

Minimum Spanning Trees

- ▶ weighted graph API
- ▶ cycles and cuts
- ▶ Kruskal's algorithm
- ▶ Prim's algorithm
- ▶ advanced topics

References:

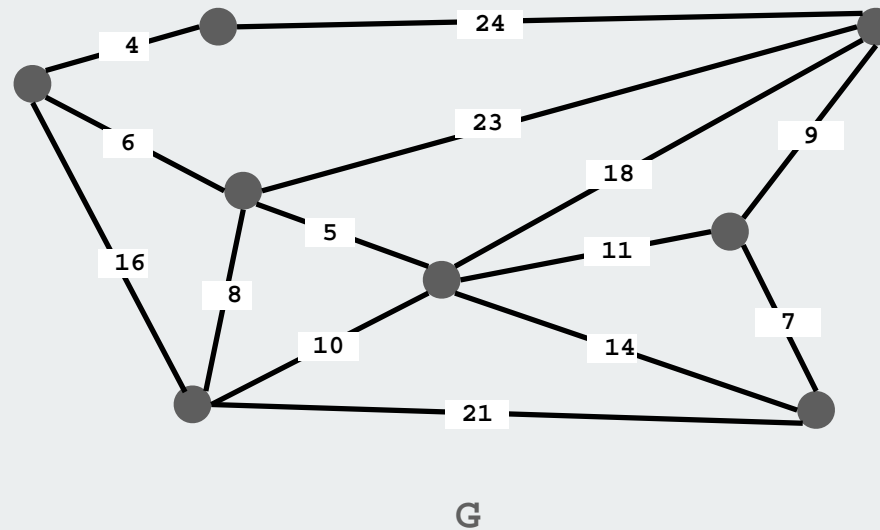
Algorithms in Java, Chapter 20

<http://www.cs.princeton.edu/introalgsds/54mst>

Minimum Spanning Tree

Given. Undirected graph G with positive edge weights (connected).

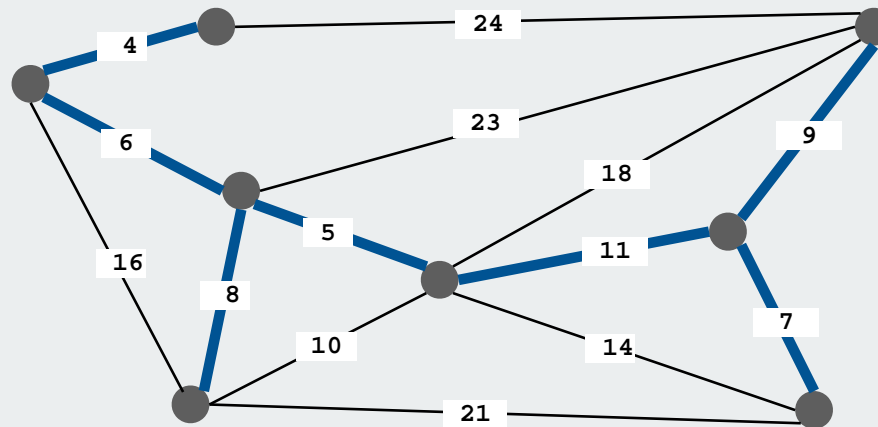
Goal. Find a min weight set of edges that connects all of the vertices.



Minimum Spanning Tree

Given. Undirected graph G with positive edge weights (connected).

Goal. Find a min weight set of edges that connects all of the vertices.



$$\text{weight}(T) = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$$

Brute force: Try all possible spanning trees

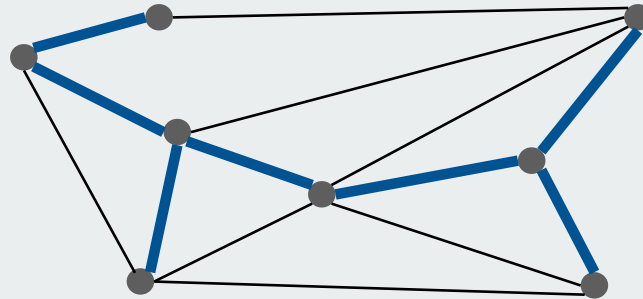
- problem 1: not so easy to implement
- problem 2: far too many of them

Ex: [Cayley, 1889]: V^{V-2} spanning trees on the complete graph on V vertices.

MST Origin

Otakar Boruvka (1926).

- Electrical Power Company of Western Moravia in Brno.
- Most economical construction of electrical power network.
- Concrete engineering problem is now a cornerstone problem-solving model in combinatorial optimization.



Otakar Boruvka

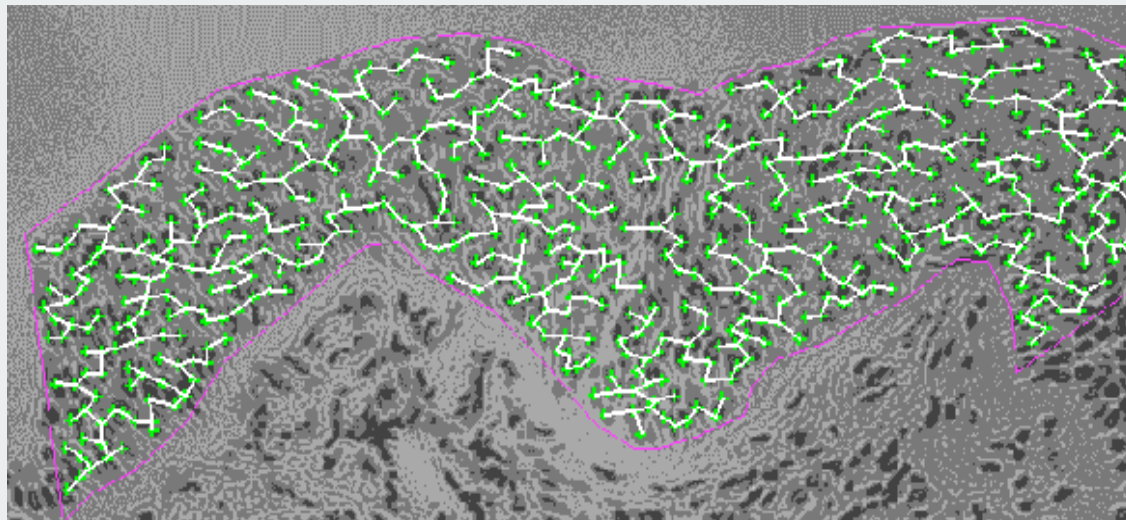
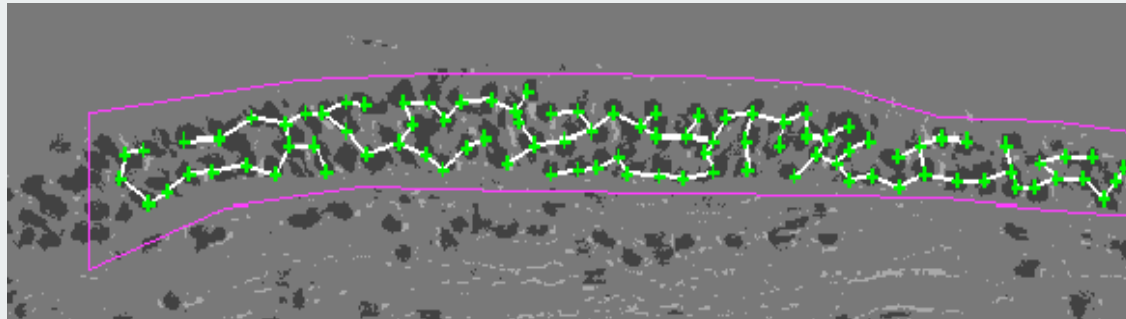
Applications

MST is fundamental problem with diverse applications.

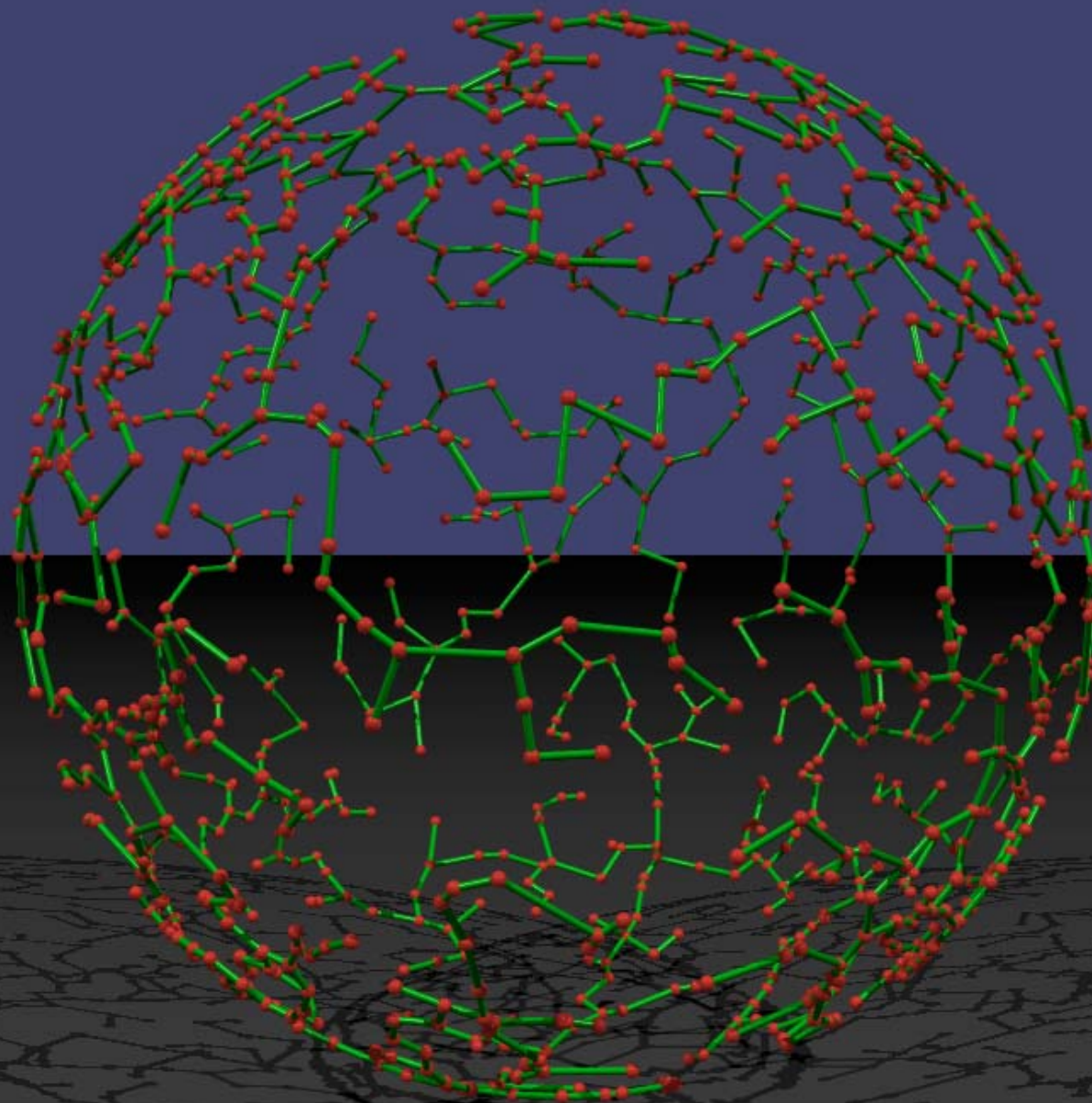
- Network design.
telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
traveling salesperson problem, Steiner tree
- Indirect applications.
max bottleneck paths
LDPC codes for error correction
image registration with Renyi entropy
learning salient features for real-time face verification
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
- Cluster analysis.

Medical Image Processing

MST describes arrangement of nuclei in the epithelium for cancer research



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



Two Greedy Algorithms

Kruskal's algorithm. Consider edges in ascending order of cost. Add the next edge to T 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 the cheapest edge to T that has exactly one endpoint in T .

Proposition. Both greedy algorithms compute an MST.

Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit." - *Gordon Gecko*



▶ weighted graph API

- ▶ cycles and cuts
- ▶ Kruskal's algorithm
- ▶ Prim's algorithm
- ▶ advanced topics

Weighted Graph API

```
public class WeightedGraph
```

| | |
|--|---|
| <code>WeightedGraph(int V)</code> | create an empty graph with V vertices |
| <code>void insert(Edge e)</code> | insert edge e |
| <code>Iterable<Edge> adj(int v)</code> | return an iterator over edges incident to v |
| <code>int V()</code> | return the number of vertices |
| <code>String toString()</code> | return a string representation |

iterate through all edges (once in each direction)

Weighted graph data type

Identical to `Graph.java` but use `Edge` adjacency sets instead of `int`.

```
public class WeightedGraph
{
    private int V;
    private SET<Edge>[] adj;

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

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

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

Weighted edge data type

```
public class Edge implements Comparable<Edge>
{
    private final int v, 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 weight()
    { return weight; }

    // See next slide for edge compare methods.
}
```

Edge abstraction needed for weights

slightly tricky accessor methods
(enables client code like this)

```
for (int v = 0; v < G.V(); v++)
{
    for (Edge e : G.adj(v))
    {
        int w = e.other(v);

        // edge v-w
    }
}
```

Weighted edge data type: compare methods

Two different compare methods for edges

- `compareTo()` so that edges are comparable (for use in `SET`)
- `compare()` so that clients can compare edges by weight.

```
public final static Comparator<Edge> BY_WEIGHT = new ByWeightComparator();

private static class ByWeightComparator 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;
    }
}

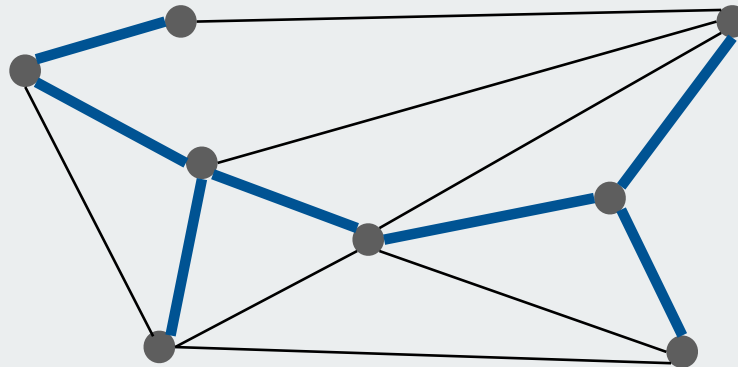
public int compareTo(Edge that)
{
    if (this.weight < that.weight) return -1;
    else if (this.weight > that.weight) return +1;
    else return 0;
}
```

- ▶ weighted graph API
- ▶ **cycles and cuts**
- ▶ Kruskal's algorithm
- ▶ Prim's algorithm
- ▶ advanced topics

Spanning Tree

MST. Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.

Def. A **spanning tree** of a graph G is a subgraph T that is connected and acyclic.



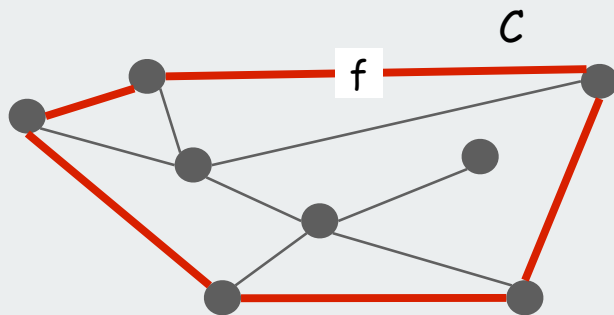
Property. MST of G is always a spanning tree.

Greedy Algorithms

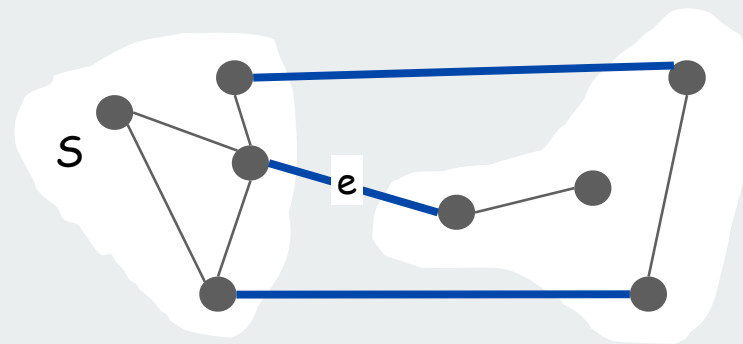
Simplifying assumption. All edge weights w_e are distinct.

Cycle property. Let C be any cycle, and let f be the **max cost** edge belonging to C . Then the MST does not contain f .

Cut property. Let S be any subset of vertices, and let e be the **min cost** edge with exactly one endpoint in S . Then the MST contains e .



f is not in the MST



e is in the MST

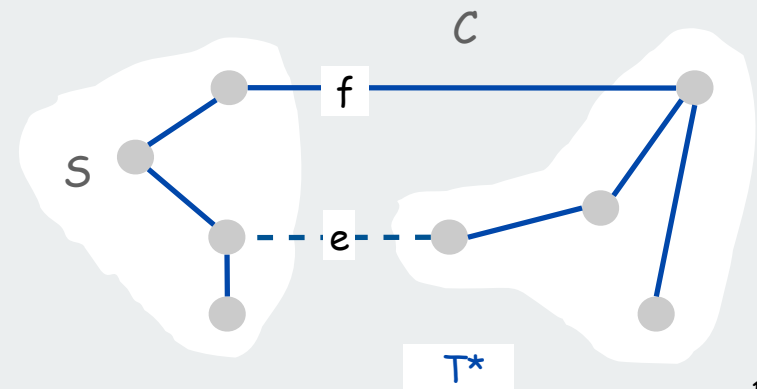
Cycle Property

Simplifying assumption. All edge weights w_e are distinct.

Cycle property. Let C be any cycle, and let f be the **max cost** edge belonging to C . Then the MST T^* does not contain f .

Pf. [by contradiction]

- Suppose f belongs to T^* . Let's see what happens.
- Deleting f from T^* disconnects T^* . Let S be one side of the cut.
- Some other edge in C , say e , has exactly one endpoint in S .
- $T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.
- Contradicts minimality of T^* . ■



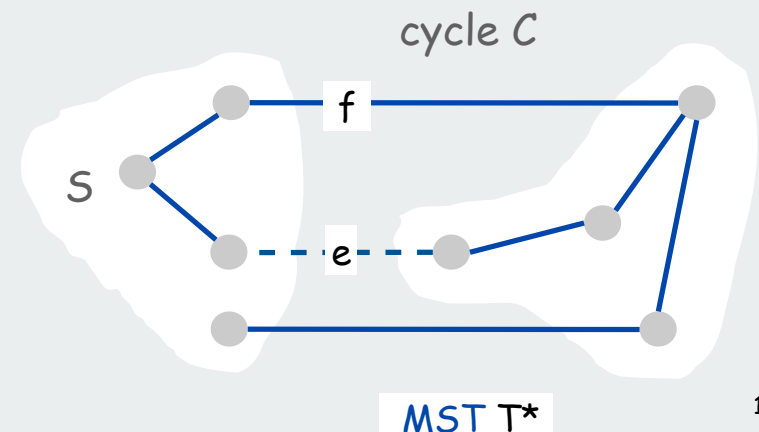
Cut Property

Simplifying assumption. All edge costs c_e are distinct.

Cut property. Let S be any subset of vertices, and let e be the **min cost** edge with exactly one endpoint in S . Then the MST T^* contains e .

Pf. [by contradiction]

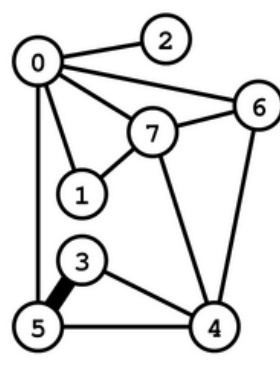
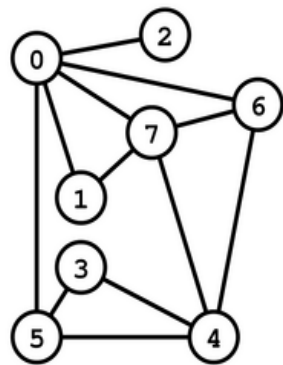
- Suppose e does not belong to T^* . Let's see what happens.
- Adding e to T^* creates a (unique) cycle C in T^* .
- Some other edge in C , say f , has exactly one endpoint in S .
- $T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.
- Contradicts minimality of T^* . ■



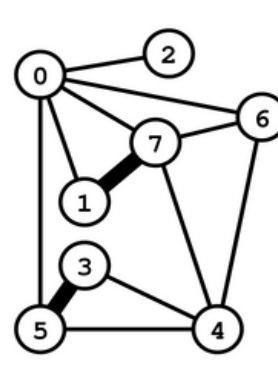
- ▶ weighted graph API
- ▶ cycles and cuts
- ▶ **Kruskal's algorithm**
- ▶ Prim's algorithm
- ▶ advanced algorithms
- ▶ clustering

Kruskal's Algorithm: Example

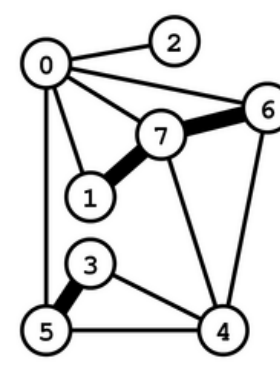
Kruskal's algorithm. [Kruskal, 1956] Consider edges in ascending order of cost. Add the next edge to T unless doing so would create a cycle.



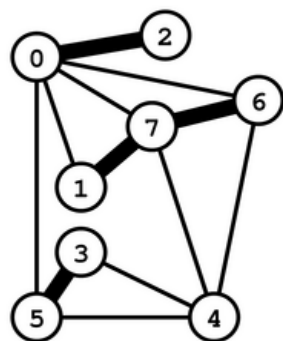
3-5



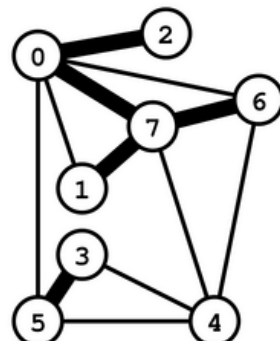
1-7



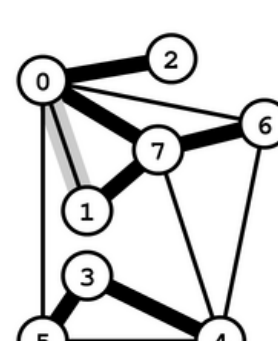
6-7



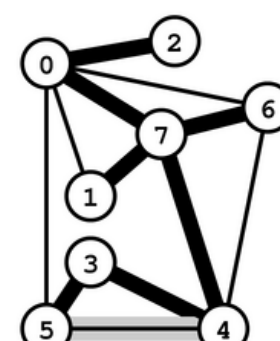
0-2



0-7



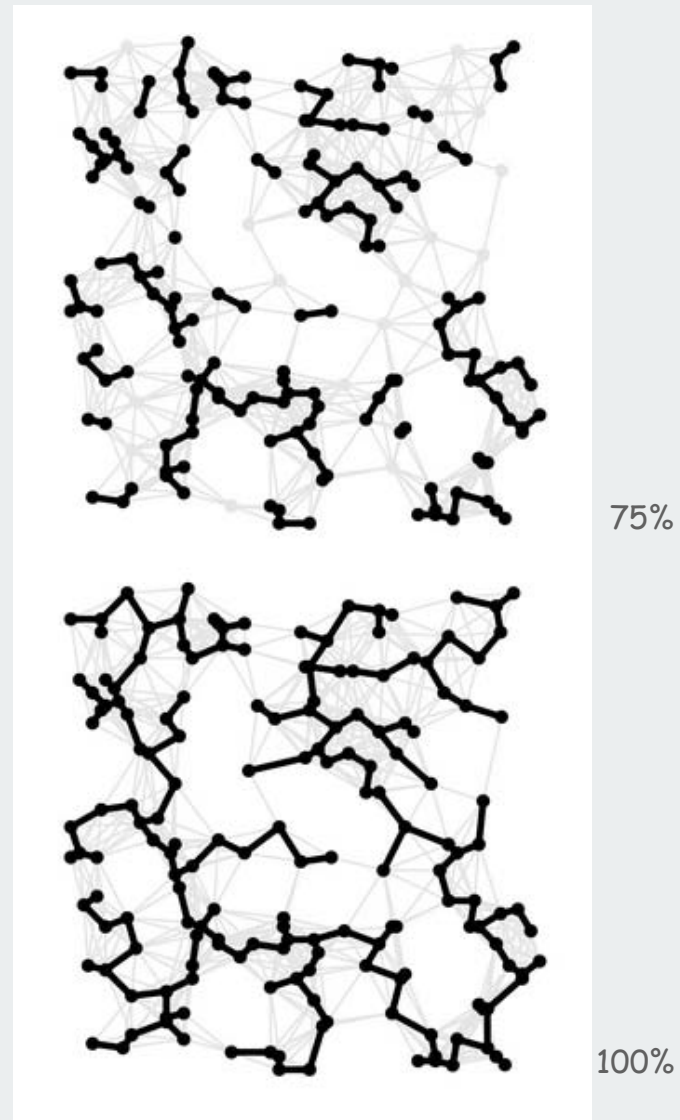
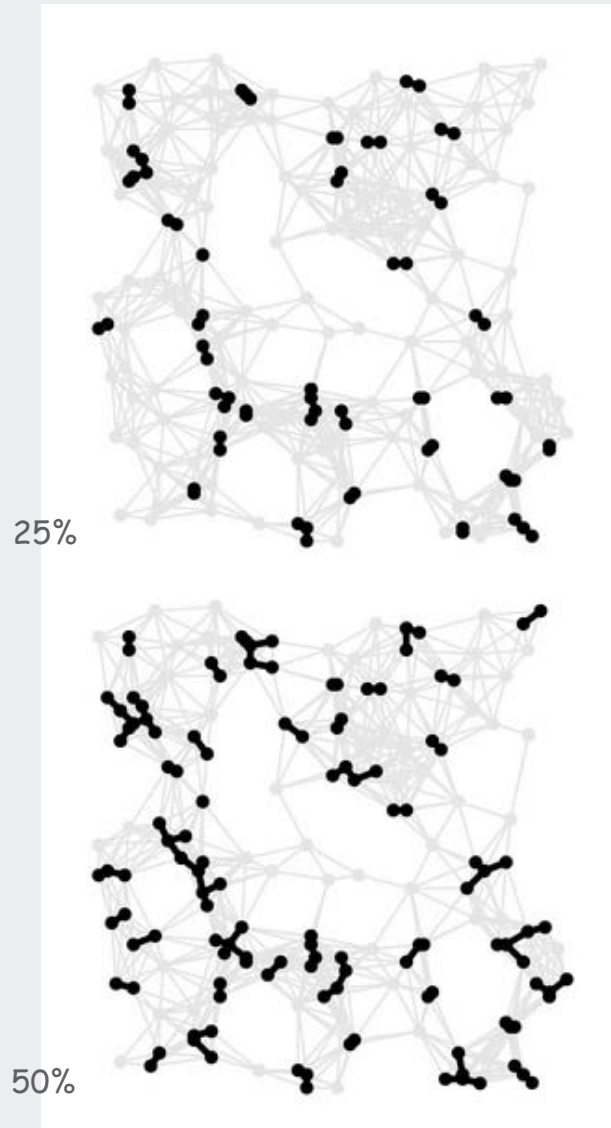
0-1 3-4



4-5 4-7

| | |
|-----|------|
| 3-5 | 0.18 |
| 1-7 | 0.21 |
| 6-7 | 0.25 |
| 0-2 | 0.29 |
| 0-7 | 0.31 |
| 0-1 | 0.32 |
| 3-4 | 0.34 |
| 4-5 | 0.40 |
| 4-7 | 0.46 |
| 0-6 | 0.51 |
| 4-6 | 0.51 |
| 0-5 | 0.60 |

Kruskal's algorithm example

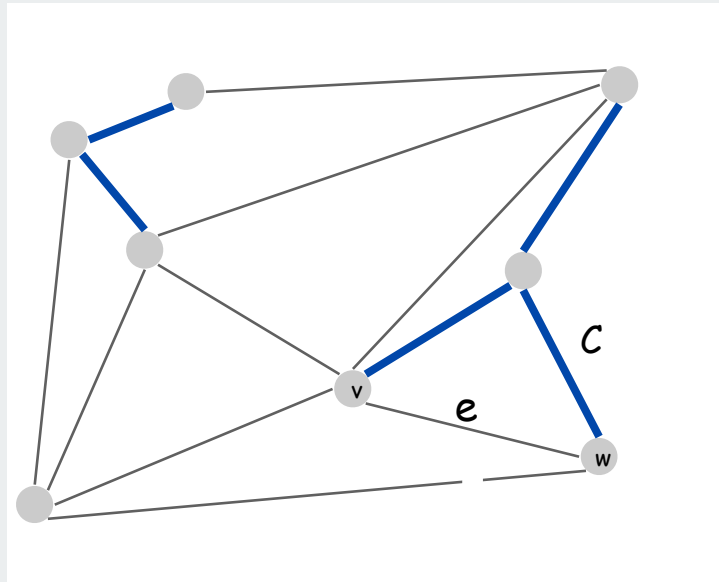


Kruskal's algorithm correctness proof

Proposition. Kruskal's algorithm computes the MST.

Pf. [case 1] Suppose that adding e to T creates a cycle C

- e is the max weight edge in C (weights come in increasing order)
- e **is not** in the MST (cycle property)

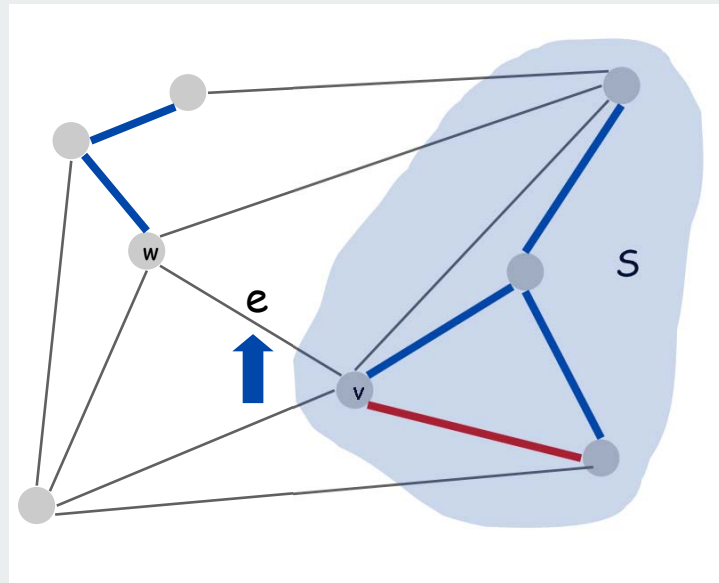


Kruskal's algorithm correctness proof

Proposition. Kruskal's algorithm computes the MST.

Pf. [case 2] Suppose that adding $e = (v, w)$ to T does not create a cycle

- let S be the vertices in v 's connected component
- w is not in S
- e is the min weight edge with exactly one endpoint in S
- e **is** in the MST (cut property) ■



Kruskal's algorithm implementation

Q. How to check if adding an edge to T would create a cycle?

A1. Naïve solution: use DFS.

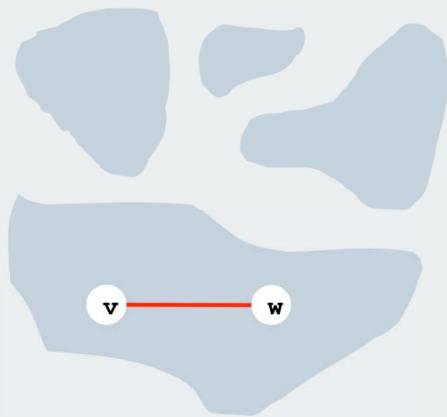
- $O(V)$ time per cycle check.
- $O(E V)$ time overall.

Kruskal's algorithm implementation

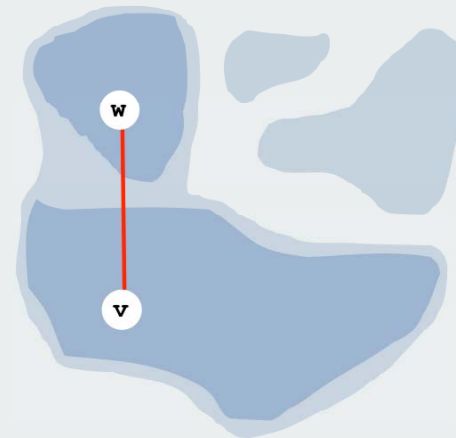
Q. How to check if adding an edge to T would create a cycle?

A2. Use the **union-find** data structure from lecture 1 (!).

- Maintain a set for each connected component.
- 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

```
public class Kruskal
{
    private SET<Edge> mst = new SET<Edge>();

    public Kruskal(WeightedGraph G)
    {
        Edge[] edges = G.edges();
        Arrays.sort(edges, Edge.BY_WEIGHT);

        UnionFind uf = new UnionFind(G.V());
        for (Edge e: edges)
            if (!uf.find(e.either(), e.other()))
            {
                uf.unite(e.either(), e.other());
                mst.add(e);
            }
    }

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

sort edges
by weight

greedily add
edges to MST

return to client iterable
sequence of edges

Easy speedup: Stop as soon as there are $V-1$ edges in MST.

Kruskal's algorithm running time

Kruskal running time: Dominated by the cost of the sort.

| Operation | Frequency | Time per op |
|-----------|-----------|--------------------|
| sort | 1 | $E \log E$ |
| union | V | $\log^* V^\dagger$ |
| find | E | $\log^* V^\dagger$ |

\dagger amortized bound using weighted quick union with path compression

recall: $\log^* V \leq 5$ in this universe

Remark 1. If edges are already sorted, time is proportional to $E \log^* V$

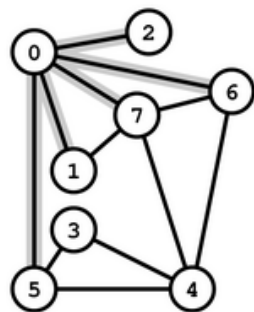
Remark 2. Linear in practice with PQ or quicksort partitioning
(see book: don't need full sort)

- ▶ weighted graph API
- ▶ cycles and cuts
- ▶ Kruskal's algorithm
- ▶ **Prim's algorithm**
- ▶ advanced topics

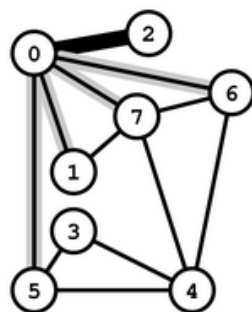
Prim's algorithm example

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

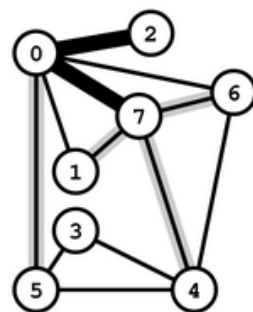
Start with vertex 0 and greedily grow tree T. At each step, add cheapest edge that has exactly one endpoint in T.



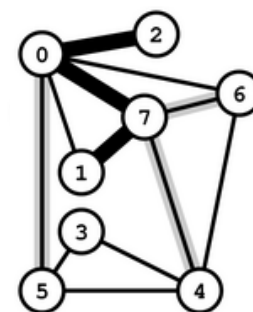
0-2 0-7 0-1 0-6 0-5



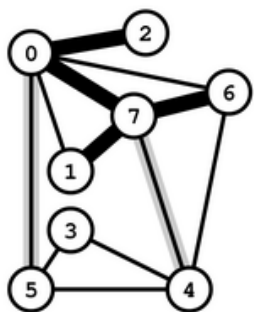
0-7 0-1 0-6 0-5



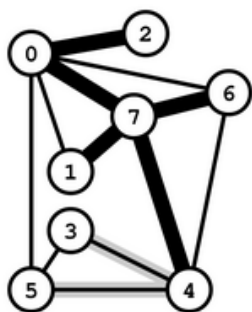
7-1 7-6 7-4 0-5



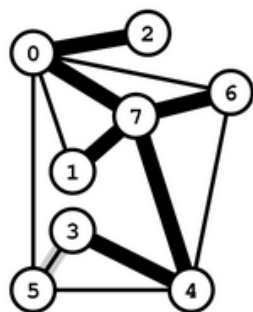
7-6 7-4 0-5



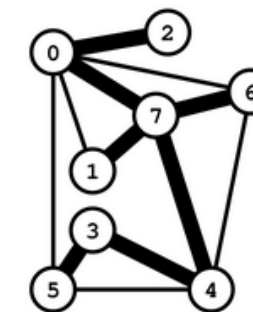
7-4 0-5



4-3 4-5



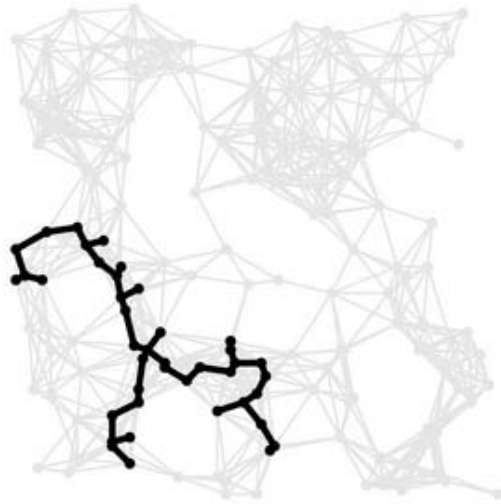
3-5



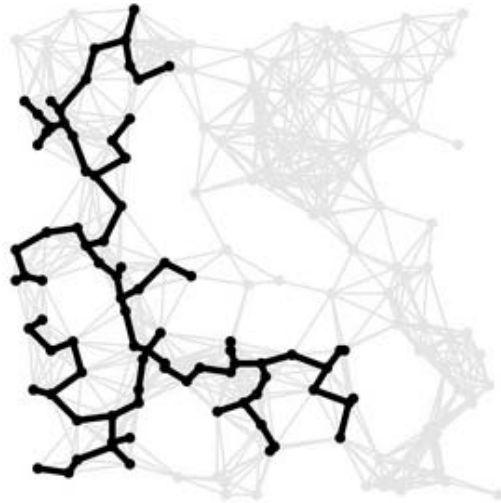
| | |
|-----|------|
| 0-1 | 0.32 |
| 0-2 | 0.29 |
| 0-5 | 0.60 |
| 0-6 | 0.51 |
| 0-7 | 0.31 |
| 1-7 | 0.21 |
| 3-4 | 0.34 |
| 3-5 | 0.18 |
| 4-5 | 0.40 |
| 4-6 | 0.51 |
| 4-7 | 0.46 |
| 6-7 | 0.25 |

Prim's Algorithm *example*

25%



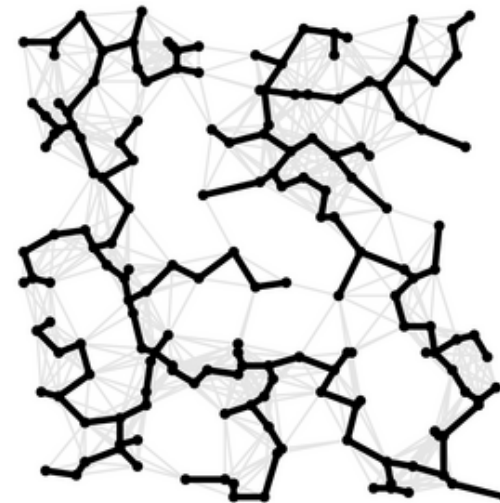
50%



75%



100%

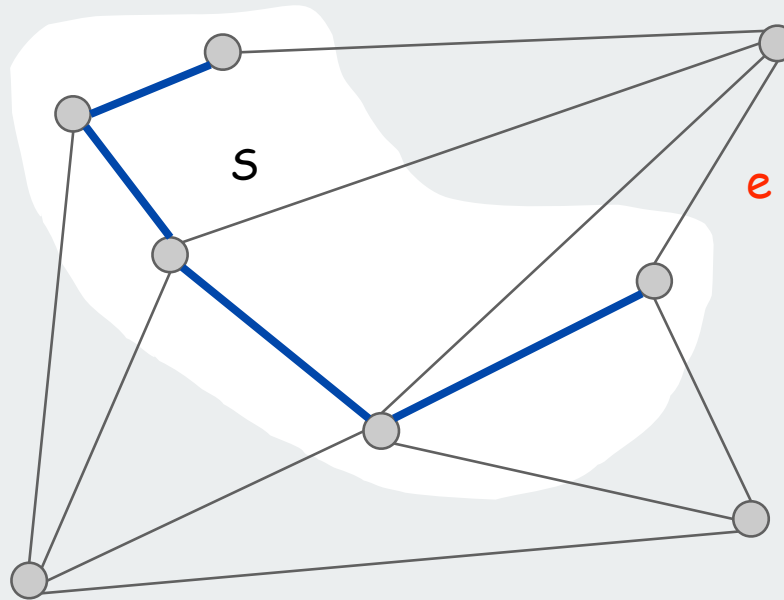


Prim's algorithm correctness proof

Proposition. Prim's algorithm computes the MST.

Pf.

- Let S be the subset of vertices in current tree T .
- Prim adds the cheapest edge e with exactly one endpoint in S .
- e is in the MST (cut property) ■



Prim's algorithm implementation

Q. How to find cheapest edge with exactly one endpoint in S ?

A1. Brute force: try all edges.

- $O(E)$ time per spanning tree edge.
- $O(E V)$ time overall.

Prim's algorithm implementation

Q. How to find cheapest edge with exactly one endpoint in S ?

A2. Maintain a **priority queue** of vertices connected by an edge to S

- Delete min to determine next vertex v to add to S .
- Disregard v if already in S .
- Add to PQ any vertex brought closer to S by v .

Running time.

- $\log V$ steps per edge (using a binary heap).
- $E \log V$ steps overall.

Note: This is a **lazy** version of implementation in Algs in Java

lazy: put all adjacent vertices (that are not already in MST) on PQ
eager: first check whether vertex is already on PQ and decrease its key

Key-value priority queue

Associate a value with each key in a priority queue.

API:

| | |
|---|--|
| <code>public class MinPQplus<Key extends Comparable<Key>, Value></code> | |
| <code>MinPQplus()</code> | create a key-value priority queue |
| <code>void put(Key key, Value val)</code> | put key-value pair into the priority queue |
| <code>Value delMin()</code> | return value paired with minimal key |
| <code>Key min()</code> | return minimal key |

Implementation:

- start with same code as standard heap-based priority queue
- use a parallel array `vals[]` (value associated with `keys[i]` is `vals[i]`)
- modify `exch()` to maintain parallel arrays (do `exch` in `vals[]`)
- modify `delMin()` to return value
- add `min()` (just returns `keys[1]`)

Lazy implementation of Prim's algorithm

```
public class LazyPrim
{
```

```
    Edge[] pred = new Edge[G.V()];
    public LazyPrim(WeightedGraph G)
    {
```

```
        boolean[] marked = new boolean[G.V()];
        double[] dist = new double[G.V()];
        MinPQplus<Double, Integer> pq;
        pq = new MinPQplus<Double, Integer>();
        dist[s] = 0.0;
```

```
        marked[s] = true;
        pq.put(dist[s], s);
        while (!pq.isEmpty())
        {
```

```
            int v = pq.delMin();
            if (marked[v]) continue;
            marked[v] = true;
            for (Edge e : G.adj(v))
            {
```

```
                int w = e.other(v);
                if (!done[w] && (dist[w] > e.weight()))
                {
                    dist[w] = e.weight(); pred[w] = e;
                    pq.insert(dist[w], w);
                }
            }
        }
    }
}
```

← `pred[v]` is edge
attaching `v` to MST

← marks vertices in MST
← distance to MST

← key-value PQ

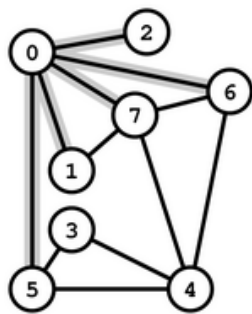
← get next vertex
← ignore if already in MST

← add to PQ any vertices
brought closer to `S` by `v`

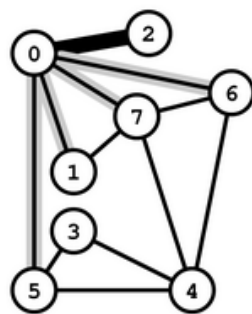
Prim's algorithm (lazy) example

Priority queue **key** is distance (edge weight); **value** is vertex

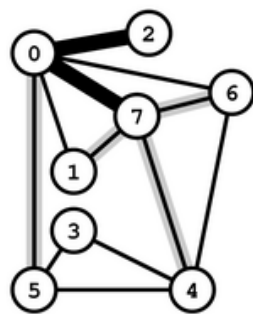
Lazy version leaves obsolete entries in the PQ
therefore may have multiple entries with same value



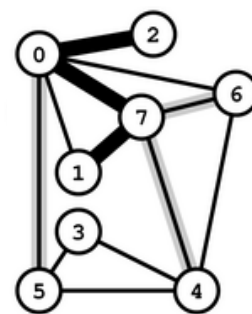
0-2 0-7 0-1 0-6 0-5



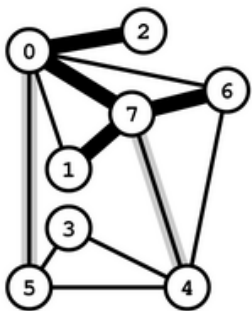
0-7 0-1 0-6 0-5



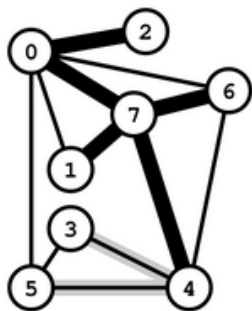
7-1 7-6 0-1 7-4 0-6 0-5



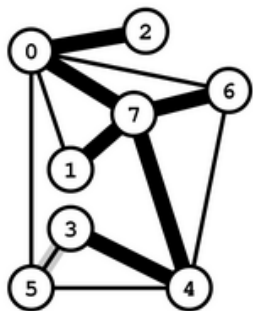
7-6 0-1 7-4 0-6 0-5



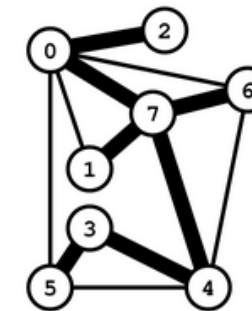
0-1 7-4 0-6 0-5



4-3 4-5 0-6 0-5



3-5 4-5 0-6 0-5



| | |
|-----|------|
| 0-1 | 0.32 |
| 0-2 | 0.29 |
| 0-5 | 0.60 |
| 0-6 | 0.51 |
| 0-7 | 0.31 |
| 1-7 | 0.21 |
| 3-4 | 0.34 |
| 3-5 | 0.18 |
| 4-5 | 0.40 |
| 4-6 | 0.51 |
| 4-7 | 0.46 |
| 6-7 | 0.25 |

red: pq value (vertex)
blue: obsolete value

Eager implementation of Prim's algorithm

Use **indexed priority queue** that supports

- contains: is there a key associated with value v in the priority queue?
- decrease key: decrease the key associated with value v

[more complicated data structure, see text]

Putative "benefit": reduces PQ size guarantee from E to V

- not important for the huge sparse graphs found in practice
- PQ size is far smaller in practice
- widely used, but practical utility is debatable

Removing the distinct edge costs assumption

Simplifying assumption. All edge weights w_e are distinct.

Fact. Prim and Kruskal don't actually rely on the assumption (our proof of correctness does)

Suffices to introduce tie-breaking rule for `compare()`.

Approach 1:

```
public int compare(Edge e, Edge f)
{
    if (e.weight < f.weight) return -1;
    if (e.weight > f.weight) return +1;
    if (e.v < f.v) return -1;
    if (e.v > f.v) return +1;
    if (e.w < f.w) return -1;
    if (e.w > f.w) return +1;
    return 0;
}
```

Approach 2: add tiny random perturbation.

- ▶ weighted graph API
- ▶ cycles and cuts
- ▶ Kruskal's algorithm
- ▶ Prim's algorithm
- ▶ advanced topics

Advanced MST theorems: does an algorithm with a linear-time guarantee exist?

| Year | Worst Case | Discovered By |
|------|------------------------------|----------------------------|
| 1975 | $E \log \log V$ | Yao |
| 1976 | $E \log \log V$ | Cheriton-Tarjan |
| 1984 | $E \log^* V, E + V \log V$ | Fredman-Tarjan |
| 1986 | $E \log (\log^* V)$ | Gabow-Galil-Spencer-Tarjan |
| 1997 | $E \alpha(V) \log \alpha(V)$ | Chazelle |
| 2000 | $E \alpha(V)$ | Chazelle |
| 2002 | optimal | Pettie-Ramachandran |
| 20xx | E | ??? |

deterministic comparison based MST algorithms



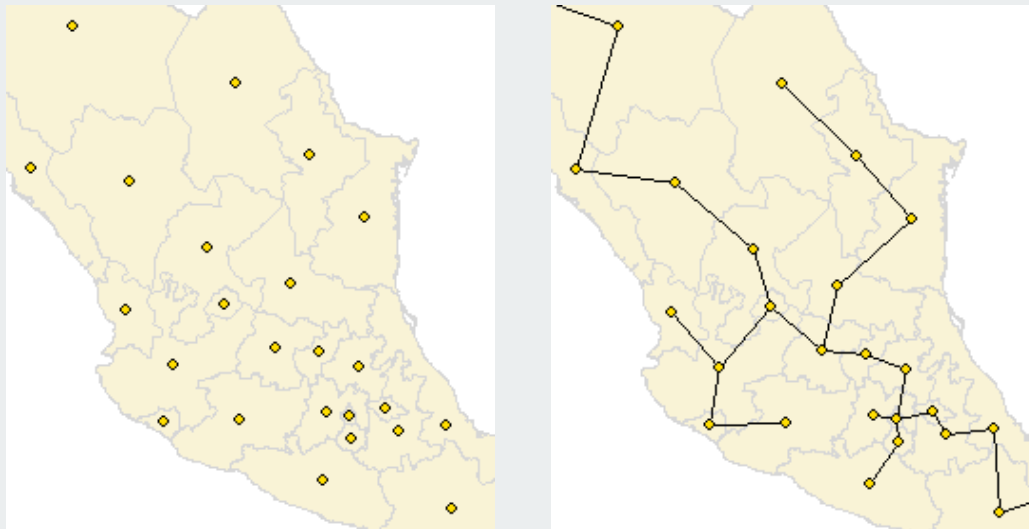
| Year | Problem | Time | Discovered By |
|------|------------------|------|---------------------|
| 1976 | Planar MST | E | Cheriton-Tarjan |
| 1992 | MST Verification | E | Dixon-Rauch-Tarjan |
| 1995 | Randomized MST | E | Karger-Klein-Tarjan |

related problems

Euclidean MST

Euclidean MST. Given N points in the plane, find MST connecting them.

- Distances between point pairs are **Euclidean** distances.



Brute force. Compute $N^2 / 2$ distances and run Prim's algorithm.

Ingenuity. Exploit geometry and do it in $O(N \log N)$
[stay tuned for geometric algorithms]

Scientific application: clustering

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

distance function. numeric value specifying "closeness" of two objects.

Fundamental problem.

Divide into clusters so that points in different clusters are far apart.

Applications.

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

Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

k-clustering of maximum spacing

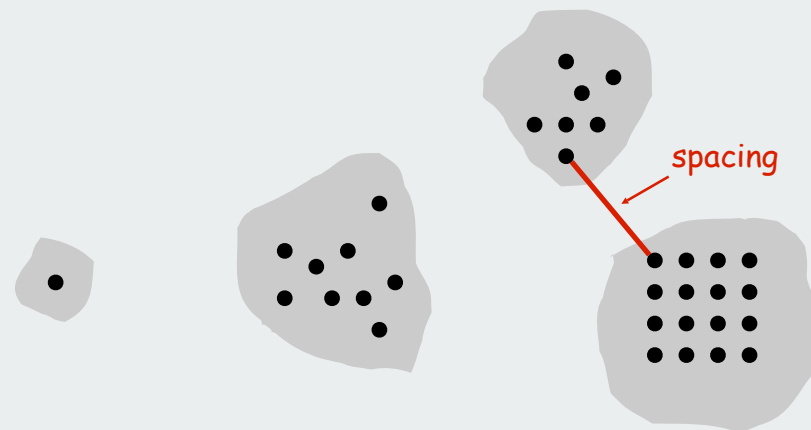
k-clustering. Divide a set of objects classify into k coherent groups.

distance function. Numeric value specifying "closeness" of two objects.

Spacing. Min distance between any pair of points in different clusters.

k-clustering of maximum spacing.

Given an integer k , find a k -clustering such that spacing is maximized.



$k = 4$

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 add an edge between them.
- Repeat until there are exactly k clusters.

Observation. This procedure is **precisely** Kruskal's algorithm (stop when there are k connected components).

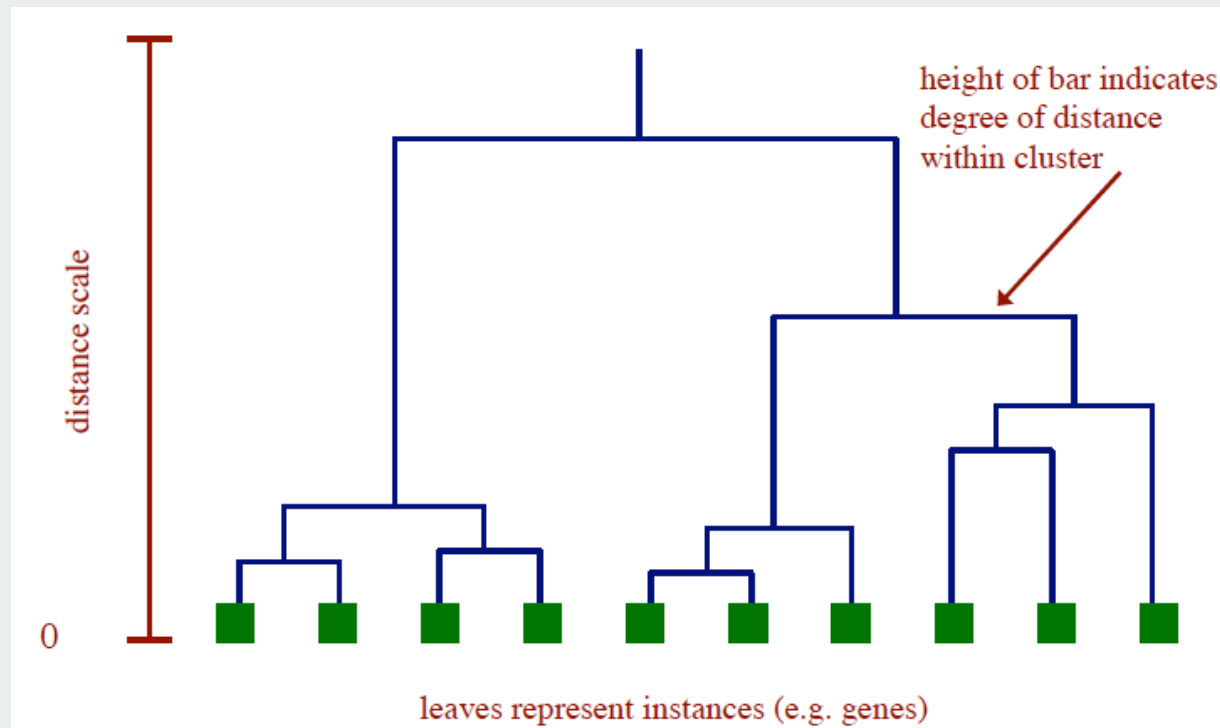
Property. Kruskal's algorithm finds a k -clustering of maximum spacing.

Clustering application: dendrograms

Dendrogram.

Scientific visualization of hypothetical sequence of evolutionary events.

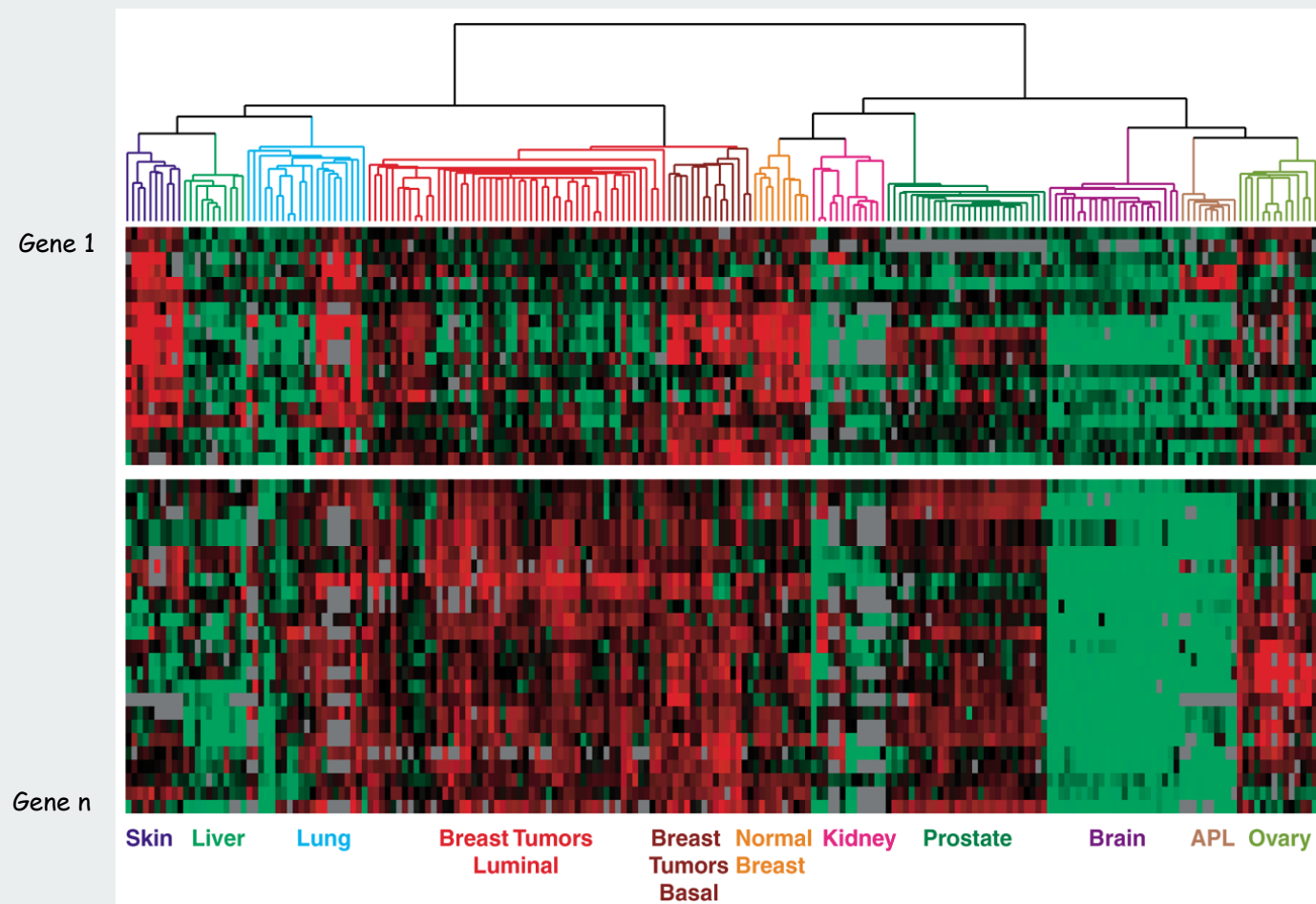
- Leaves = genes.
- Internal nodes = hypothetical ancestors.



Reference: <http://www.biostat.wisc.edu/bmi576/fall-2003/lecture13.pdf>

Dendrogram of cancers in human

Tumors in similar tissues cluster together.



Reference: Botstein & Brown group

■ gene expressed
■ gene not expressed