4.3 Minimum Spanning Trees

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

Minimum spanning tree

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

graph $G$

Minimum spanning tree

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

not a tree (not connected)
**Minimum spanning tree**

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

**Minimum spanning tree problem**

*Input.* Connected, undirected graph $G$ with positive edge weights.

*Output.* A spanning tree of minimum weight.

---

**Minimum spanning tree**

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

**Minimum spanning tree problem**

*Input.* Connected, undirected graph $G$ with positive edge weights.

*Output.* A spanning tree of minimum weight.

---

Brute force. Try all spanning trees?
Minimum spanning trees: quiz 1

Let $G$ be a connected edge-weighted graph with $V$ vertices and $E$ edges. How many edges are in a MST of $G$?

A. $V - 1$
B. $V$
C. $E - 1$
D. $E$
E. I don't know.

Network design

MST of bicycle routes in North Seattle

http://www.flickr.com/photos/ewedistrict/21980840

Models of nature

MST of random graph

http://algo.inria.fr/brousin/gallery.html

Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research

http://www.bccrc.ca/ci/tud01_archlevel.html
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, computer, road).

Simplifying assumptions

For simplicity, we assume
- The graph is connected. $\Rightarrow$ MST exists.
- The edge weights are distinct. $\Rightarrow$ MST is unique.
Cut property

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

**Def.** A crossing edge connects a vertex in one set with a vertex in the other.

**Cut property.** Given any cut, the crossing edge of min weight is in the MST.

---

**Greedy MST algorithm demo**

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

---

**Minimum spanning trees: quiz 2**

Which is the min weight edge crossing the cut \( \{2, 3, 5, 6\} \)?

- **A.** 0–7 (0.16)
- **B.** 2–3 (0.17)
- **C.** 0–2 (0.26)
- **D.** 5–7 (0.28)
- **E.** I don't know.

---

Suppose min-weight crossing edge \( e \) is not in the MST.

- Adding \( e \) to the MST creates a cycle.
- Some other edge \( f \) in cycle must be a crossing edge.
- Removing \( f \) and adding \( e \) is also a spanning tree.
- Since weight of \( e \) is less than the weight of \( f \), that spanning tree has lower weight.
- Contradiction. □
Greedy MST algorithm: correctness proof

**Proposition.** The greedy algorithm computes the MST.

**Pf.**
- Any edge colored black is in the MST (via cut property).
- Fewer than $V-1$ black edges $\implies$ cut with no black crossing edges.
  (consider cut whose vertices are any one connected component)

![Diagram of a cut with no black crossing edges and fewer than V-1 edges colored black]

Greedy MST algorithm: efficient implementations

**Proposition.** The greedy algorithm computes the MST.

**Efficient implementations.** Find cut? Find min-weight edge?

- Ex 1. Kruskal's algorithm. [stay tuned]
- Ex 2. Prim's algorithm. [stay tuned]

Removing two simplifying assumptions

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

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

![Graph with weights]

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

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

![Graph with weights]

Greed is good

Gordon Gecko (Michael Douglas) address to Teldar Paper Stockholders in Wall Street (1986)
4.3 Minimum Spanning Trees

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

Weighted edge API

Edge abstraction needed for weighted edges.

```java
public class Edge implements Comparable<Edge> {
    private final int v;
    private final int w;
    private final double weight;

    public Edge(int v, int w, double weight) {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int either() {
        return v;
    }

    public int other(int vertex) {
        if (vertex == v) return w;
        else return v;
    }

    public int compareTo(Edge that) {
        if (this.weight < that.weight) return -1;
        else if (this.weight > that.weight) return +1;
        else return 0;
    }
}
```

Weighted edge

Java implementation

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

Edge-weighted graph API

```java
public class EdgeWeightedGraph {
    private final int V;
    private final int E;
    private final double[] weight;

    public EdgeWeightedGraph(int V) {
        this.V = V;
        this.E = 0;
        this.weight = new double[V * V];
    }

    public EdgeWeightedGraph(In in) {
        this(V);
        read(in);
    }

    void addEdge(Edge e) {
        addEdge(e.v(), e.w(), e.weight());
    }

    Iterable<Edge> adj(int v) {
        return new Iterable<Edge>() {
            public Iterator<Edge> iterator() {
                return new Iterator<Edge>() {
                    int i = 0;

                    public boolean hasNext() {
                        return i < E;
                    }

                    public Edge next() {
                        return new Edge(i / V, i % V, weight[i]);
                    }
                };
            }
        };
    }

    Iterable<Edge> edges() {
        return new Iterable<Edge>() {
            public Iterator<Edge> iterator() {
                return new Iterator<Edge>() {
                    int i = 0;

                    public boolean hasNext() {
                        return i < E;
                    }

                    public Edge next() {
                        return new Edge(i / V, i % V, weight[i]);
                    }
                };
            }
        };
    }

    String toString() {
        return new String[] { V, E, weight };
    }
}
```

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

Maintain vertex-indexed array of Edge lists.

```
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(EdgeWeightedGraph G) {
    constructor

    Iterable<Edge> edges() {
        edges in MST

    double weight() {
        weight of MST
    }
}
```

4.3 Minimum Spanning Trees
Kruskal's algorithm demo

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](image)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.16</td>
</tr>
<tr>
<td>2-3</td>
<td>0.17</td>
</tr>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>0-2</td>
<td>0.26</td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
</tr>
<tr>
<td>1-3</td>
<td>0.29</td>
</tr>
<tr>
<td>1-5</td>
<td>0.32</td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>4-5</td>
<td>0.35</td>
</tr>
<tr>
<td>1-2</td>
<td>0.36</td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
</tr>
<tr>
<td>0-4</td>
<td>0.38</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>6-4</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Kruskal's algorithm: correctness proof

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

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

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

![Add edge to tree](image)

Kruskal's algorithm: implementation challenge

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

How difficult to implement?

- **A.** $E + V$
- **B.** $V$
- **C.** $\log V$
- **D.** $\log^* V$
- **E.** 1

![Add edge to tree](image)
Kruskal's algorithm: implementation challenge

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

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

![Diagram of Case 1: adding $v-w$ creates a cycle](image)

![Diagram of Case 2: add $v-w$ to $T$ and merge sets containing $v$ and $w$](image)

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

4.3 Minimum Spanning Trees

**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

build priority queue (or sort)
greedily add edges to MST
edge $v$-$w$ does not create cycle
merge connected components
add edge $e$ to MST

Kruskal's algorithm: running time

often called fewer than $E$ times
Prim's algorithm demo

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

![An edge-weighted graph](image)

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 $e = \text{min weight edge connecting a vertex on the tree to a vertex not on the tree.}$
- Cut = set of vertices connected on tree.
- No crossing edge is black.
- No crossing edge has lower weight.

![Edge e = 7-5 added to tree](image)

Prim's algorithm: implementation challenge

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

How difficult?

A. $E$
B. $V$
C. $\log E$
D. $1$
E. I don’t know.

![Priority queue of crossing edges](image)
**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 marked (both in $T$).
- Otherwise, let $w$ be the unmarked vertex (not in $T$):
  - add $e$ to $T$ and mark $w$
  - add to PQ any edge incident to $w$ (assuming other endpoint not in $T$)

\[
\text{1-7 is min weight edge with exactly one endpoint in } T
\]

**Prim's algorithm: lazy implementation demo**

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

\[
\begin{align*}
0-7 & : 0.16 \\
2-3 & : 0.17 \\
1-7 & : 0.19 \\
0-2 & : 0.26 \\
5-7 & : 0.28 \\
1-3 & : 0.29 \\
1-5 & : 0.32 \\
2-7 & : 0.34 \\
4-5 & : 0.35 \\
1-2 & : 0.36 \\
4-7 & : 0.37 \\
0-4 & : 0.38 \\
6-2 & : 0.40 \\
3-6 & : 0.52 \\
6-0 & : 0.58 \\
6-4 & : 0.93
\end{align*}
\]

\[
\text{an edge-weighted graph}
\]

---

```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() && mst.size() < G.V() - 1) {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (marked[v] && marked[w]) continue;
            if (!marked[v]) visit(G, v);
            if (!marked[w]) visit(G, w);
        }
    }

    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;
    }
}
```
Lazy Prim’s algorithm: running time

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

**Pf.**

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete min</td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
<tr>
<td>insert</td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
</tbody>
</table>

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

**Observation.** For each vertex $v$, need only **lightest** edge connecting $v$ to $T$.
- MST includes at most one edge connecting $v$ to $T$. Why?
- If MST includes such an edge, it must take lightest such edge. Why?

**Prim’s algorithm: eager implementation demo**

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

**Prim’s algorithm: eager implementation demo**

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V-1$ edges.
**Prim’s algorithm: eager implementation**

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

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

- Delete min vertex $v$ and add its associated edge $e = v \rightarrow w$ to $T$.
- Update PQ by considering all edges $e = x \rightarrow v$ 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 lightest edge connecting $x$ to $T$.

PQ has at most one entry per vertex

---

**Indexed priority queue:**

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

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

**Binary heap implementation.** [see Section 2.4 of textbook]

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

**Prim’s algorithm: which priority queue?**

Depends on PQ implementation: $V$ insert, $V$ delete-min, $E$ decrease-key.

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

$^+$ amortized

**Bottom line.**

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

• introduction
• greedy algorithm
• edge-weighted graph API
• Kruskal’s algorithm
• Prim’s algorithm
• context

Does a linear-time MST algorithm exist?

deterministic compare–based MST algorithms

<table>
<thead>
<tr>
<th>year</th>
<th>worst case</th>
<th>discovered by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1975</td>
<td>$E \log \log V$</td>
<td>Yao</td>
</tr>
<tr>
<td>1976</td>
<td>$E \log \log V$</td>
<td>Cheriton-Tarjan</td>
</tr>
<tr>
<td>1984</td>
<td>$E \log^* V, E + V \log V$</td>
<td>Fredman-Tarjan</td>
</tr>
<tr>
<td>1986</td>
<td>$E \log (\log^* V)$</td>
<td>Gabow-Galil-Spencer-Tarjan</td>
</tr>
<tr>
<td>1997</td>
<td>$E \alpha(V) \log \alpha(V)$</td>
<td>Chazelle</td>
</tr>
<tr>
<td>2000</td>
<td>$E \alpha(V)$</td>
<td>Chazelle</td>
</tr>
<tr>
<td>2002</td>
<td>optimal</td>
<td>Pettie-Ramachandran</td>
</tr>
<tr>
<td>20xx</td>
<td>$E$</td>
<td>???</td>
</tr>
</tbody>
</table>

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

Euclidean MST

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

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

Ingenuity. Exploit geometry and do it in $N \log N$ time.

Maximum Spanning Tree

Problem. Given an edge-weighted graph $G$, find a spanning tree that maximizes the sum of the edge weights.

Running time. $E \log E$ (or better).
**MINIMUM BOTTLENECK SPANNING TREE**

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

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

---

**Scientific application: clustering**

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

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

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

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

---

**Single-link clustering**

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

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

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

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

**Single-link clustering algorithm**

“Well-known” algorithm in science literature for single-link clustering:
- Form $V$ clusters of one object each.
- Find the closest pair of objects such that each object is in a different cluster, and merge the two clusters.
- Repeat until there are exactly $k$ clusters.

**Observation.** This is Kruskal’s algorithm.
(stopping when $k$ connected components)

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

Tumors in similar tissues cluster together.