges

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

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

MST edges

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

stop since $V-1$ edges

MST edges

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

edges on PQ (sorted by weight)

3-6  0.52
6-0  0.58
6-4  0.93
**Prim's algorithm: lazy implementation**

```java
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**

```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;  }
```

**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>

**Lazy Prim's algorithm: running time**

**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 \(T\)

pq has at most one entry per vertex
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

v edgeTo[] distTo[]
0 - -
7 0–7 0.16
2 0–2 0.26
4 0–4 0.38
6 6–0 0.58
vertices on PQ (sorted by weight)

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.

v edgeTo[] distTo[]
0 - -
7 0–7 0.16
2 0–2 0.26
4 0–4 0.38
6 6–0 0.58
vertices on PQ (sorted by weight)
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 Prim's algorithm - Eager implementation]

MST edges
0-7

vertices on PQ
(sorted by weight)

- 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 - Eager implementation]

MST edges
0-7

vertices on PQ
(sorted by weight)
**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

**v edgeTo[] distTo[]**

<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)

- Decrease key of vertex 3 from 0.29 to 0.17
- Decrease key of vertex 6 from 0.58 to 0.40

**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

<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>

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 - 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
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

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
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
}
```

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)`.

Indexed priority queue implementation

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 (^{(1)})</td>
<td>( d \log_d V )</td>
<td>( d \log_d V )</td>
<td>( d \log_d V )</td>
<td>( E \log_d V )</td>
</tr>
<tr>
<td>Fibonacci heap (^{(2)})</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
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.
Tumors in similar tissues cluster together.

Reference: Botstein & Brown group