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:

- Connected.
- Acyclic.
- Includes all of the vertices.

Minimum spanning tree

Def. A spanning tree of $G$ is a subgraph $T$ that is:

- Connected.
- Acyclic.
- Includes all of the vertices.
Minimum spanning tree

**Def.** A spanning tree of \( G \) is a subgraph \( T \) that is:

- Connected.
- Acyclic.
- Includes all of the vertices.

Minimum spanning tree problem

**Input.** Connected, undirected graph \( G \) with positive edge weights.

![edge-weighted graph G](image)

minimum spanning tree \( T \)

(weight = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7)

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

Minimum spanning tree

**Def.** A spanning tree of \( G \) is a subgraph \( T \) that is:

- Connected.
- Acyclic.
- Includes all of the vertices.
MST: 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/owedistrict/21380840

Models of nature

MST of random graph

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

Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research

http://www.bcrcc.ca/en/ta02_archlevel.html
4.3 MINIMUM SPANNING TREES

Simplifying assumptions

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

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, computer, road).

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.

MST: 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.

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.
**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.

![Greedy MST graph](image)

**MST edges**

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

---

**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 \( \Rightarrow \) cut with no black crossing edges.
  (consider cut whose vertices are any one connected component)

![MST graph with cut](image)

**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]
- **Ex 3.** Borůvka's algorithm.

---

**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 equal weights](image)

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

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

![Graph with disconnected components](image)
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.

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

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

    public int either()
    {
        return v;
    }

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

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

Weighted edge: Java implementation

Idiom for processing an edge e: int v = e.either(), w = e.other(v);
Conventions. Allow self-loops and parallel edges.

Edge-weighted graph API

- public class EdgeWeightedGraph
- EdgeWeightedGraph(int V)
- create an empty graph with V vertices
- 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

Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of Edge lists.

Minimum spanning tree API

Q. How to represent the MST?

- public class MST
  - constructor
  - Iterable<Edge> edges()
    - edges in MST
  - double weight()
    - weight of MST
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
class MST
{
    private EdgeWeightedGraph edges;
    private double weight;
    MST(EdgeWeightedGraph G)
    constructor
    for (Edge e : G)
    edges.add(e);
    double weight()
    weight of MST
}
```

Kruskal's algorithm demo

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

Kruskal's algorithm demo

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

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.16</td>
</tr>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>0-2</td>
<td>0.26</td>
</tr>
<tr>
<td>2-3</td>
<td>0.17</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>

- an edge-weighted graph
- a minimum spanning tree
Kruskal’s algorithm: visualization

Kruskal’s algorithm computes the MST.

Pf. Suppose Kruskal’s algorithm colors the edge $e = v\rightarrow w$ black.

- Cut $= \text{set of vertices connected to } v \text{ in tree } T$.
- No crossing edge is black.
- No crossing edge has lower weight. Why?

![Graph with edges colored]

Kruskal’s algorithm: correctness proof

Challenge. Would adding edge $v\rightarrow w$ 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

Efficient solution. Use the union-find data structure.

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

Add edge to tree

**Case 1:** adding $v\rightarrow w$ creates a cycle

**Case 2:** add $v\rightarrow w$ to $T$ and merge sets containing $v$ and $w$.
Kruskal's algorithm: Java implementation

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

    public KruskalMST(EdgeWeightedGraph G) {
        MinPQ<Edge> pq = new MinPQ<Edge>(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;
    }
}
```

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 $E \log* V$.

4.3 Minimum Spanning Trees

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

**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.

![Prim's algorithm demo](image)

an edge-weighted graph

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

Prim's algorithm demo

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.

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.
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 $e = v - w$ 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 to PQ any edge incident to $w$ (assuming other endpoint not in $T$)
  - add $e$ to $T$ and mark $w$

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

MST edges

0-7  1-7  0-2  2-3  5-7  4-5  6-2
**Prim's algorithm: 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;
}
```

**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>

**Prim's algorithm: eager implementation**

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

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

![an edge-weighted graph](image-url)
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.

```
v edgeTo[] distTo[]
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
4   4-5   0.35
6   6-2   0.40
```

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

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 and 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 $T$

Indexed priority queue

Associate an index between 0 and $N-1$ with each key in a priority queue.
- Supports insert and delete-the-minimum.
- Supports decrease-key given the index of the key.

```
public class IndexedMinPQ<Key extends Comparable<Key>>

    IndexMinPQ(int N) create indexed priority queue with indices 0, 1, ..., N-1
    void insert(int i, Key key) associate key with index i
    void decreaseKey(int i, Key key) decrease the key associated with index i
    boolean contains(int i) is i an index on the priority queue?
    int deleteMin() remove a minimal key and return its associated index
    boolean isEmpty() is the priority queue empty?
    int size() number of keys in the priority queue
```

Indexed priority queue implementation

Binary heap implementation. [see Section 2.4 of textbook]
- 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[i]) to implement decreaseKey(i, key).

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

black: on MST
red: on PQ
pq has at most one entry per vertex

pqY has at most one entry per vertex

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

black: on MST
red: on PQ
pqY has at most one entry per vertex

pqY has at most one entry per vertex

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

black: on MST
red: on PQ
pqY has at most one entry per vertex

pqY has at most one entry per vertex
**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>$E \log_{d+1} V$</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.

**Does a linear-time MST algorithm exist?**

<table>
<thead>
<tr>
<th>deterministic compare–based MST algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1975</td>
</tr>
<tr>
<td>1976</td>
</tr>
<tr>
<td>1984</td>
</tr>
<tr>
<td>1986</td>
</tr>
<tr>
<td>1997</td>
</tr>
<tr>
<td>2000</td>
</tr>
<tr>
<td>2002</td>
</tr>
<tr>
<td>20xx</td>
</tr>
</tbody>
</table>

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

**4.3 Minimum Spanning Trees**

- introduction
- 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 $N \log N$ time.
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 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.