5.3 Minimum Spanning Trees

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

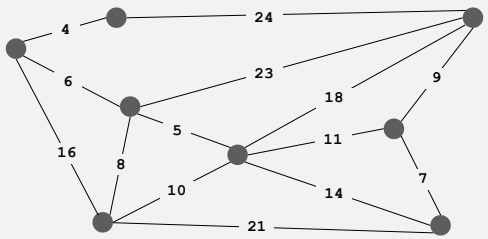
Reference: Algorithms in Java, 3rd edition, Part 5, Chapter 20

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.



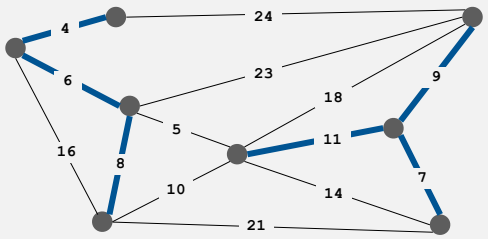
graph G

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.



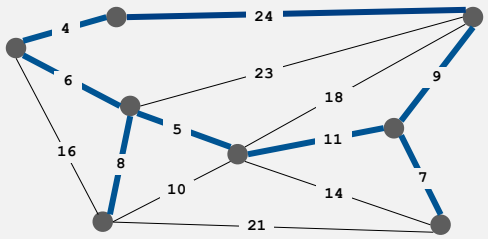
not connected

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.



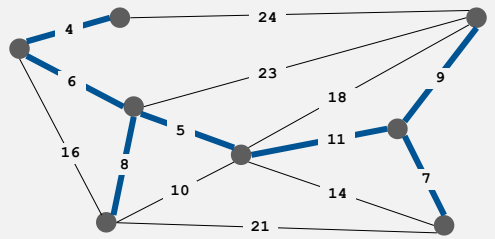
not acyclic

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.



spanning tree T: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Brute force. Try all spanning trees.

Applications

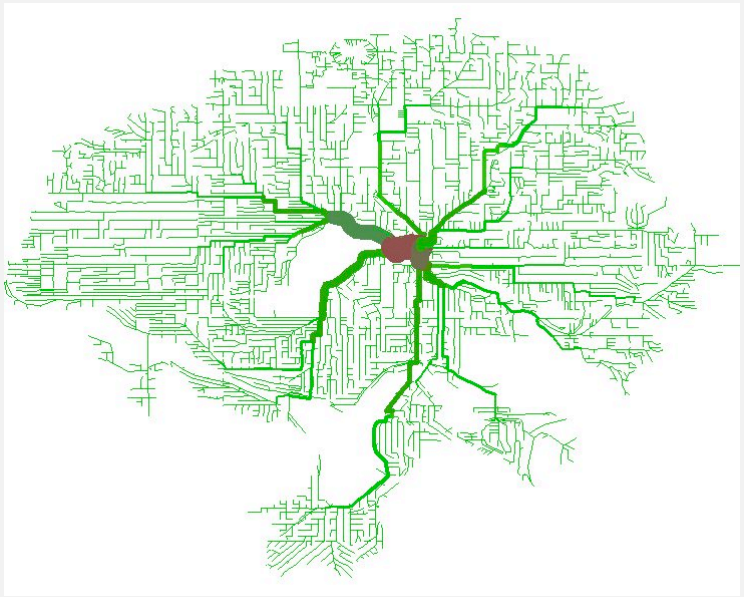
MST is fundamental problem with diverse applications.

- 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).

http://www.ics.uci.edu/~eppstein/gina/mst.html

Network design

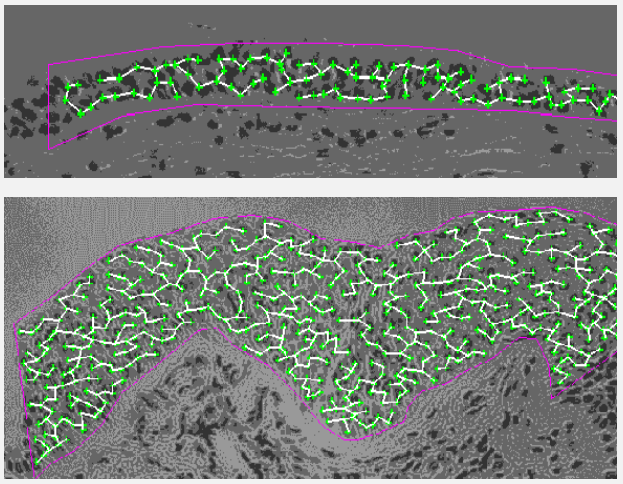
MST of bicycle routes in North Seattle



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

Medical image processing

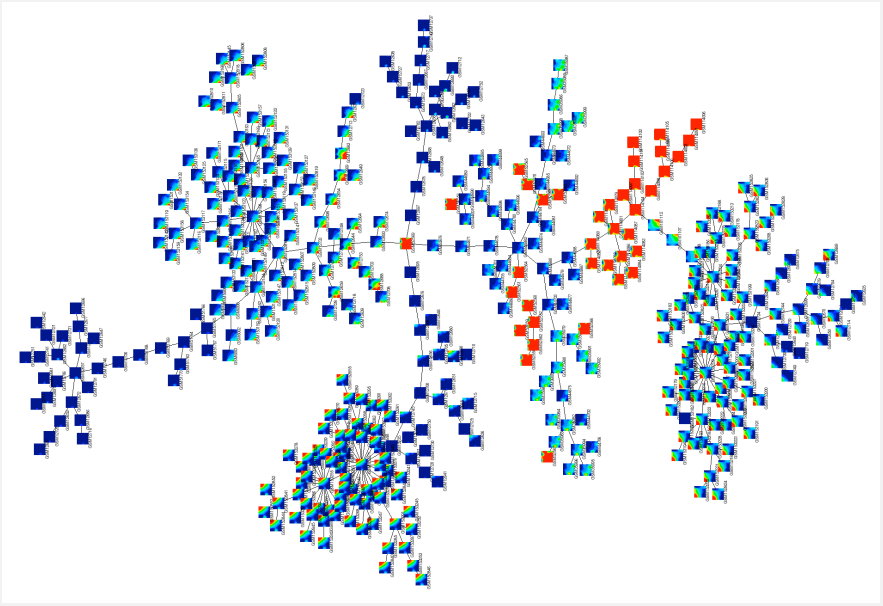
MST describes arrangement of nuclei in the epithelium for cancer research



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

Genetic research

MST of tissue relationships measured by gene expression correlation coefficient



http://riodb.ibase.aist.go.jp/CELLPEDIA

Two greedy algorithms

Kruskal's algorithm. Consider edges in ascending order of weight.
Add to T 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.

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

Proposition. Both greedy algorithms compute MST.

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

Edge API

Edge abstraction needed for weighted edges.

public class Edge implements Comparable<Edge>	
Edge(int v, int w, double weight)	*create a weighted edge v-w*
int either()	*either endpoint*
int other(int v)	*the endpoint that's not v*
double weight()	*the weight*
Comparator<Edge> ByWeight()	*compare by edge weight*

Weighted graph API

public class WeightedGraph		
	WeightedGraph(int V)	*create an empty graph with V vertices*
	WeightedGraph(In in)	*create a graph from input stream*
void	addEdge(Edge e)	*add edge e*
void	removeEdge(Edge e)	*delete edge e*
Iterable<Edge>	adj(int v)	*return an iterator over edges incident to v*
int	V()	*return number of vertices*

Conventions.

- Allow self-loops.
- Allow parallel edges (provided they have different weights).

Weighted graph API

public class WeightedGraph		
	WeightedGraph(int V)	*create an empty graph with V vertices*
	WeightedGraph(In in)	*create a graph from input stream*
void	addEdge(Edge e)	*add edge e*
void	removeEdge(Edge e)	*delete edge e*
Iterable<Edge>	adj(int v)	*return an iterator over edges incident to v*
int	V()	*return number of vertices*

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

iterate through all edges (once in each direction)

Weighted graph: adjacency-set implementation

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

   public WeightedGraph(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.either(), w = e.other(v);
      adj[v].add(e);
      adj[w].add(e);
   }

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

same as Graph, but adjacency sets of Edges instead of integers

constructor

add edge to both adjacency sets

Weighted edge: Java implementation

```
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 = Math.min(v, w);
      this.w = Math.max(v, 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 compare methods.
}
```

constructor

either endpoint

other endpoint

weight of edge

Weighted edge: Java implementation (cont)

```
public 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;
   }
}
```

order edges by weight (for sorting in Kruskal)

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

lexicographic order, breaking ties by weight (for use in a symbol table)

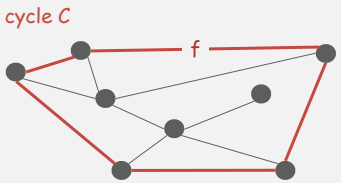
- weighted graph API
- **cycles and cuts**
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Cycle and cut properties

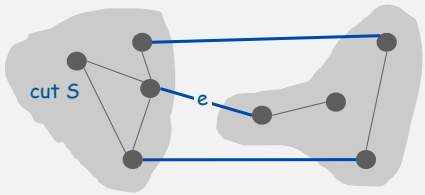
Simplifying assumption. All edge weights w_e are distinct.

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

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



f is not in the MST T*



e is in the MST T*

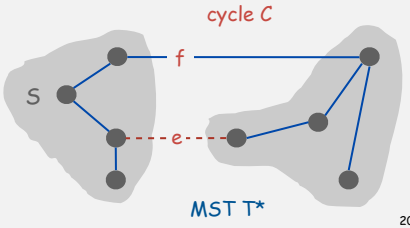
Cycle property: correctness proof

Simplifying assumption. All edge weights w_e are distinct.

Cycle property. Let C be any cycle, and let f be the max weight 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 $w_e < w_f$, weight(T) < weight(T*).
- Contradicts minimality of T*. ▪



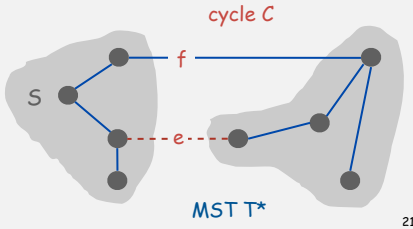
Cut property: correctness proof

Simplifying assumption. All edge weights w_e are distinct.

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

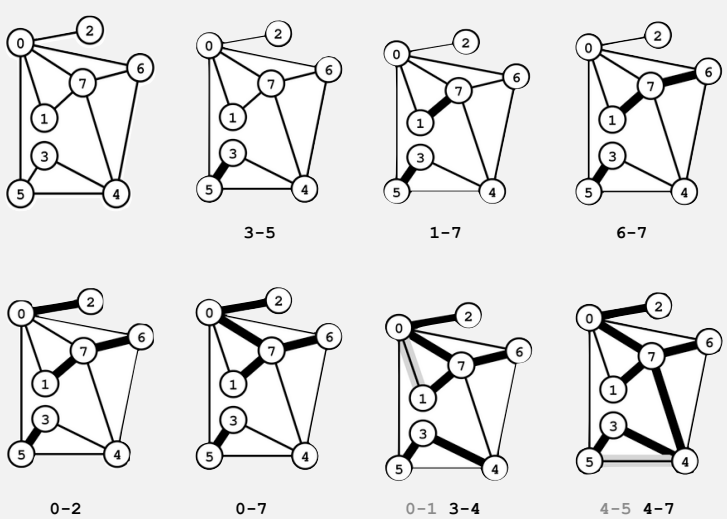
Pf. [by contradiction]

- Suppose e does not belong to T*. Let's see what happens.
- Adding e to T* creates a 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 $w_e < w_f$, weight(T) < weight(T*).
- Contradicts minimality of T*. ▪



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.



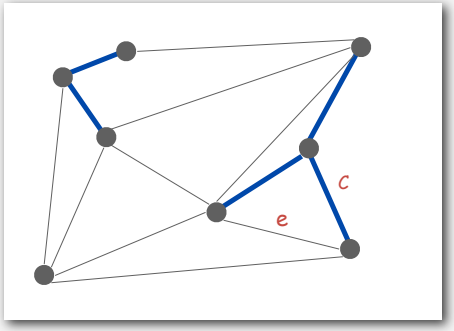

```
3-5  0.18
1-7  0.21
6-7  0.25
0-2  0.29
0-7  0.31
0-1  0.32
3-4  0.34
4-5  0.40
4-7  0.46
0-6  0.51
4-6  0.51
0-5  0.60
```


Kruskal's algorithm: correctness proof

Proposition. Kruskal's algorithm computes the MST.

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

- Edge e is the max weight edge in C. ← why max weight?
- Edge e is not in the MST (cycle property).

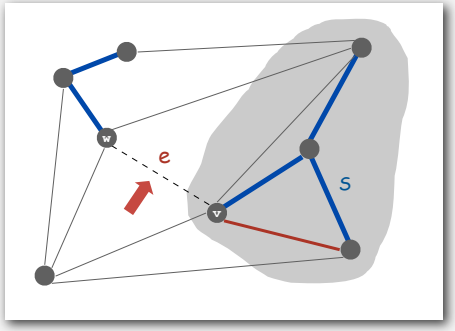


Kruskal's algorithm: correctness proof

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.
- Vertex w is not in S.
- Edge e is the min weight edge with exactly one endpoint in S. (why min weight?)
- Edge e is in the MST (cut property). ▪

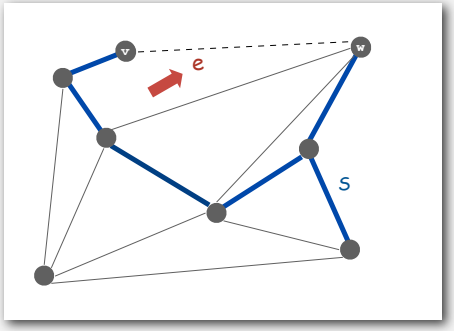


Kruskal implementation challenge

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

How difficult?

- O(E + V) time.
- O(V) time. ← run DFS from v, check if w is reachable (T has at most V-1 edges)
- O(log V) time.
- O(log* V) time. ← use the union-find data structure !
- Constant time.

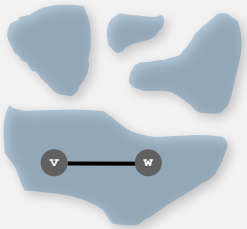


Kruskal's algorithm implementation

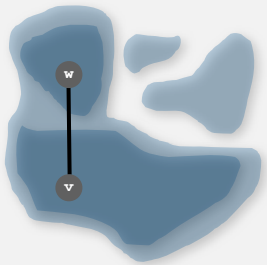
Problem. Check if adding an edge v-w to T creates a cycle.

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

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

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

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

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

get all edges in graph

sort edges by weight

greedily add edges to MST

Kruskal's algorithm running time

Proposition. Kruskal's algorithm computes MST in O(E log E) time.

Pf.

operation	frequency	time per op
sort	1	E log E
union	V	log* V †
find	E	log* V †

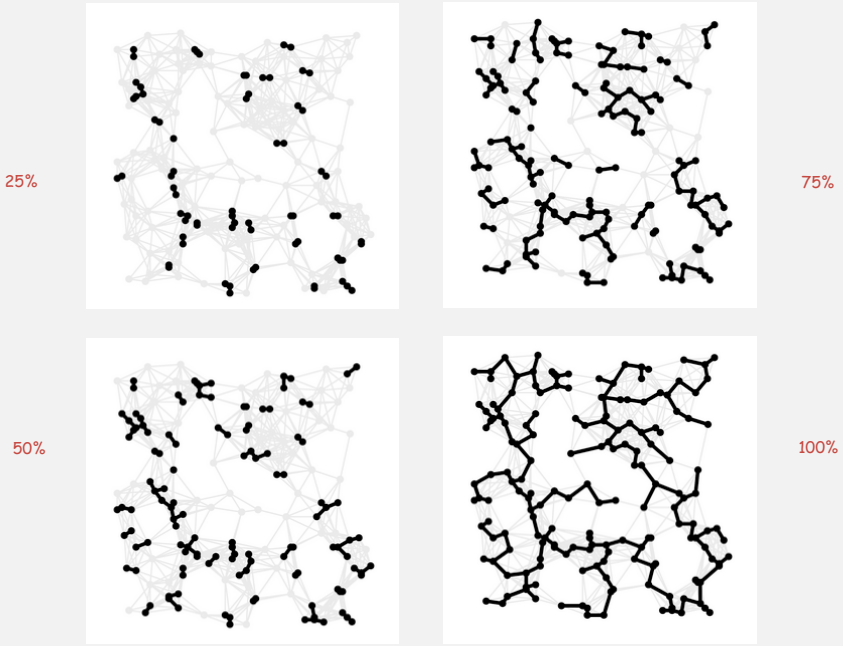
† amortized bound using weighted quick union with path compression

Improvements.

- Stop as soon as there are V-1 edges.
- If edges are already sorted, time is proportional to E log* V.

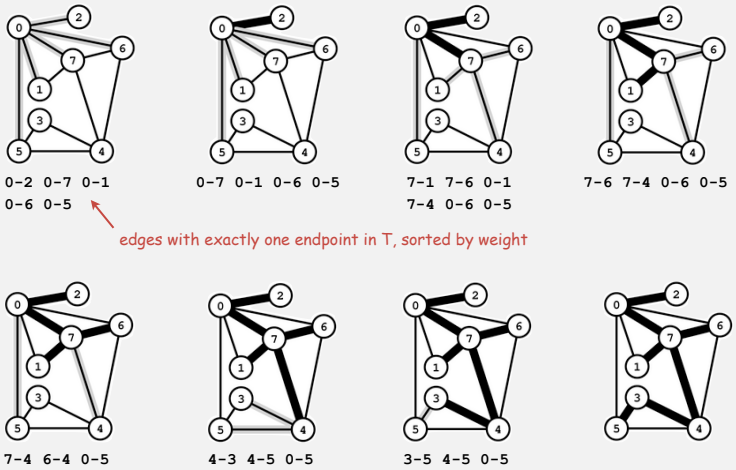
recall: log* V ≤ 5 in this universe

Kruskal's algorithm example



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.



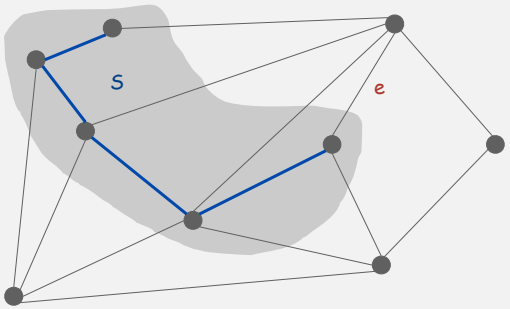

```
0-1 0.32
0-2 0.29
0-5 0.60
0-6 0.51
0-7 0.31
1-7 0.21
3-4 0.34
3-5 0.18
4-5 0.40
4-6 0.51
4-7 0.46
6-7 0.25
```

Prim's algorithm correctness proof

Proposition. Prim's algorithm computes the MST.

Pf.

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

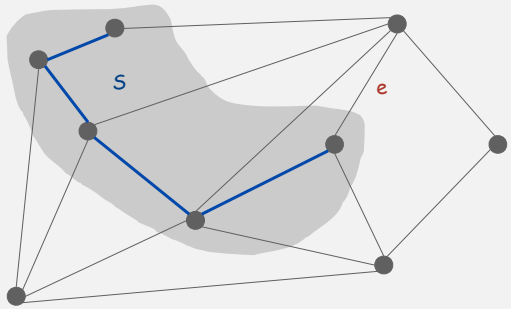


Prim implementation challenge

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

How difficult?

- O(E) time. ← try all edges
- O(V) time.
- O(log E) time. ← use a priority queue !
- O(log* E) time.
- Constant time.

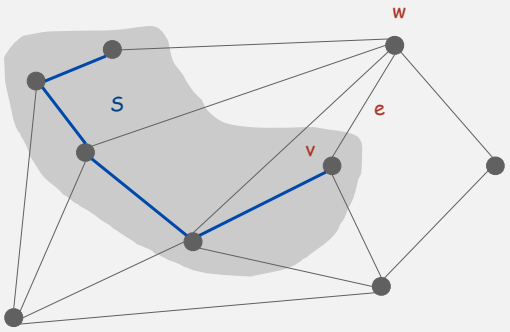


Prim's algorithm implementation (lazy)

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.

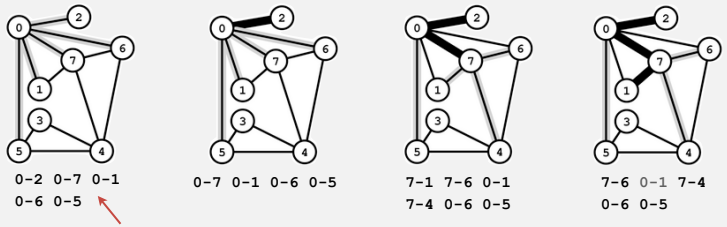
- Delete min to determine next edge e = v-w to add to T.
- Disregard if both v and w are in S.
- Let w be vertex not in S:
  - add to PQ any edge incident to w (assuming other endpoint not in S)
  - add w to S



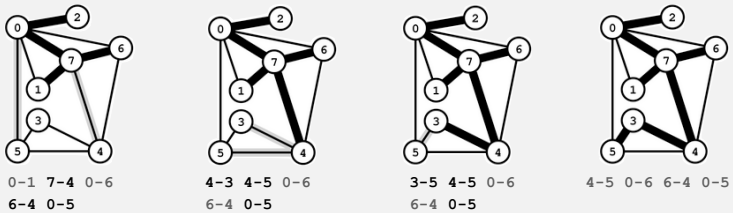
Prim's algorithm example: lazy implementation

Use PQ: key = edge.

(lazy version leaves some obsolete entries on the PQ)



black = PQ edge with exactly one endpoint in S, sorted by weight
gray = PQ edge with both endpoints in S (obsolete)




```
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
```

Lazy implementation of Prim's algorithm

```
public class LazyPrim
{
   private boolean[] scanned;     // vertices in MST
   private Queue<Edge> mst;       // edges in the MST
   private MinPQ<Edge> pq         // the priority queue of edges

    public LazyPrim(WeightedGraph G)
    {
        scanned = 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.
}
```

comparator by edge weight (instead of by lexicographic order)

Lazy implementation of Prim's algorithm

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

for each edge v-w, add to PQ if w not already in S

```
private void prim(WeightedGraph G, int s)
{
   scan(G, s);
   while (!pq.isEmpty())
   {
      Edge e = pq.delMin();
      int v = e.either(), w = e.other(v);
      if (scanned[v] && scanned[w]) continue;
      mst.enqueue(e);
      if (!scanned[v]) scan(G, v);
      if (!scanned[w]) scan(G, w);
   }
}
```

repeatedly delete the min weight edge v-w from PQ

ignore if both endpoints in S

add e to MST and scan v and w

Prim's algorithm running time

Proposition. Prim's algorithm computes MST in $O(E \log E)$ time.

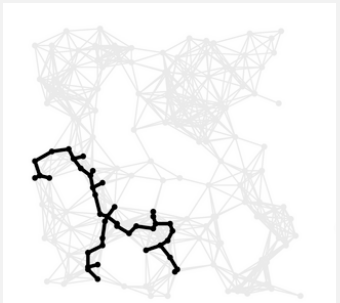
operation	frequency	time per op
delete min	E	E log E
insert	E	E log E

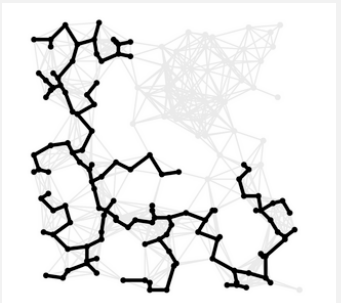
Improvements.

- Stop when MST has V-1 edges.
- 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 $O(E + V \log V)$.

Prim's algorithm example

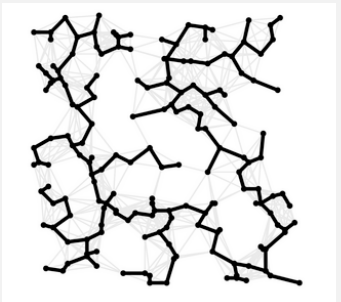
25%





75%

50%



100%

Removing the distinct edge weight assumption

Simplifying assumption. All edge weights are distinct.

Approach 1. Introduce tie-breaking rule for `compare()` in `ByWeight`.

```
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;
}
```

`return e.compareTo(f);`

Approach 2. Prim and Kruskal still find MST if equal weights!
(only our proof of correctness fails)

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

Does a linear-time MST algorithm exist?

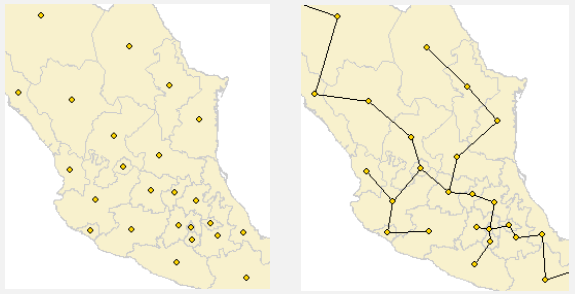
deterministic compare-based MST algorithms

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 α(V) log α(V)	Chazelle
2000	E α(V)	Chazelle
2002	optimal	Pettie-Ramachandran
20xx	E	???

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.

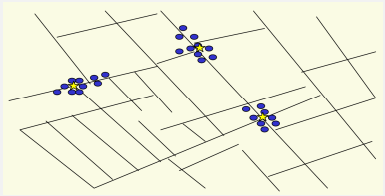


Brute force. Compute ~ $N^2/2$ distances and run Prim's algorithm.
Ingenuity. Exploit geometry and do it in ~ c N lg N.

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.



outbreak of cholera deaths in London in 1850s (Nina Mishra)

Applications.

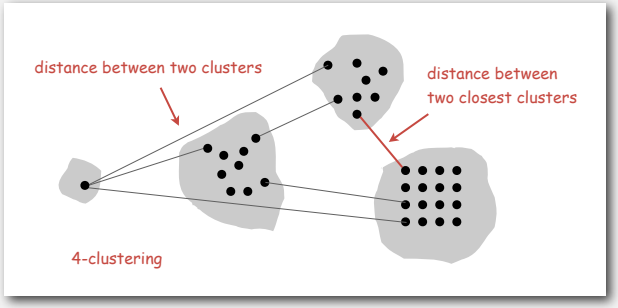
- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases.
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

Single linkage

k-clustering. Divide a set of objects classify into k coherent groups.
Distance function. Numeric value specifying "closeness" of two objects.

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

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



Single-linkage clustering algorithm

"Well-known" algorithm for single-linkage 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.

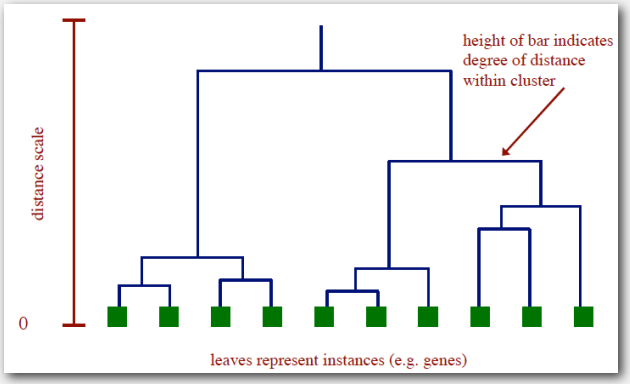
Observation. This procedure is precisely Kruskal's algorithm (stopping when there are k connected components).

Alternate solution. Run Prim's algorithm and delete k-1 max weight edges.

Clustering application: dendrograms

Dendrogram.
Scientific visualization of hypothetical sequence of evolutionary events.

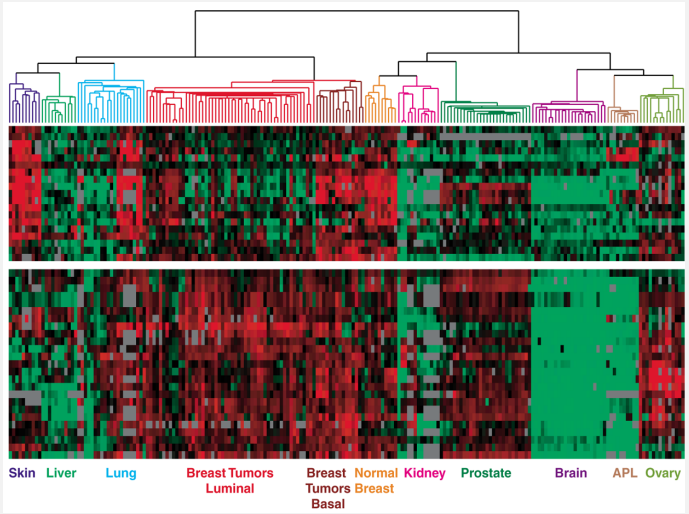
- Leaves = genes.
- Internal nodes = hypothetical ancestors.



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