4.3 Minimum Spanning Trees

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

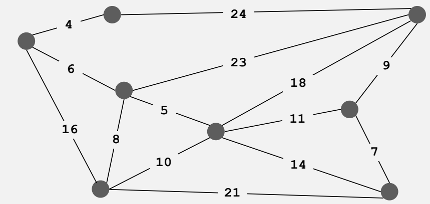
Reference: Algorithms in Java, 3rd edition, Part 5, Chapter 20

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.



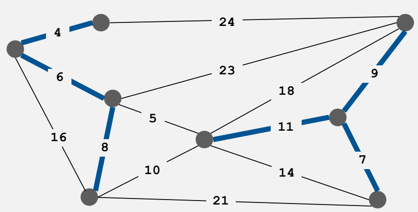
graph G

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.



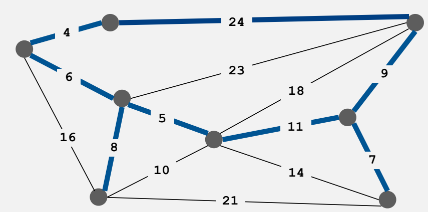
not connected

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.



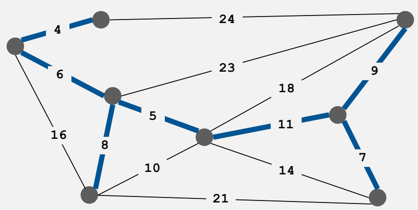
not acyclic

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?

NO! Reason 1: How?

Reason 2: There are V^{V-2} of them.

Applications

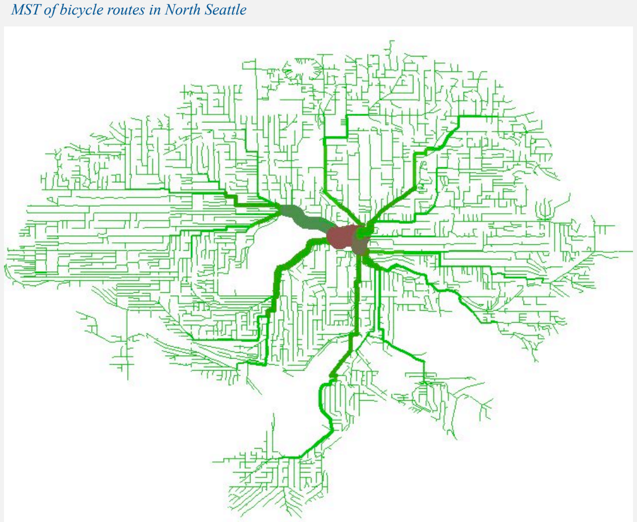
MST is fundamental problem with diverse applications.

- 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).

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

Network design

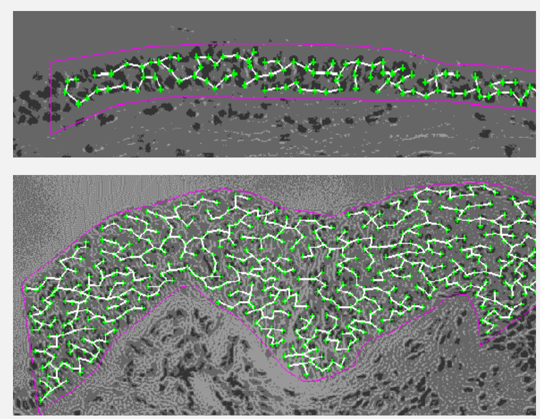
MST of bicycle routes in North Seattle



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

Medical image processing

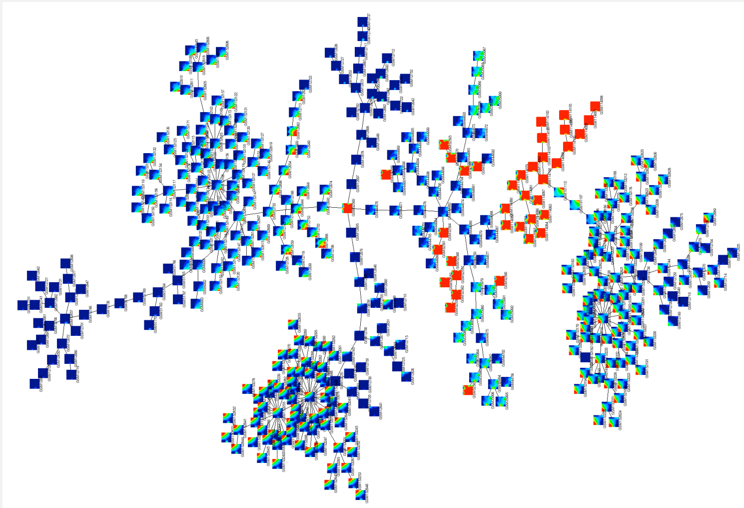
MST describes arrangement of nuclei in the epithelium for cancer research



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

Genetic research

MST of tissue relationships measured by gene expression correlation coefficient



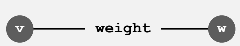
http://riodb.ibase.aist.go.jp/CELLPEDIA

- **weighted graph API**
- greedy algorithm
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Edge API

Edge abstraction needed for weighted edges.

`public class 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*



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

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 edge e*
`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*

Conventions.
- Allow self-loops.
- Allow 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 edge e*
`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*

```
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-lists implementation

```
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];  }
}
```

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
{
   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;  }

}
```

constructor

either endpoint

other endpoint

weight of edge

Weighted edge comparator

Clients need to compare edge weights.

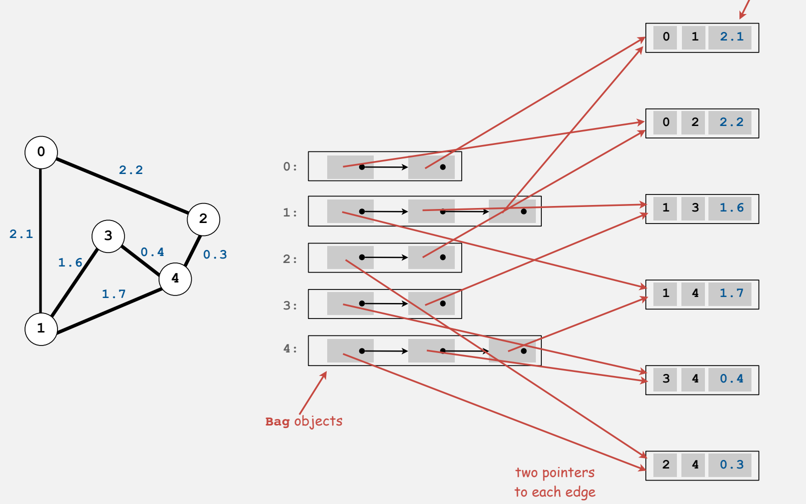
```
private static class ByWeight implements Comparator<Edge>
{
   public int compare(Edge e, Edge f)
   {
      if (e.weight() < f.weight()) return -1;
      if (e.weight() > f.weight()) return +1;
      return 0;
   }
}
```

order edges by weight

Note: different clients may use different `Comparator` implementations

Edge-weighted graph: adjacency-list representation

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

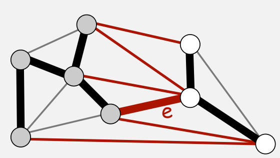


- weighted graph API
- **greedy algorithm**
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Cut property

Simplifying assumption. Edge weights are different.

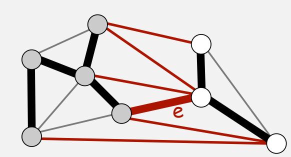
Cut property. Given any cut, the minimum-weight crossing edge is in the MST.



Cut property: correctness proof

Simplifying assumption. Edge weights are different.

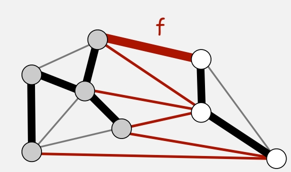
Cut property. Given any cut, the minimum-weight crossing edge is in the MST.



Pf.

Let e be the min-weight crossing edge

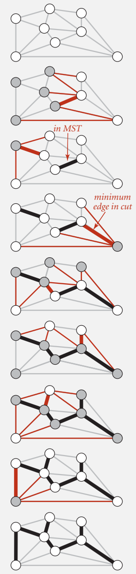
- Suppose e is not in the MST.
- Adding to the MST e creates a cycle C.
- Some other edge f in C must be a crossing edge.
- Removing f and adding e is also a spanning tree.
- Since $w_e < w_f$, that spanning tree is lower weight.
- Contradiction. ▪



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

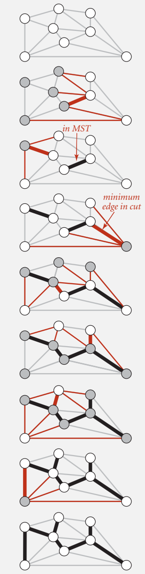


Greedy MST algorithm

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.

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.

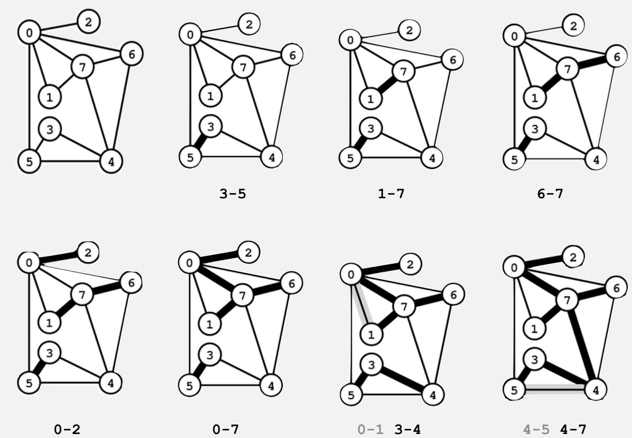
" Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit. " — Gordon Gecko

Proposition. Both algorithms compute MST.
Proof. Vertices touched by black edges define a cut.

- weighted graph API
- greedy algorithm
- **Kruskal's algorithm**
- Prim's algorithm
- advanced topics

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.



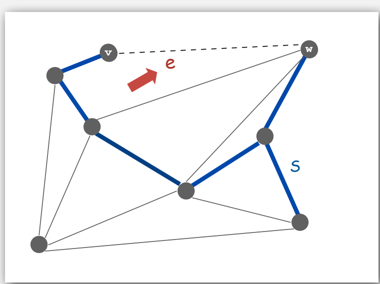
3-5	0.18
1-7	0.21
6-7	0.25
0-2	0.29
0-7	0.31
0-1	0.32
3-4	0.34
4-5	0.40
4-7	0.46
0-6	0.51
4-6	0.51
0-5	0.60

Kruskal 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 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;  }
}
```

build priority queue

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
build pq	1	E
del min	E	log E
union	V	log* V †
find	E	log* V †

† amortized bound using weighted quick union with path compression

Improvements.

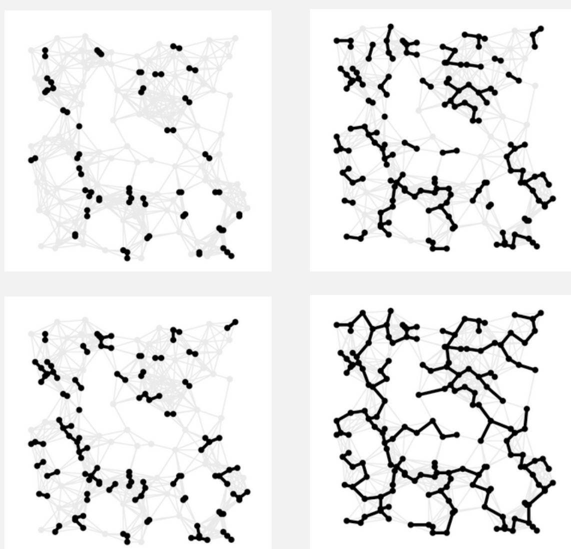
recall: log* V ≤ 5 in this universe

- If edges are already sorted, worst case time is ~E log* V.
- Stop as soon as V-1 edges on MST: only a fraction of edges leave pq.

Kruskal's algorithm

25%

75%



50%

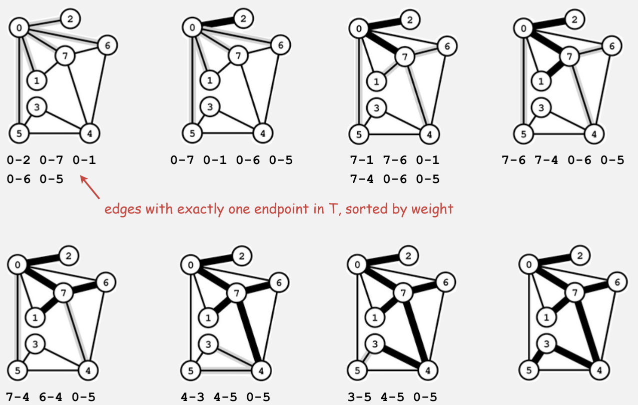
100%

- weighted graph API
- greedy algorithm
- 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 to T the edge of min weight with exactly one endpoint in T.



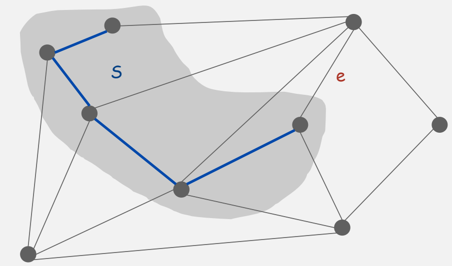
0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25

Prim 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.

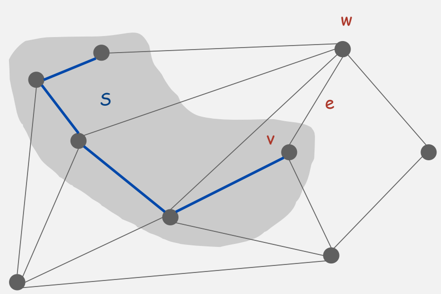


Prim's algorithm implementation (lazy)

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.

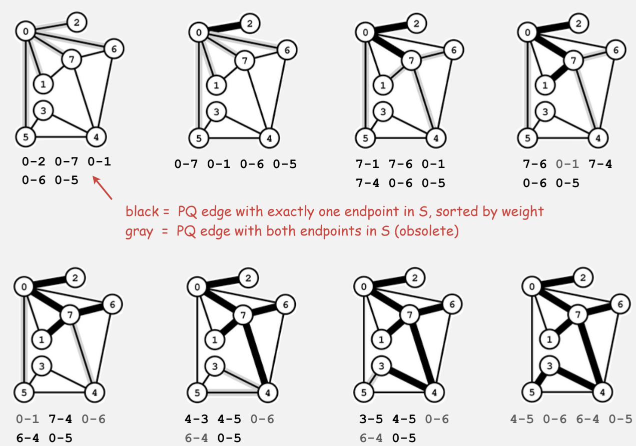
- Delete min to determine next edge e = v-w to add to T.
- Disregard if both v and w are in S.
- Let w be vertex not in S:
  - add to PQ any edge incident to w (assuming other endpoint not in S)
  - add w to S



Prim's algorithm example: lazy implementation

Use PQ: key = edge.

(lazy version leaves some obsolete entries on the PQ)



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 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.
}
```

comparator by edge weight

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);
}
```

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

```
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);
   }
}
```

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.

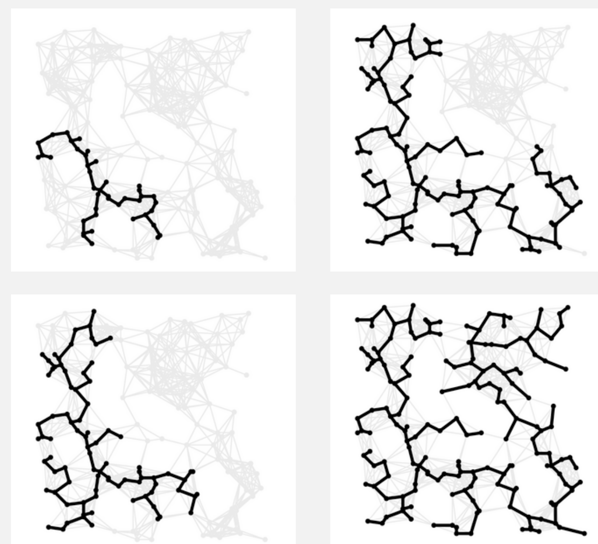
Pf.

operation	frequency	time per op
delete min	V	lg E
insert	E	2 lg E

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).

Prim's algorithm



- weighted graph API
- greedy algorithm
- Kruskal's algorithm
- Prim's algorithm
- **advanced topics**

Removing the distinct edge weight assumption

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)

Does a linear-time MST algorithm exist?

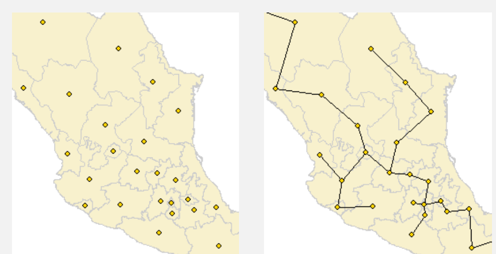
deterministic compare-based MST algorithms

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.

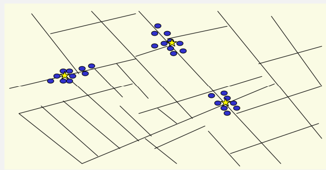


Brute force. Compute ~ $N^2/2$ distances and run Prim's algorithm.
Ingenuity. Exploit geometry and do it in ~ c N lg N.

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)

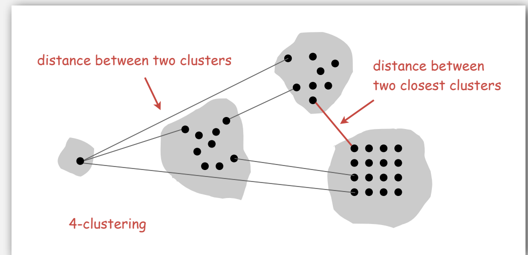
Applications.
- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases.
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

Single-link clustering

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

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

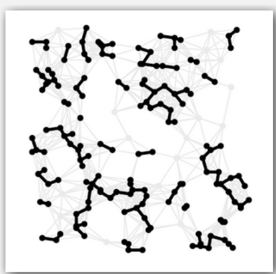
Single-link clustering. Given an integer k, find a k-clustering that maximizes the distance between two closest clusters.



Single-link clustering algorithm

"Well-known" algorithm 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.

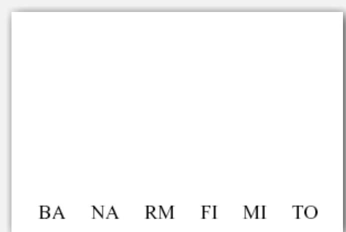
Observation. This is Kruskal's algorithm (stop when k connected components).



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

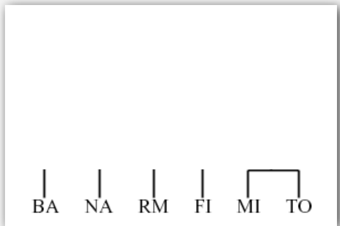
Dendrogram

Dendrogram. Tree diagram that illustrates arrangement of clusters.



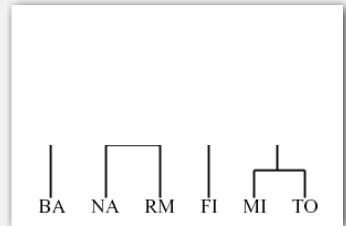
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html

Dendrogram. Tree diagram that illustrates arrangement of clusters.



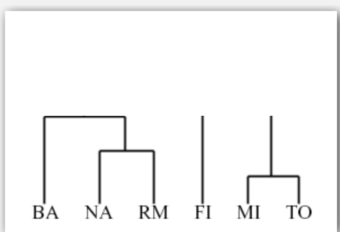
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html

Dendrogram. Tree diagram that illustrates arrangement of clusters.



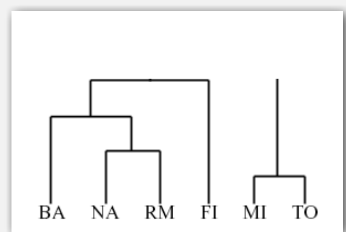
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html

Dendrogram. Tree diagram that illustrates arrangement of clusters.



http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html

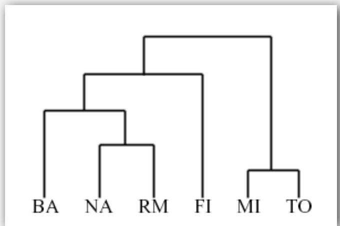
Dendrogram. Tree diagram that illustrates arrangement of clusters.



http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html

Dendrogram

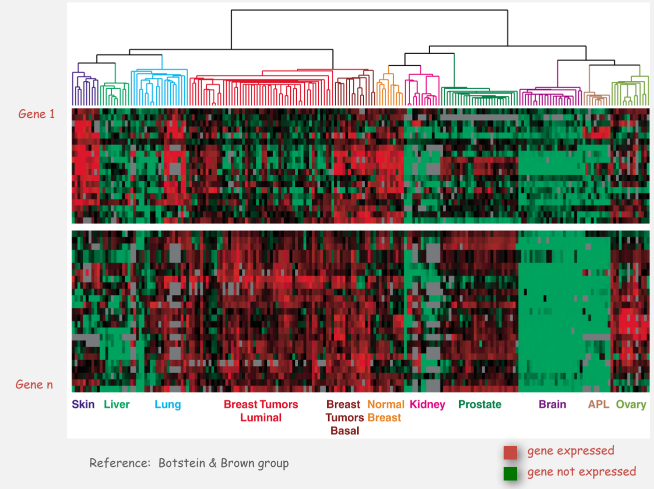
Dendrogram. Tree diagram that illustrates arrangement of clusters.



http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html

Dendrogram of cancers in human

Tumors in similar tissues cluster together.



Reference: Botstein & Brown group