4.3 Minimum Spanning Trees

- weighted graph API
- greedy algorithm
- Kruskal’s algorithm
- Prim’s algorithm
- advanced topics

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

- weighted graph API
- greedy algorithm
- Kruskal’s algorithm
- Prim’s algorithm
- advanced topics

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

- weighted graph API
- greedy algorithm
- Kruskal’s algorithm
- Prim’s algorithm
- advanced topics

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?

**NO!**

- **Reason 1:** How?
- **Reason 2:** There are $V^{V-2}$ of them.

**Applications**

MST is fundamental problem with diverse applications.

- 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.
- Network design (communication, electrical, hydraulic, cable, computer, road).
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).

Network design

*Network design* is fundamental problem with diverse applications.

- 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.
- Network design (communication, electrical, hydraulic, cable, computer, road).
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).

Medical image processing

*MST describes arrangement of nuclei in the epithelium for cancer research*.

Medical image processing
Genetic research

MST of tissue relationships measured by gene expression correlation coefficient

![Image of tissue relationship network](http://riodb.ibase.aist.go.jp/CELLPEDIA)

Edge API

Edge abstraction needed for weighted edges.

```java
public class Edge
{
    private int v, w;
    private 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 v)
    {
        return v == w ? w : v;
    }

    public double weight()
    {
        return weight;
    }
}
```

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

Edge-weighted graph API

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

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

    public void addEdge(Edge e)
    {
        weight[e.v][e.w] = e.weight;
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.other(e.v));
    }

    public void addEdge(EdgeWeightedGraph e)
    {
        addEdge(e.either(), e.o
### Edge-weighted graph API

**public class EdgeWeightedGraph**

- `EdgeWeightedGraph(int V)` create an empty graph with `V` vertices
- `EdgeWeightedGraph(Edge e)` create a graph from input stream
- `void addEdge(Edge e)` add edge `v`
- `Iterable<Edge> adj(int v)` return an iterator over edges incident to `v`
- `int V()` return number of vertices
- `int E()` return number of edges

```java
define the constructor and methods for EdgeWeightedGraph
```

```java
for (int v = 0; v < G.V(); v++)
    for (Edge e : G.adj(v))
        int w = e.other(v);
        // process edge v-w
```

### Weighted graph: adjacency-lists implementation

**public class EdgeWeightedGraph**

```java
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];  }
}
```

### Weighted edge: Java implementation

**public class Edge**

```java
public class 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 weight()   {  return weight;  }
}
```

### Weighted edge comparator

**private static class ByWeight implements Comparator<Edge>**

```java
private static class ByWeight implements Comparator<Edge> {    public int compare(Edge e, Edge f)    {       if (e.weight() < f.weight()) return -1;       if (e.weight() > f.weight()) return +1;       return 0;    } }
```

**Clients need to compare edge weights.**

```java
private static class ByWeight implements Comparator<Edge> {    public int compare(Edge e, Edge f)    {       if (e.weight() < f.weight()) return -1;       if (e.weight() > f.weight()) return +1;       return 0;    } }
```

**Note: different clients may use different Comparator implementations**

```java
Clients need to compare edge weights.
```
**Edge-weighted graph: adjacency-list representation**

Maintain vertex-indexed array of Edge lists (use Bag abstraction)

![Diagram of edge-weighted graph with adjacency-list representation](image)

---

**Cut property**

**Simplifying assumption.** Edge weights are different.

**Cut property.** Given any cut, the minimum-weight crossing edge is in the MST.

![Diagram illustrating cut property](image)

---

**Cut property: correctness proof**

**Simplifying assumption.** Edge weights are different.

**Cut property.** Given any cut, the minimum-weight crossing edge is in the MST.

**Pf.**

Let $e$ be the min-weight crossing edge

- Suppose $e$ is not in the MST.
- Adding to the MST $e$ creates a cycle $C$.
- Some other edge $f$ in $C$ must be a crossing edge.
- Removing $f$ and adding $e$ is also a spanning tree.
- Since $w_e < w_f$, that spanning tree is lower weight.
- Contradiction. $\blacksquare$
Greedy MST algorithm

Greedy algorithm. The following method computes the MST:
• start with all edges colored gray
• find a cut having no black edges
• color its minimum-weight edge black
• continue until V-1 edges are colored black

Proof.
Any black edge is in the MST, by the cut property.
Once we have V-1 of them, we have the MST.

Two special cases of the greedy algorithm

Kruskal’s algorithm. Consider edges in ascending order of weight.
Color black the next edge unless doing so would create a cycle.

Prim’s algorithm. Start with any vertex s and greedily grow a tree T from s.
At each step, add to T the edge of min weight with exactly one endpoint in T.

Proposition. Both algorithms compute MST.
Proof. Vertices touched by black edges define a cut.
Kruskal’s algorithm

Kruskal’s algorithm. [Kruskal 1956] Consider edges in ascending order of weight. Add to T the next edge unless doing so would create a cycle.

Problem. Check if adding an edge v-w to T creates a cycle.

How difficult?
- O(E + V) time.
- O(V) time.
- O(log V) time.
- O(log* V) time.
- Constant time.

Kruskal implementation challenge

Problem. Check if adding an edge v-w to T creates a cycle.

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

Case 1: adding v-w creates a cycle
Case 2: add v-w to T and merge sets
Kruskal’s algorithm: Java implementation

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

    public KruskalMST(WeightedGraph G) {
        pq = new MinPQ<Edge>(G.edges(), new ByWeight());
        UnionFind uf = new UnionFind(G.V());
        while (!pq.isEmpty() && mst.size() < G.V()-1) {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (!uf.find(v, w)) { // Edge v-w does not create a cycle.
                uf.union(v, w); // Merge components.
                mst.enqueue(e); // Add edge to mst.
            }
        }
    }

    public Iterable<Edge> mst() {
        return mst;
    }
}
```

Kruskal’s algorithm running time

**Proposition.** Kruskal’s algorithm computes MST in \(O(E \log E)\) time.

**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>del min</td>
<td>(E)</td>
<td>(\log E)</td>
</tr>
<tr>
<td>union</td>
<td>(V)</td>
<td>(\log^* V)</td>
</tr>
<tr>
<td>find</td>
<td>(E)</td>
<td>(\log^* V)</td>
</tr>
</tbody>
</table>

† Amortized bound using weighted quick union with path compression.

**Improvements.**
- If edges are already sorted, worst case time is \(\sim E \log^* V\).
- Stop as soon as \(V-1\) edges on MST: only a fraction of edges leave \(pq\).
Prim’s algorithm example

Prim’s algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]
Start with vertex 0 and greedily grow tree T. At each step, add to T the edge of min weight with exactly one endpoint in T.

Problem. Find min weight edge with exactly one endpoint in S.

Efficient solution. Maintain a PQ of edges with (at least) one endpoint in S.
• Delete min to determine next edge e = v-w to add to T.
• Disregard if both v and w are in S.
• Let w be vertex not in S:
  - add to PQ any edge incident to w (assuming other endpoint not in S)
  - add w to S

Use PQ: key = edge.
(lazy version leaves some obsolete entries on the PQ)
public class LazyPrimMST
{
    private boolean[] marked;    // vertices in MST
    private Queue<Edge> mst;      // edges in the MST
    private MinPQ<Edge> pq;       // the priority queue of edges

    public LazyPrimMST(WeightedGraph G)
    {
        marked = new boolean[G.V()];
        mst = new Queue<Edge>();
        pq = new MinPQ<Edge>(Edge.ByWeight());
        prim(G, 0);
    }

    public Iterable<Edge> mst()
    { return mst; }

    // See next slide for prim() implementation.
}

37

Lazy implementation of Prim’s algorithm

private void visit(WeightedGraph G, int v)
{
    marked[v] = true;
    for (Edge e : G.adj(v))
        if (!marked[e.other(v)])
            pq.insert(e);
}

private void prim(WeightedGraph G, int s)
{
    visit(G, s);
    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;
        mst.enqueue(e);
        if (!marked[v]) visit(G, v);
        if (!marked[w]) visit(G, w);
    }
}

38

Prim’s algorithm running time

Proposition. Prim’s algorithm computes MST in \(O(E \log E)\) time.
Pf.

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete min</td>
<td>V</td>
<td>(\lg E)</td>
</tr>
<tr>
<td>insert</td>
<td>E</td>
<td>(2 \lg E)</td>
</tr>
</tbody>
</table>

Improvements.

- Eagerly eliminate obsolete edges from PQ.
- Maintain on PQ at most one edge incident to each vertex \(v\) not in \(T\) ⇒ at most \(V\) edges on PQ.
- Use fancier priority queue: best in theory yields \(O(E + V \log V)\).
Simplifying assumption. All edge weights are distinct.

Solution. Prim and Kruskal still find MST if equal weights are present! (only our proof of correctness fails, and that can be fixed)

Removing the distinct edge weight assumption

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 $\sim c N \log N$.

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

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-Calli-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>
**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.

![Image of clusters](image1.png)

**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.

![Dendrogram](image2.png)

**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](image3.png)

**Dendrogram.** Tree diagram that illustrates arrangement of clusters.

![Map of Italy](image4.png)
Dendrogram. Tree diagram that illustrates arrangement of clusters.
Dendrogram. Tree diagram that illustrates arrangement of clusters.

Dendrogram of cancers in human

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group

Gene 1

Gene n

Skin  Liver  Lung  Breast Tumors  Lymphoid  Breast  Prostate  Brain  APL  Dray

Gene expressed
Gene not expressed