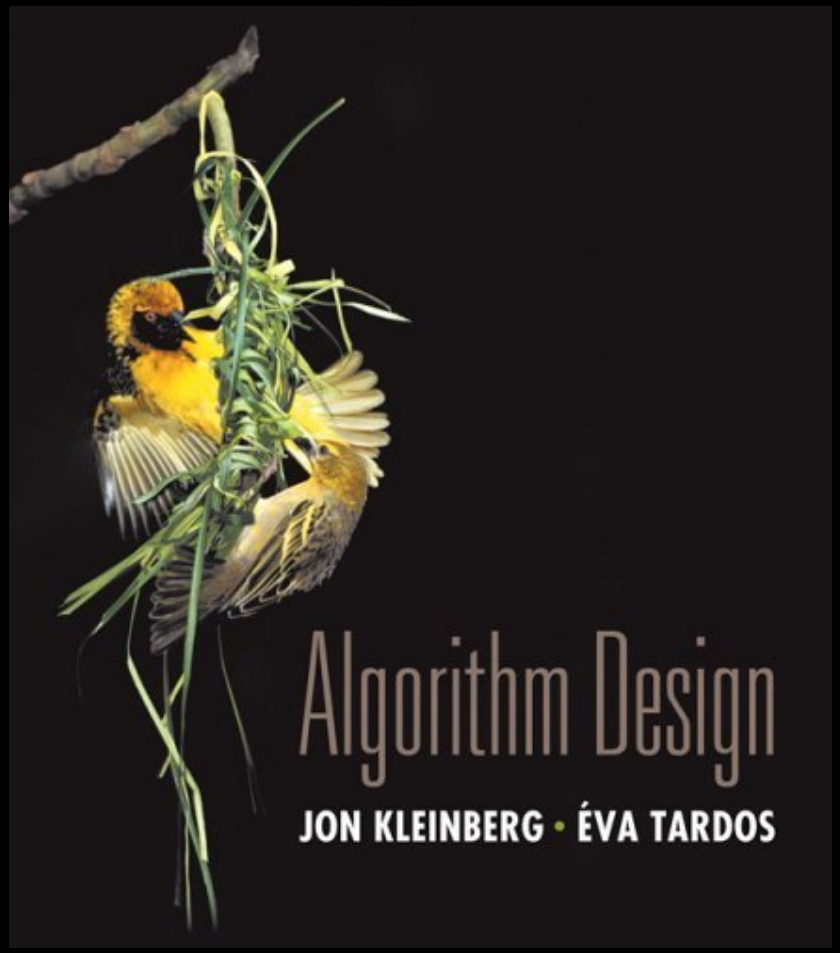


Chapter 4

Greedy Algorithms

4.5 Minimum Spanning Tree

Il problema del calcolo di un Minimum Spanning Tree (MST)

Input:

- un grafo non orientato e pesato $G=(V,E,w)$

Soluzione ammissibile:

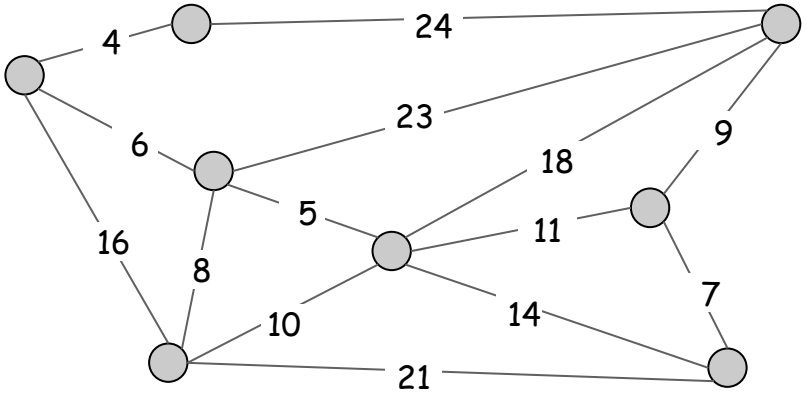
- un albero di copertura (uno spanning tree) di G, ovvero un albero $T=(V,F)$ con $F \subseteq E$

Misura della soluzione (da minimizzare):

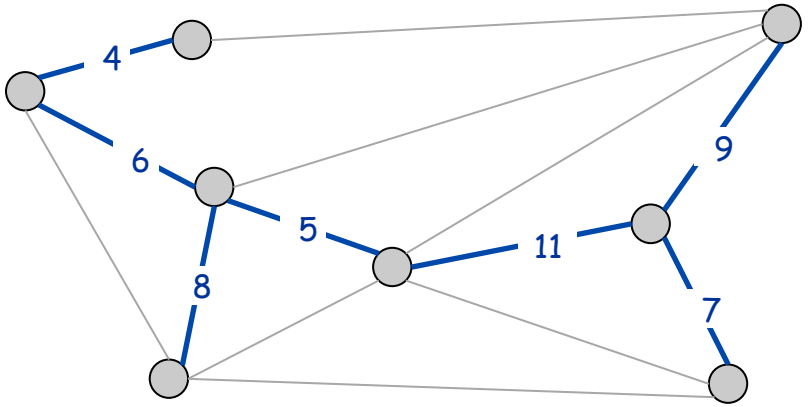
- costo di T: $\sum_{e \in F} w(e)$

Minimum Spanning Tree

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.



$G = (V, E)$



T, $\Sigma_{e \in T} c_e = 50$

Cayley's Theorem. There are n^{n-2} spanning trees of K_n.

↑

can't solve by brute force

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.

Greedy Algorithms

Kruskal's algorithm. Start with $T = \phi$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Remark. All three algorithms produce an MST.

Riepilogo: regole del taglio e del ciclo

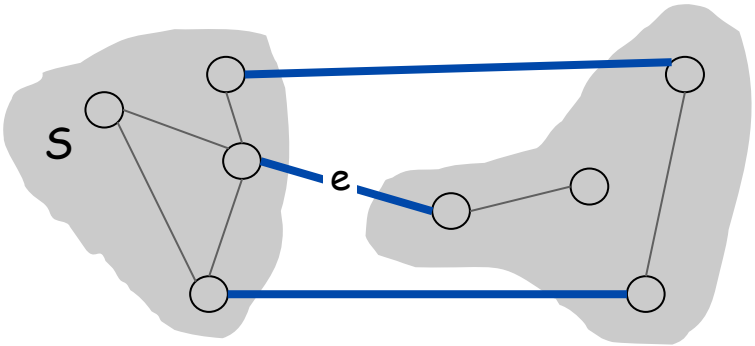
Scegli un taglio del grafo che non è attraversato da **archi blu**. Tra tutti gli archi non ancora colorati che attraversano il taglio, scegline uno di costo minimo e coloralo di blu (cioè, **aggiungilo alla soluzione**).

Scegli un ciclo nel grafo che non contiene **archi rossi**. Tra tutti gli archi non ancora colorati del ciclo, scegline uno di costo massimo e coloralo di rosso (cioè, **scartalo per sempre**).

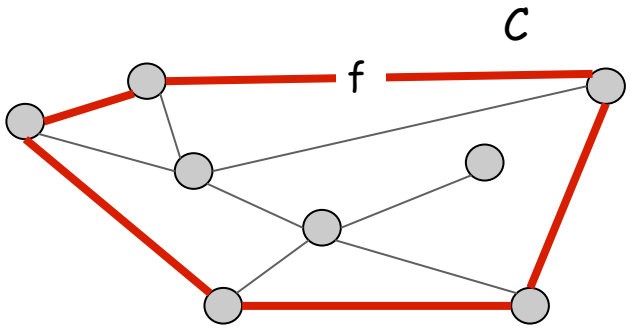
Simplifying assumption. All edge costs c_e are distinct.

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

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.



e is in the MST



f is not in the MST

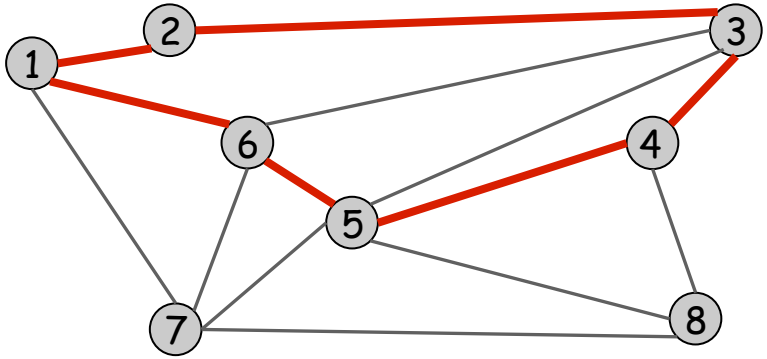
Riepilogo: regole del taglio e del ciclo

Scegli un taglio del grafo che non è attraversato da **archi blu**. Tra tutti gli archi non ancora colorati che attraversano il taglio, scegline uno di costo minimo e coloralo di blu (cioè, **aggiungilo alla soluzione**).

Scegli un ciclo nel grafo che non contiene **archi rossi**. Tra tutti gli archi non ancora colorati del ciclo, scegline uno di costo massimo e coloralo di rosso (cioè, **scartalo per sempre**).

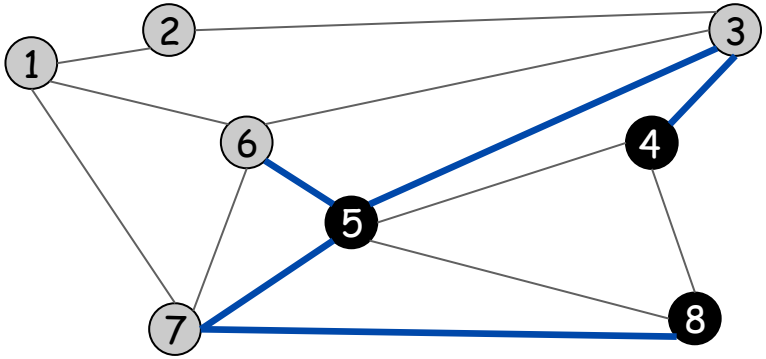
Cycles and Cuts

Cycle. Set of edges the form a-b, b-c, c-d, …, y-z, z-a.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1

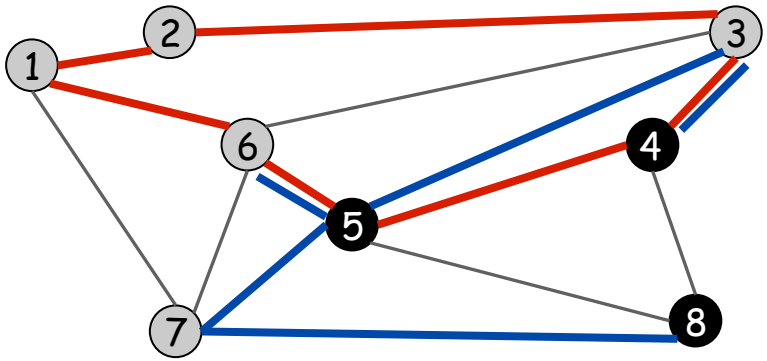
Cutset. **A cut is a subset of nodes S. The corresponding cutset D is the subset of edges with exactly one endpoint in S.**



Cut S = { 4, 5, 8 }
Cutset D = 5-6, 5-7, 3-4, 3-5, 7-8

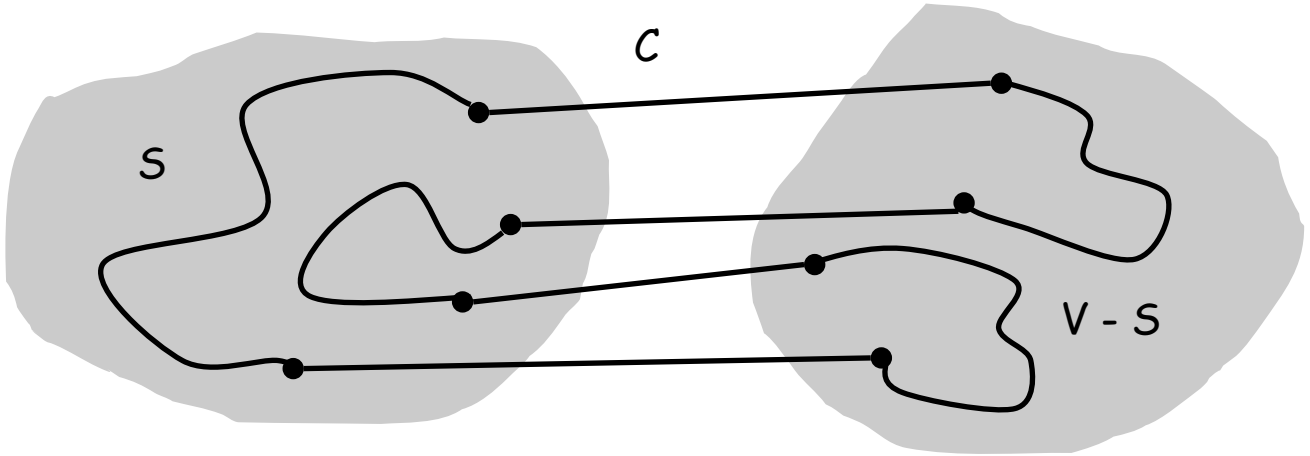
Cycle-Cut Intersection

Claim. A cycle and a cutset intersect in an even number of edges.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

Pf. (by picture)

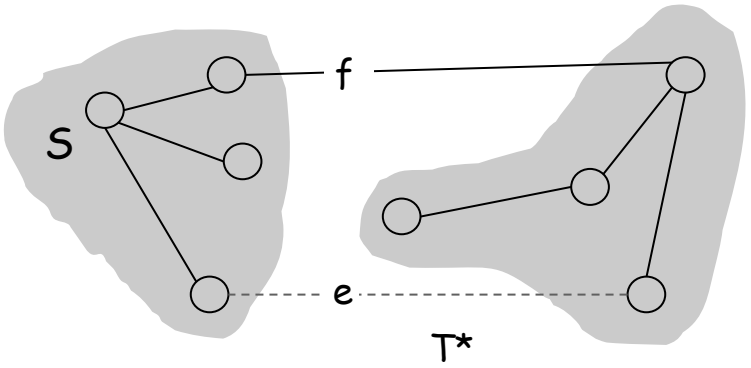


Simplifying assumption. All edge costs c_e are distinct.

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

Pf. (exchange argument)

- Suppose e does not belong to T*, and let's see what happens.
- Adding e to T* creates a cycle C in T*.
- Edge e is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say f, that is in both C and D.
- T' = T* $\cup$ { e } - { f } is also a spanning tree.
- Since $c_e < c_f$, cost(T') < cost(T*).
- This is a contradiction. ▪

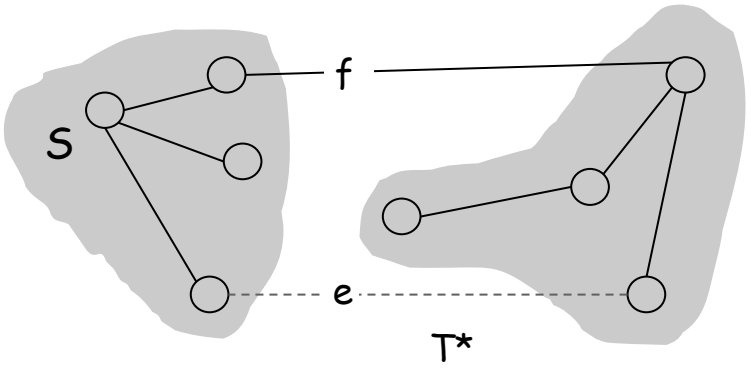


Simplifying assumption. All edge costs c_e are distinct.

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

Pf. (exchange argument)

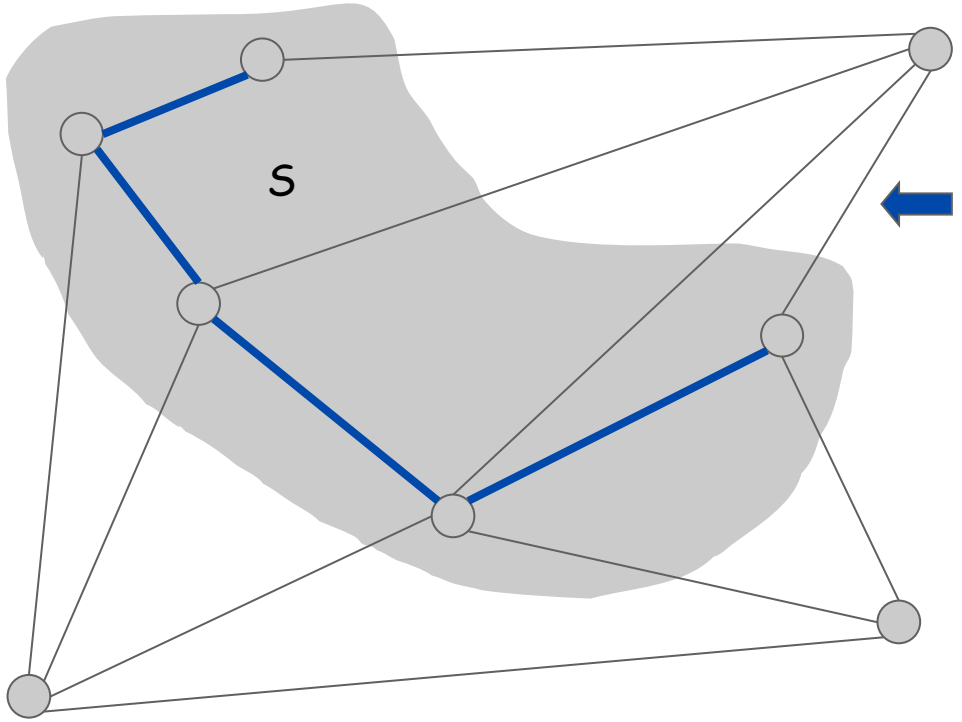
- Suppose f belongs to T^*, and let's see what happens.
- Deleting f from T^* creates a cut S in T^*.
- Edge f is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say e, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e < c_f$, $cost(T') < cost(T^*)$.
- This is a contradiction. ▪



Prim's Algorithm: Proof of Correctness

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

- Initialize S = any node.
- Apply cut property to S.
- Add min cost edge in cutset corresponding to S to T, and add one new explored node u to S.



Implementation: Prim's Algorithm

Implementation. Use a priority queue ala Dijkstra.

- Maintain set of explored nodes S.
- For each unexplored node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

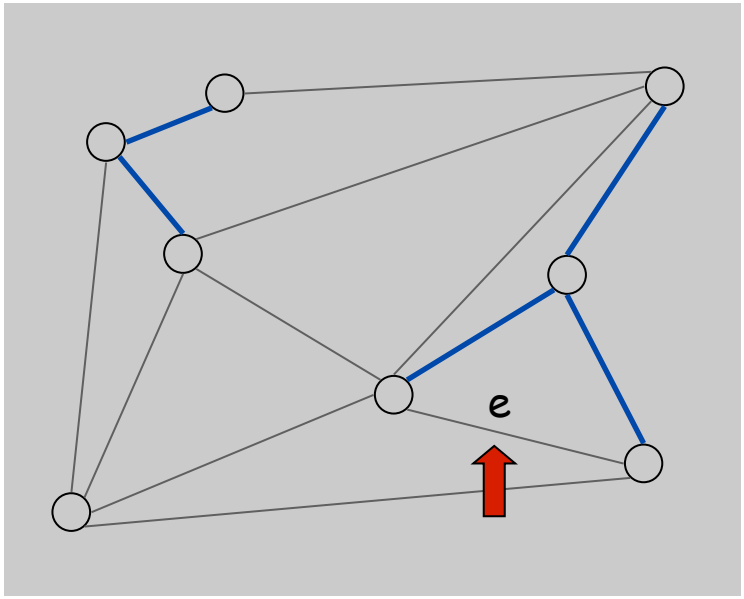
```
Prim(G, c) {
    foreach (v ∈ V) a[v] ← ∞
    Initialize an empty priority queue Q
    foreach (v ∈ V) insert v onto Q
    Initialize set of explored nodes S ← ϕ

    while (Q is not empty) {
        u ← delete min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (c_e < a[v]))
                decrease priority a[v] to c_e
}
```

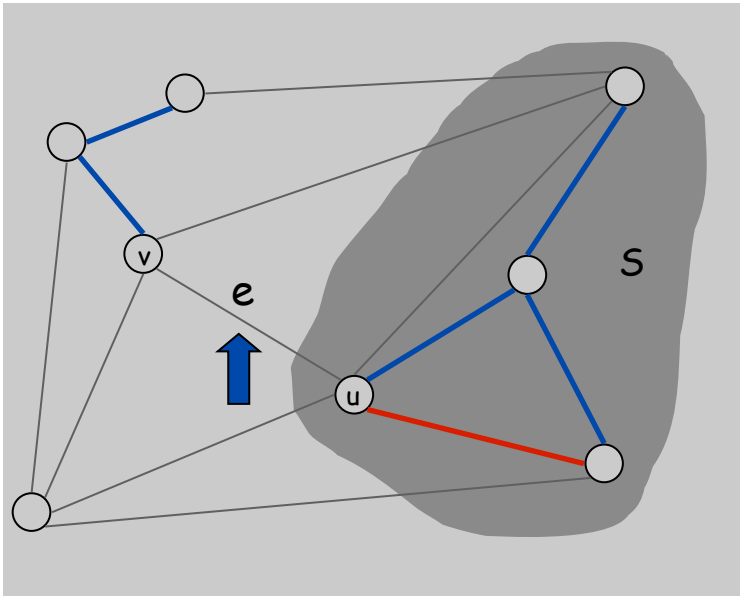
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e according to cycle property.
- Case 2: Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.



Case 1



Case 2

Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- $O(m \log n)$ for sorting and $O(m \alpha(m, n))$ for union-find.

$m \le n^2 \Rightarrow \log m$ is $O(\log n)$

essentially a constant

```
Kruskal(G, c) {
    Sort edges weights so that c_1 ≤ c_2 ≤ ... ≤ c_m.
    T ← φ

    foreach (u ∈ V) make a set containing singleton u

    for i = 1 to m
        (u,v) = e_i
        if (u and v are in different sets) {
            T ← T ∪ {e_i}
            merge the sets containing u and v
        }
    return T
}
```

are u and v in different connected components?

merge two components

NEW DATA STRUCTURE!

WE NOW NEED A NEW DATA STRUCTURE TO MANAGE SUBSET OPERATIONS EFFICIENTLY!

THIS PART FOLLOWS THE BOOK :

Demetrescu et Al,
Algoritmi e Strutture Dati

Lexicographic Tiebreaking

To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

e.g., if all edge costs are integers, perturbing cost of edge e_i by i / n^2

Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

```
boolean less(i, j) {
   if      (cost(e_i) < cost(e_j)) return true
   else if (cost(e_i) > cost(e_j)) return false
   else if (i < j)                 return true
   else                            return false
}
```

Small perturbations of edge costs have no impacts!

Assume $c(e) \geq 1$ for all $e \in E$ and fix $b = 1/n^2$. Now, consider the new instance $I_b = \langle G(V,E), c_b : E \to R+ \rangle$ such that:

$c_b(e) = c(e) \pm b$ and $c(e) \neq c(e')$ for any $e \neq e'$ (all distinct!)

THM. Let T_b be any MST for I_b then T_b is also a MST for the original instance $I = \langle G(V,E), c : E \to R+ \rangle$

Proof. By contradiction. Absurd hyp: exists T^* better than T_b for I,

We use the following facts:

I) $C(T^*, I_b) < C(T^*, I) + 1/n$ (since b=1/n^2)

II) $C(T^*, I) \leq C(T_b, I) - 1$ (absurd hyp.)

III) $C(T_b, I) < C(T_b, I_b) + 1/n$ (since b=1/n^2)

(I) ← (II) → IV) $C(T^*, I_b) < C(T_b, I) - 1 + 1/n$

(IV)←(III) → $C(T^*, I_b) < C(T_b, I_b) + 1/n - 1 + 1/n$ (for $n > 2$)

$< C(T_b, I_b)$ (absurd!)

⊙

EXCERCISE N.1 (MST Properties)

Input: Connected Graph G(V,E); $e \in E$.
Output: Decide whether an MST T exists s.t. $e \in T$

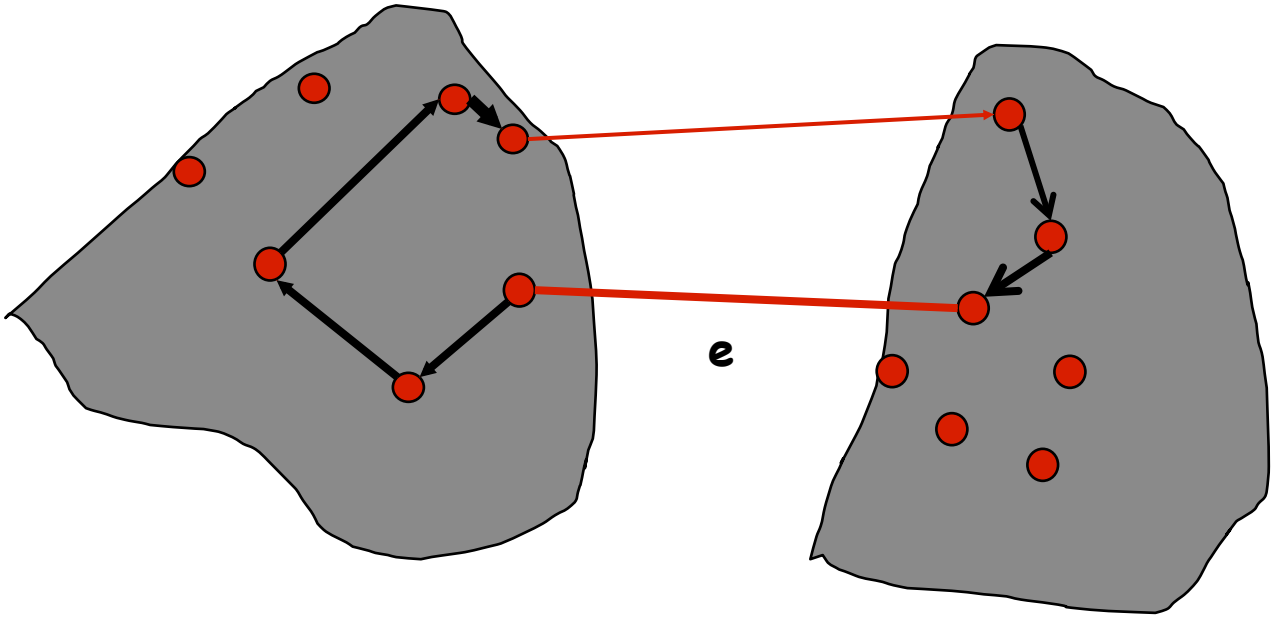
Provide an algorithm working in O(m+n) time.

Hint: **Combine the CUT Property and the CYCLE one to decide whether** e is a MINIMAL BRIDGE.

See this next lesson!

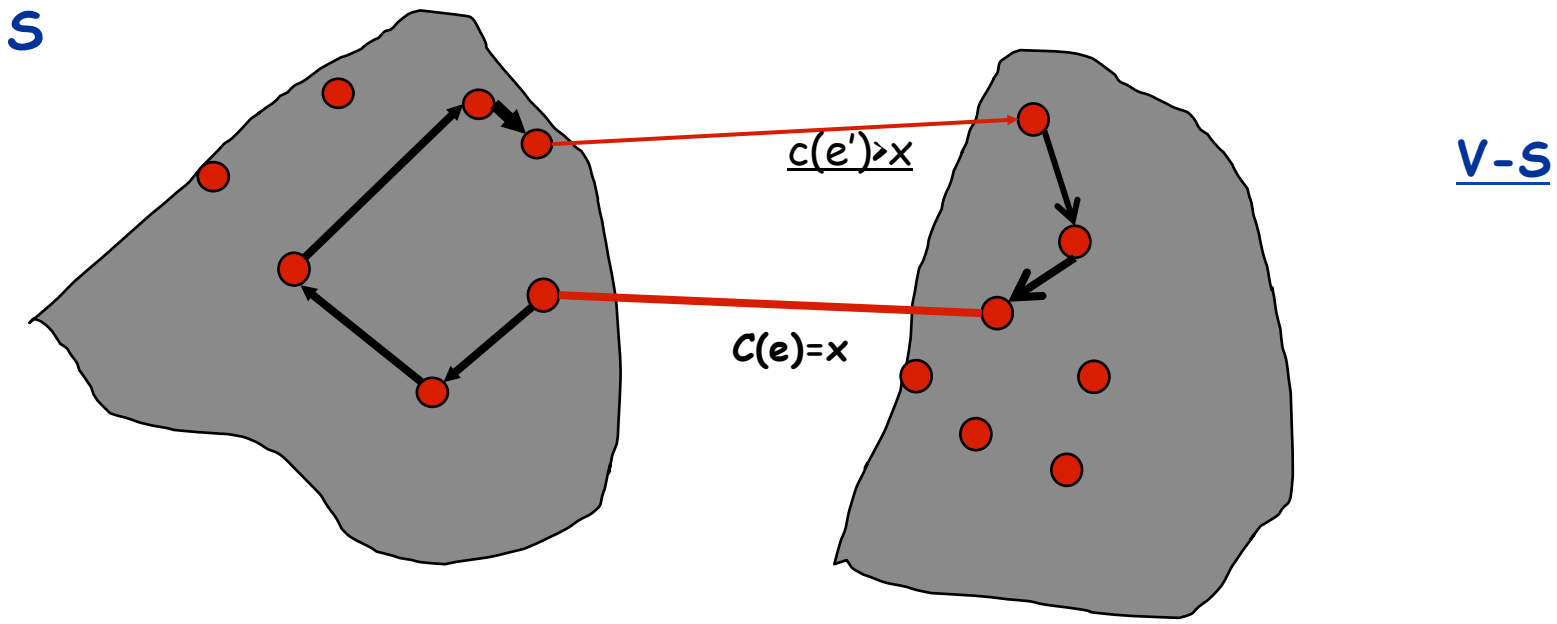
Let **e** := **(u,w)** and **c(e)** := **c**

I) case: there is a Path (forming a cycle with **e**) where **e** is the most expensive!



Then, by the cycle property → **e** cannot belong to any MST

II case: the Set **S** of all nodes reachable from **u** with edges cheaper than **x** does not contain **w** and, so, the set **V-S** contains **w** and all nodes reachable from **w** with edges cheaper than **x**



Then, by the cut property → **e** belongs to any MST

Excersise n.2

Prove or Confute the following Statements:

a) Let $\langle G(V,E)\ w\rangle$ be s.t. G is connected and all edges have distinct weights. Let e^* be the edge of minimal weight. Does e^* always belong to an MST ?

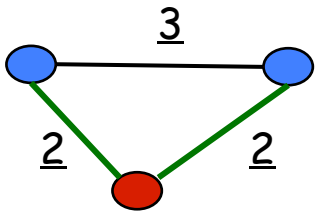
b) Let T be an MST for $\langle G(V,E),w\rangle$ and consider the NEW instance $\langle G(V,E),w^2\rangle$ where

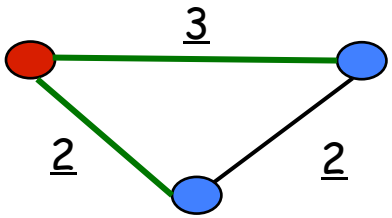
$$\text{for any } e \in E : w^2(e) = (w(e))^2$$

Is T an MST for $\langle G(V,E),w^2\rangle$ as well?

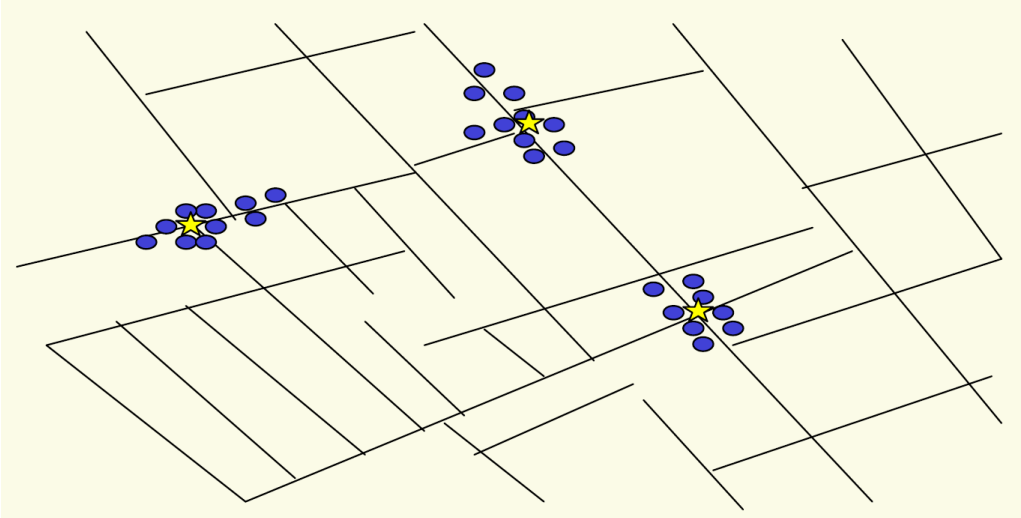
Shortest Path Tree vs Minimum Spanning Tree

- Consider an **instance** of the **MST** problem: **<G(V,E);c: E → R+>**
- **FACT 1:** The MST ***T(s)*** computed by Prim's Algorithm may depend by the source node **s**. But **T(s)** is a feasible and optimal solution for instance <G(V,E);c: E → R+>, **for any choice of s' in V.**
- Consider an **instance** of the **SPT** problem: **<G(V,E);c: E → R+;s>**
- **FACT 2: The SPT T(s) (and its cost) computed by *any* correct algorithm may depend on the choice of s**





4.7 Clustering



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

Clustering

Clustering. Given a set U of n objects labeled p_1, …, p_n, classify into coherent groups.

↑ photos, documents. micro-organisms

Distance function. Numeric value specifying "closeness" of two objects:

distance(p_i, pj)

↑ number of corresponding pixels whose intensities differ by some threshold

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

- 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.

Clustering of Maximum Spacing

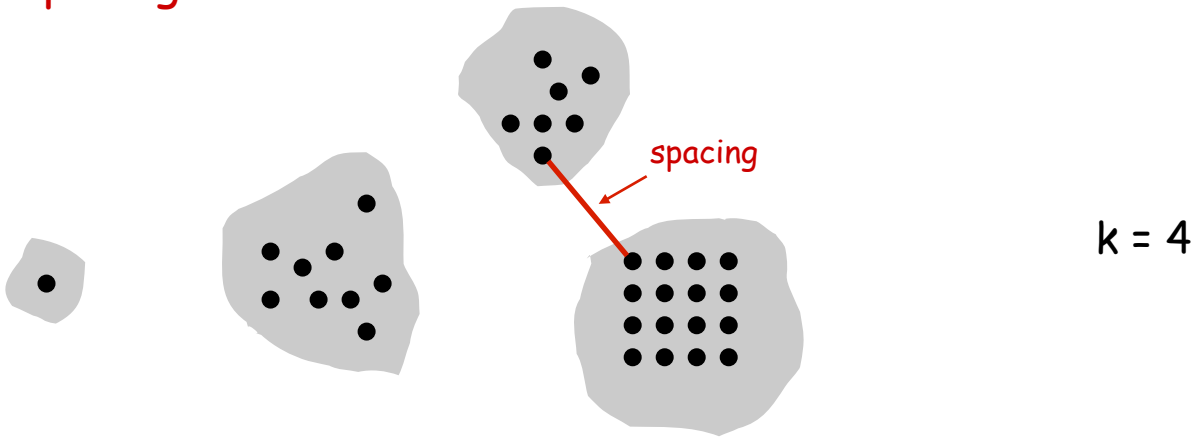
k-clustering. Divide objects into k non-empty groups.

Distance function. Assume it satisfies several natural properties.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

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

Clustering of maximum spacing. Given an integer k, find a k-clustering of maximum spacing.



Greedy Clustering Algorithm

Single-link k-clustering algorithm.

- Form a graph on the vertex set **U**, corresponding to **n** clusters.
- Find the **closest** pair of objects (p,p') such that p & p' are not in the same cluster, and add an edge between them: so merging 2 clusters.
- Repeat **n-k** times until there are exactly **k** clusters.

Key Obs. 1. This procedure is precisely Kruskal's algorithm (except we stop when there are **k** connected components).

Key Obs. 2. Equivalent to finding an MST **T** and deleting the **k-1** most expensive edges from **T** (thus forming **k** connected components).

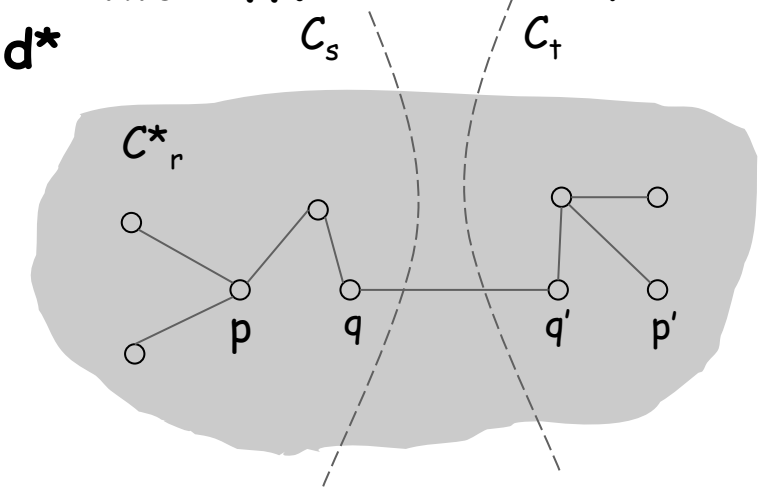
(Proofs of 1 and 2: Excercises for STAGE STUDENTS - CFU F)

Greedy Clustering Algorithm: Analysis

Theorem. Let C^* denote the clustering $C^*_1, \ldots, C^*_k$ formed by deleting the **k-1** most expensive edges of an MST. Then, C^* is a k-clustering of **maximal spacing**.

Pf. Let C denote some other clustering $C_1, \ldots, C_k$.

- The spacing of C^* is the length **d*** of the **(k-1)st most expensive edge**.
- Let p, p' be in the same cluster in C^*, say C^*_r, but different clusters in C, say C_s and C_t.
- Some edge (q, q') on p-->p' path in C^*_r spans **two diff.** clusters in C.
- **All** edges on p-->p' path have length ≤ **d*** since Kruskal chooses them.
- Spacing of C is ≤ **d*** since q and q' are in **different** clusters of C. ▪

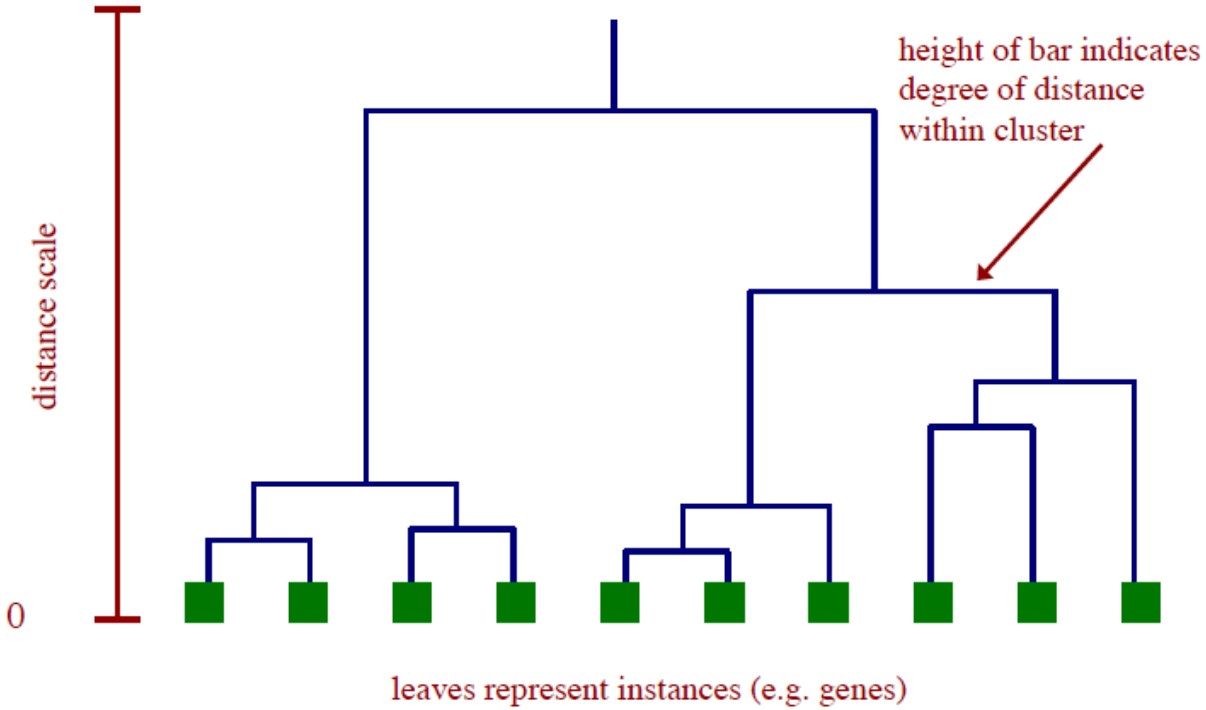


Applications: Genetic

Dendrogram

Dendrogram. Scientific visualization of hypothetical sequence of evolutionary events.

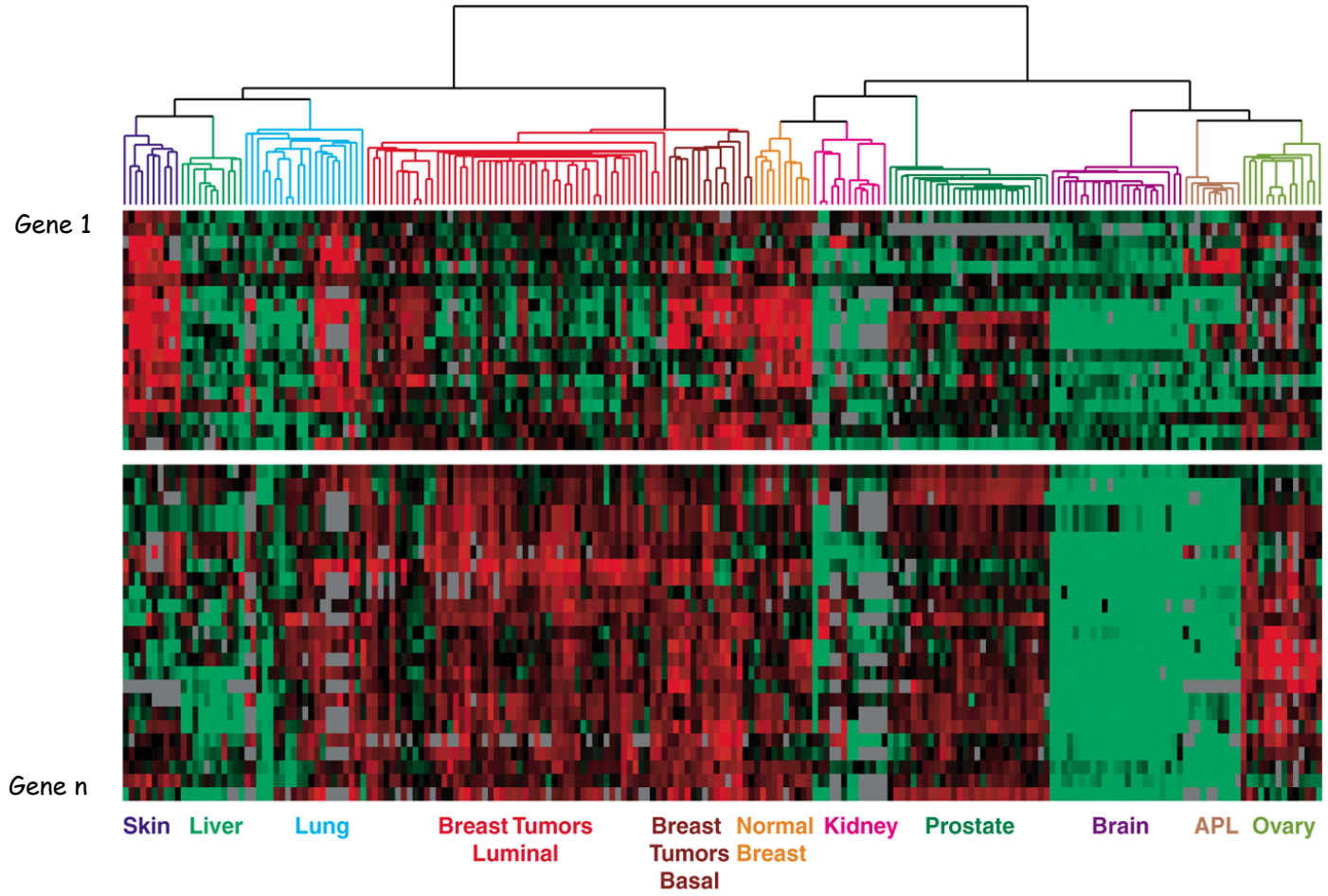
- Leaves = genes.
- Internal nodes = hypothetical ancestors.



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

Exercise 1 (KT p. 188)

INPUT: $G(V,E)$; $c:E \rightarrow R^+$ (all distinct); an edge **e** in E;
QUESTION: Decide whether **e** belongs to some MST.

Apply the two main properties of MST:

1) CUT PROPERTY: if **e** is the cheapest bridge in some cut **(S,V-S)** then **e** belongs to some **MST**

2) CYCLE PROPERTY: if **e** belongs to some cycle formed by edges, all cheaper than **e**, then **e** does not belong to any **MST**

How can we combine (1) and (2) to set the question?

How detect which is the case for input $G(V,E)$;$c:E \to R^+$; **e=(u,w)** in E ?

Simple Idea:

1) **Remove** all edges from E which are more expensive than **e** and remove also **e**. **Set** **G'(V,E')** as this new graph.

2) **Start** a *BFS* (or a *DFS*) from u (or from w) over **G'(V,E')**

3) **If** the computed Tree **T(u)** contains **w** then **return**: e does not belong to any MST, otherwise **return:** it does!