Minimum Spanning Tree

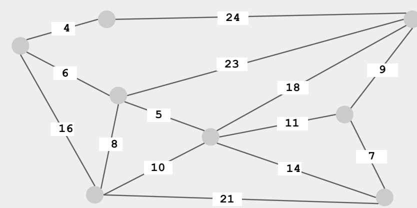
- introduction
- Weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced algorithms
- clustering

References: Algorithms in Java (Part 5), Chapter 20
Intro to Algs and Data Structures, Section 5.4

introduction
weighted graph API
cycles and cuts
Kruskal's algorithm
Prim's algorithm
advanced algorithms
clustering

Minimum Spanning Tree

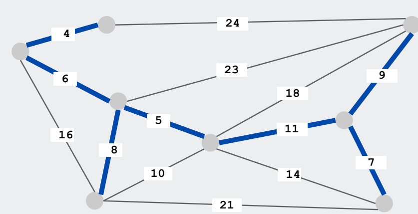
MST. Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.



G

Minimum Spanning Tree

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



`cost(T) = 50`

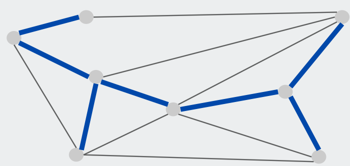
Brute force: Try all possible spanning trees
- problem 1: not so easy to implement
- problem 2: far too many of them

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 in combinatorial optimization.



Otakar Boruvka

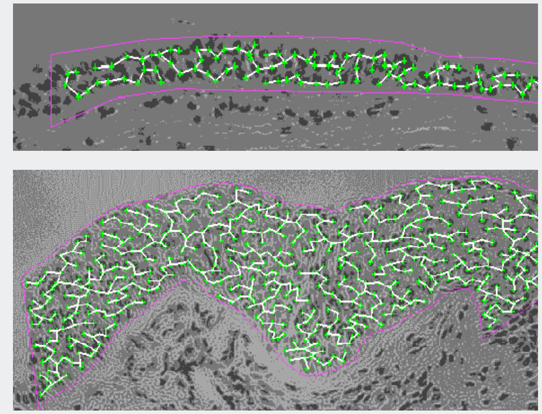
Applications

MST is fundamental problem with diverse applications.

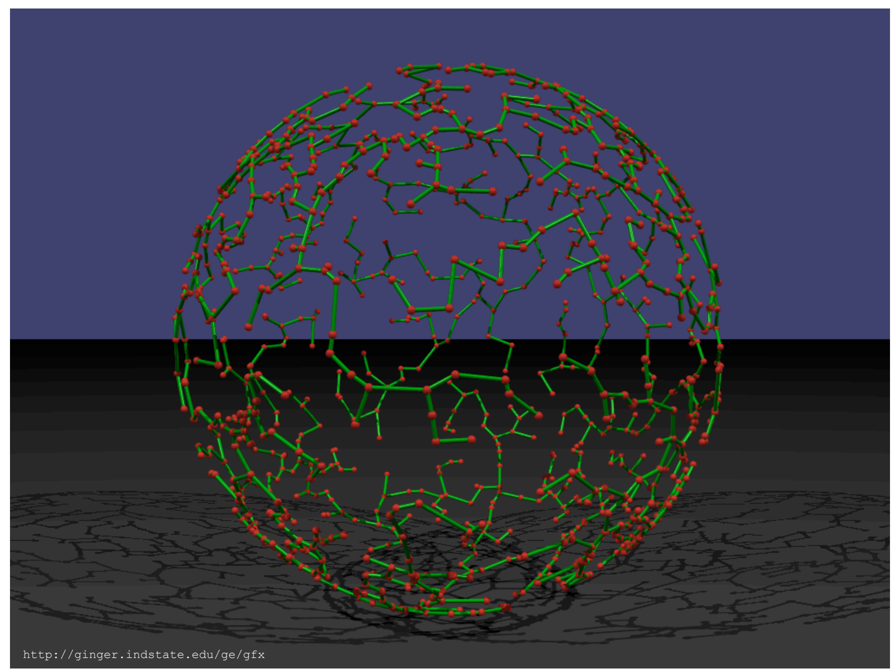
- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree
- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy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

Theorem. 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 Interface

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

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

iterate through all edges (once in each direction)

introduction

weighted graph API

cycles and cuts

Kruskal's algorithm

Prim's algorithm

advanced algorithms

clustering

Edge data type

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

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

   public int either()
   {  return v; }

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

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

Weighted graph: Java implementation

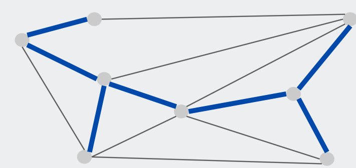
Identical to `Graph.java` but use `Edge` adjacency lists instead of `int`.

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

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

   public void insert(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];  }

}
```


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.

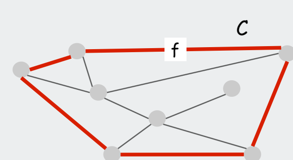
introduction
weighted graph API
cycles and cuts
Kruskal's algorithm
Prim's algorithm
advanced algorithms
clustering

Greedy Algorithms

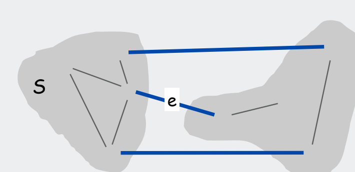
Simplifying assumption. All edge costs c_e are distinct.

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

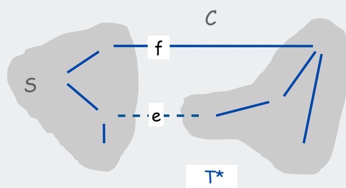
Cycle Property

Simplifying assumption. All edge costs c_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 $c_e < c_f$, cost(T) < cost(T*).
- Contradicts minimality of T*. ▪



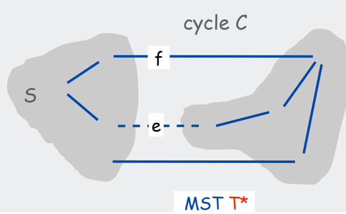
Cut Property

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.

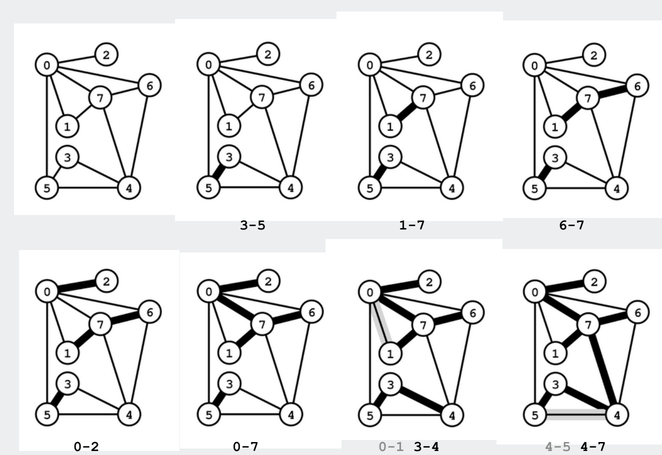
Pf. [by contradiction]

- 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 $c_e < c_f$, cost(T) < cost(T*).
- Contradicts minimality of T*. ▪

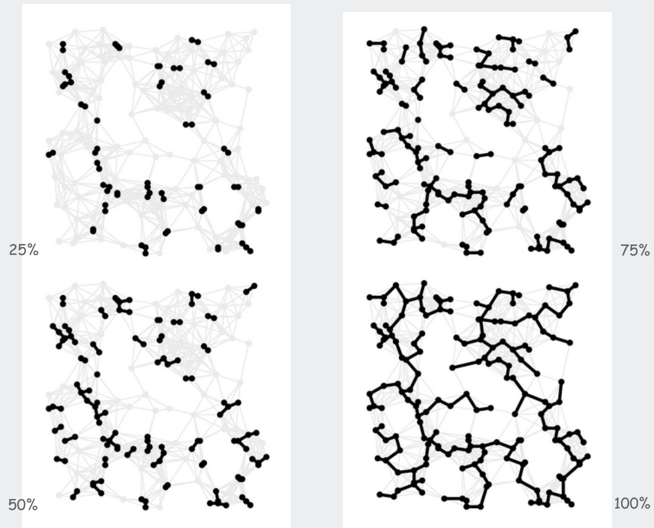


Kruskal's Algorithm: Example

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.



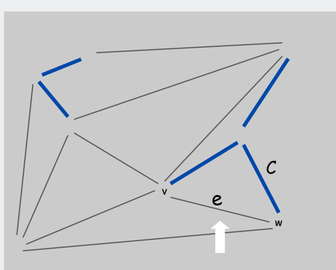
Kruskal's algorithm example



Kruskal's algorithm correctness proof

Theorem. 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)

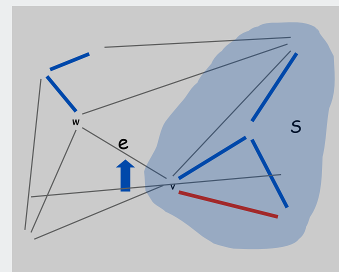


Kruskal's algorithm correctness proof

Theorem. 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)

QED



Kruskal's algorithm implementation

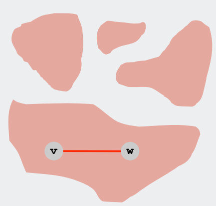
Q. How to check if adding an edge to T would create a cycle?

A1. Naïve solution: use DFS.
- O(V) time per cycle check.
- O(E V) time overall.

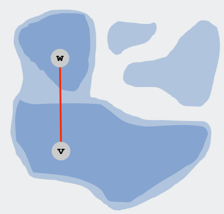
Kruskal's algorithm implementation

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 Sequence<Edge> mst
                    = new Sequence<Edge>();

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

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

   }

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

sort edges

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 †
find	E	log* V †

† amortized bound using weighted quick union with path compression

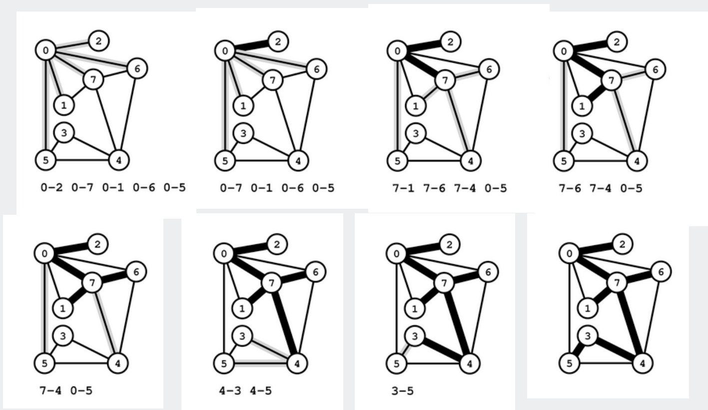
recall: log* V ≤ 5 in this universe

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

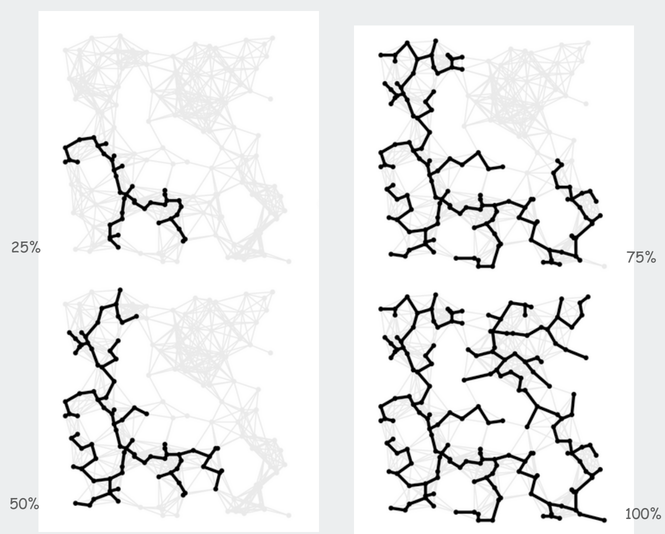
Remark 2. With PQ or quicksort partitioning, time depends on number of edges shorter than longest edge in the MST (may be small in practice).

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.



Prim's Algorithm example

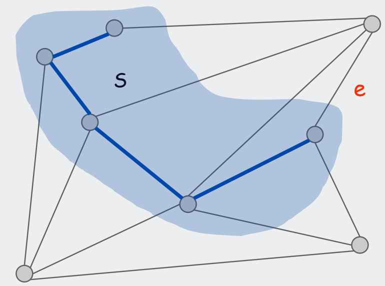


Prim's algorithm correctness proof

Theorem. 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)

QED.



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 edges with (at least) one endpoint in S
- Delete min to determine next edge e to add to T.
- Disregard e if both endpoints are in S.
- Upon adding e to T, add to PQ the edges incident to the endpoint not already in S.

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 nodes on PQ
book: first check whether the other endpoint is was put in S during the time it was on the PQ

Prim's Algorithm: Java Implementation

```
public class LazyPrim
{
   private Sequence<Edge> mst = new Sequence<Edge>();
   public LazyPrim(WeightedGraph G)
   {
      boolean[] marked = new boolean[G.V()];
      MinPQ<Edge> pq = new MinPQ<Edge>();
      int s = 0;
      marked[s] = true;
      for (Edge e : G.adj(s)) pq.insert(e);
      while (!pq.isEmpty())
      {
        Edge e = pq.delMin();
        int v = e.either(), w = e.other(v);
        if (!marked[v])
         {
            mst.add(e); marked(v) = true;
            for (Edge f : G.adj(v)) pq.insert(f);
         }
        else if (!marked[w])
         {
            mst.add(e); marked(w) = true;
            for (Edge f : G.adj(w)) pq.insert(f);
         }
      }
   }
}
```

marks vertices in s

PQ

add to PQ all edges incident to s

add to PQ all edges incident to e.v

Removing the distinct edge costs assumption

Simplifying assumption. All edge costs c_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 `compareTo()`.

Approach 1:

```
public int compareTo(Edge f)
{
   Edge e = this;
   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.

Advanced 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	???

deterministic comparison based MST algorithms

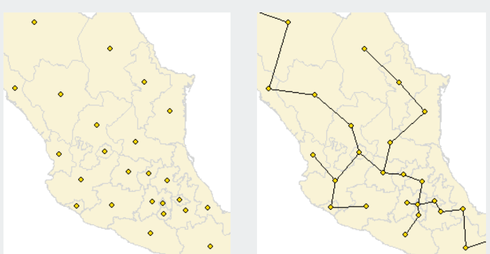
Year	Problem	Time	Discovered By
1976	Planar MST	E	Cheriton-Tarjan
1992	MST Verification	E	Dixon-Rauch-Tarjan
1995	Randomized MST	E	Karger-Klein-Tarjan

related problems

Euclidean MST

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

- Distances between point pairs are Euclidean distances.



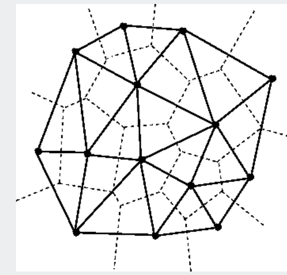
Brute force. Compute $N^2 / 2$ distances and run Prim's algorithm.
Ingenuity. Exploit geometry and do it in O(N log N).

Euclidean MST

Key geometric fact.
Edges of the Euclidean MST are edges of the Delaunay triangulation.

Euclidean MST algorithm.

- Compute Delaunay triangulation.
- Run Kruskal's MST algorithm on Delaunay edges.



Running time. O(N log N).

- O(N) Delaunay edges since it is planar.
- O(N log N) for Delaunay.
- O(N log N) for Kruskal.

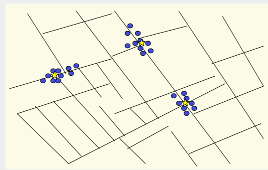
In practice. Use any set of edges of size O(N) that contains the Delaunay with high probability

Lower bound. Any comparison-based Euclidean MST algorithm requires Ω(N log N) comparisons.

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.



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

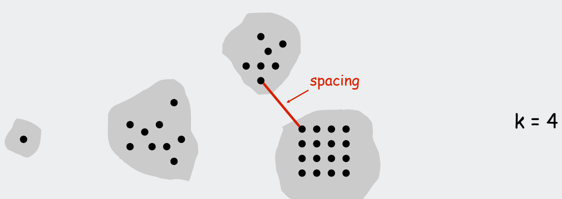
Applications.
- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

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.



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.

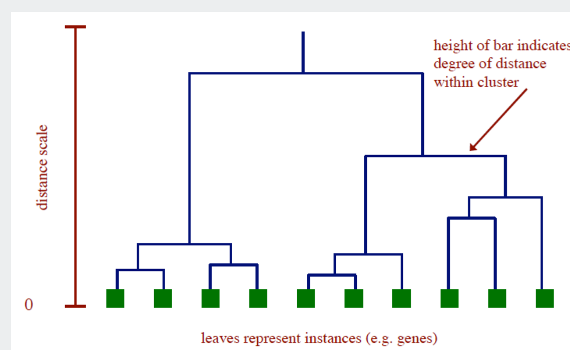
Observation. This procedure is precisely Kruskal's algorithm (stop when there are k connected components).

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

Dendrogram

Dendrogram.
Scientific visualization of hypothetical sequence of evolutionary events.

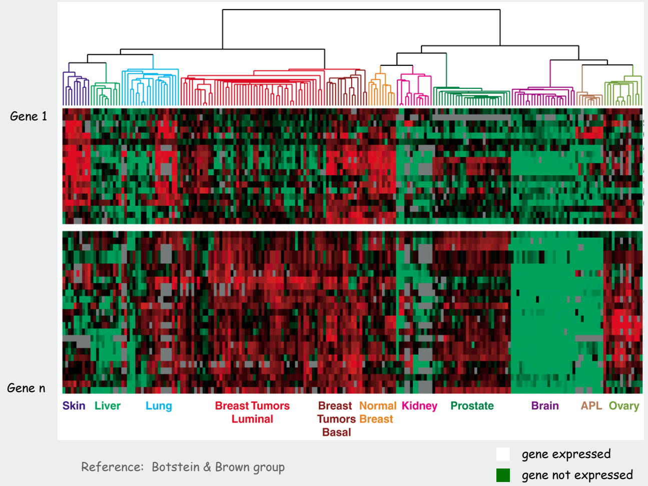
- Leaves = genes.
- Internal nodes = hypothetical ancestors.



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