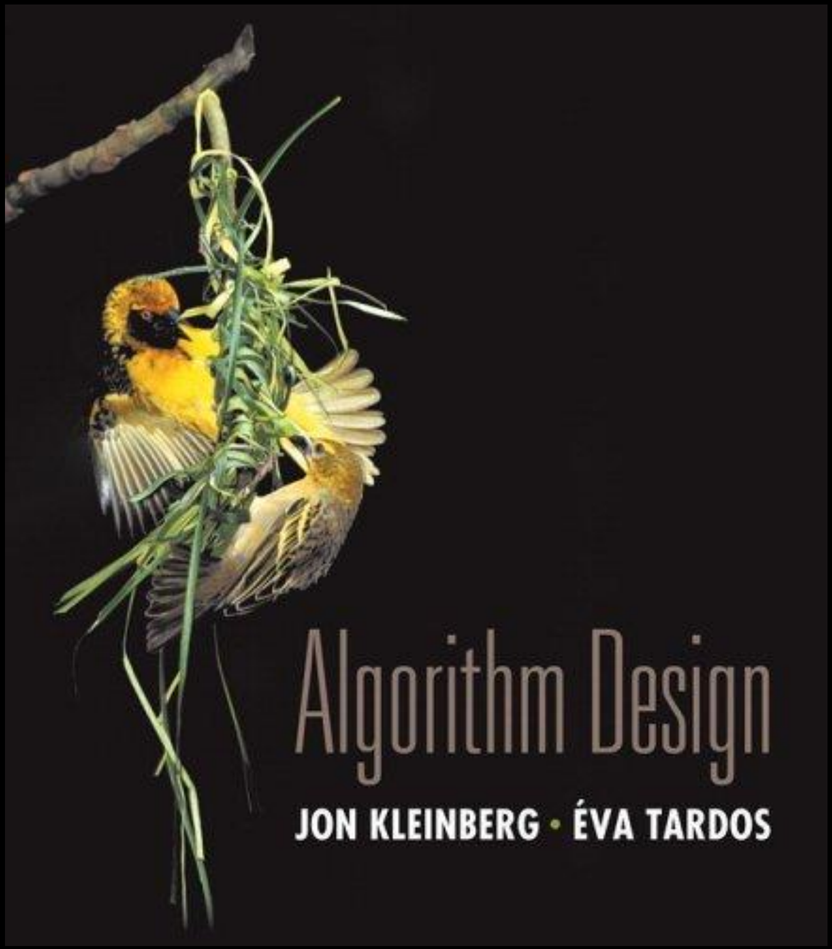


Chapter 4

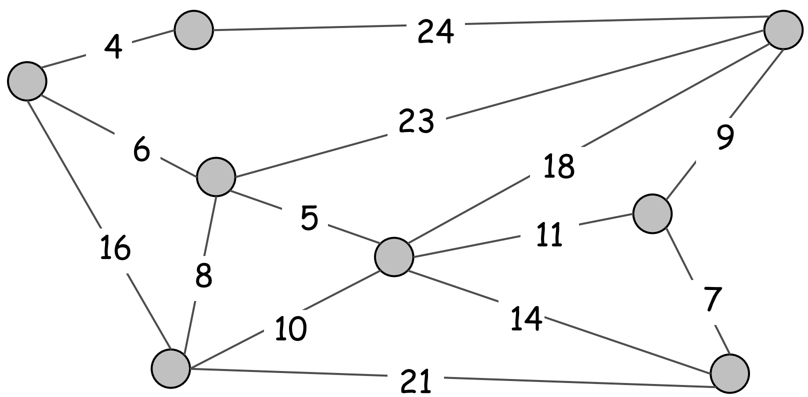
Greedy Algorithms

Slides by Kevin Wayne.

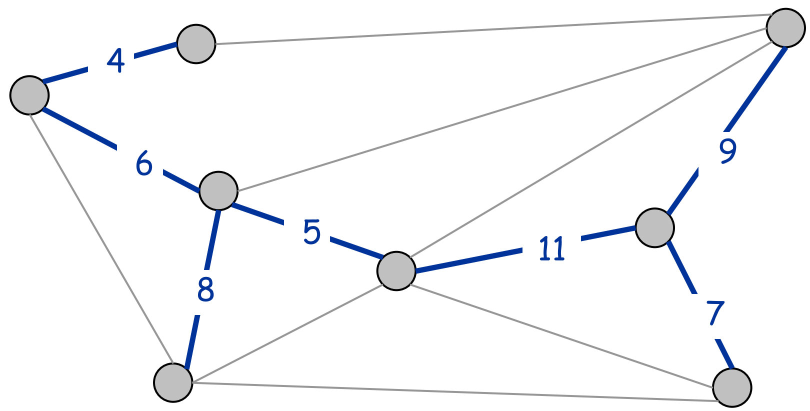
4.5 Minimum Spanning Tree

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

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.

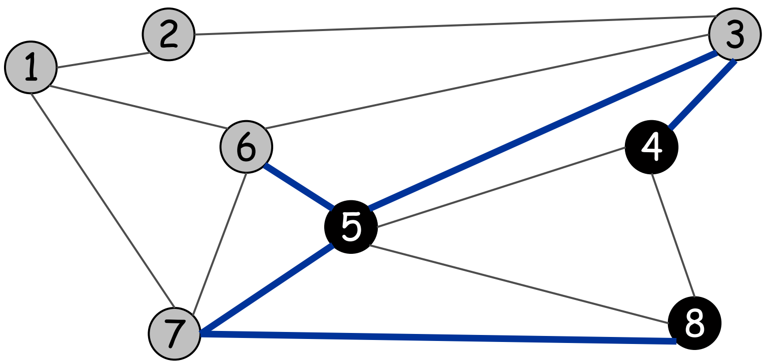
Reverse-Delete algorithm. Start with $T = E$. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

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.

Cycles and Cuts

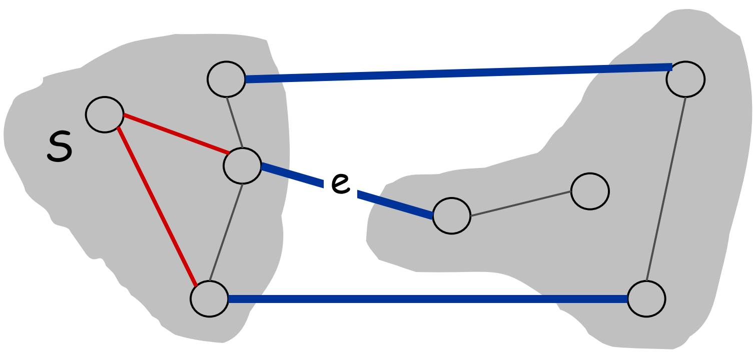
Cut. A cut for a graph G=(V,E) is a subset of nodes S.



Cut S = { 4, 5, 8 }
Crossing edges = 5-6, 5-7, 3-4, 3-5, 7-8

- An edge **e** cross a cut S if **e** has an edpoint in S and the other one in V-S.
- A cut S is crossed by X if some edge of X crosses S

Cut property. Suppose X is included in the set of edges of a MST for G=(V,E). Pick a cut S that is not crossed by X. If **e** is the lightest edge that crosses S then $X \cup$ **e** is also part of a MST.



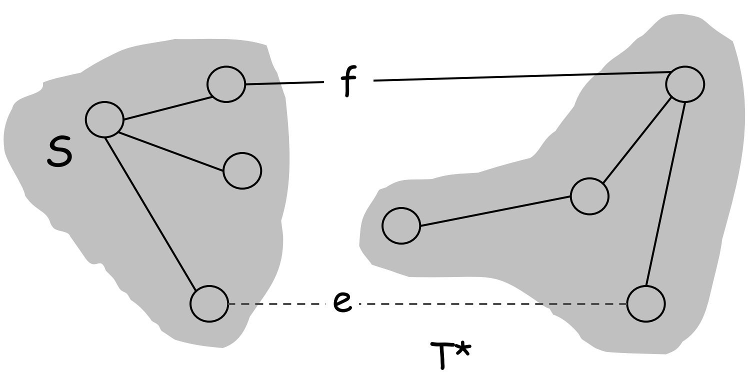
X = red edges
e is in the MST

Greedy Algorithms

Cut property. Suppose X is included in the set of edges of a MST for G=(V,E). Pick a cut S that is not crossed by X. If **e** is the lightest edge that crosses S then $X \cup$ **e** is also part of a MST.

Pf. (exchange argument)

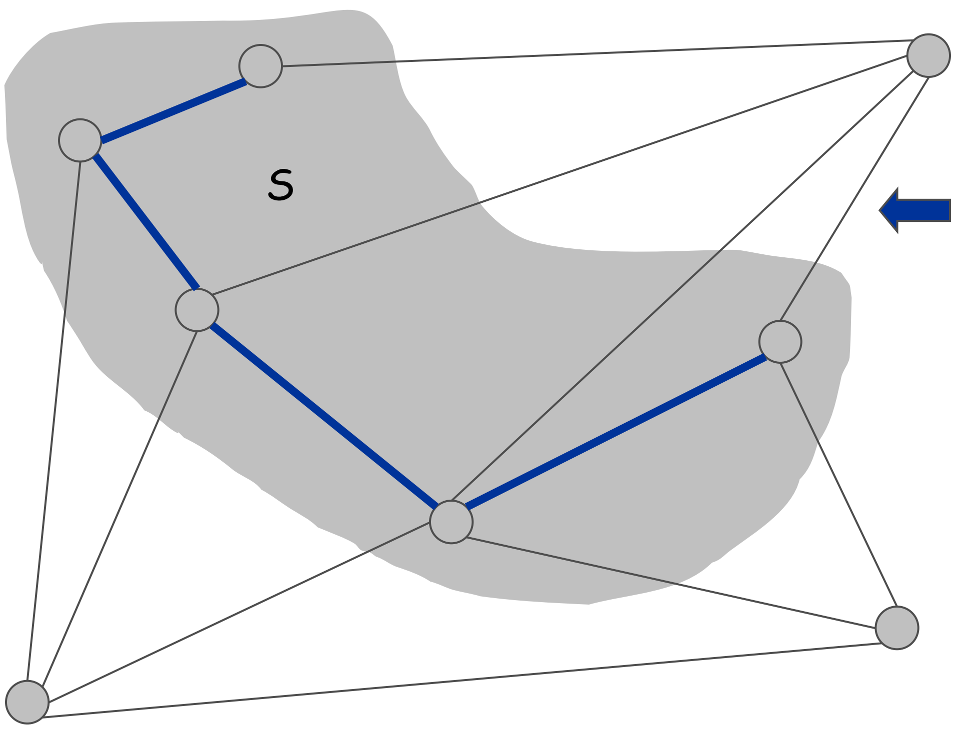
- Let T*=(V,E*) be a MST for G such that $X \subseteq E^*$
- Suppose e does not belong to T*, and let's see what happens.
- Adding **e** to T* creates a cycle C in $T^* \cup$ **e** that contains e.
- There exists another edge, say f, that is in C and cross S.
- f does not belong to X because X does not cross S
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e <= c_f$, cost(T') <= cost(T*).
- T' is also a MST and $X \cup e$ is part of it ▪



Prim's Algorithm

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

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



Bad Implementation: Prim's Algorithm

Implementation (Naïve)

- Maintain set of explored nodes S.
- Find the lightest edge that crosses S in O(m) time
- Total complexity O(m.n)

Good Implementation: Prim's Algorithm

Implementation. Use a priority queue.

- 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.

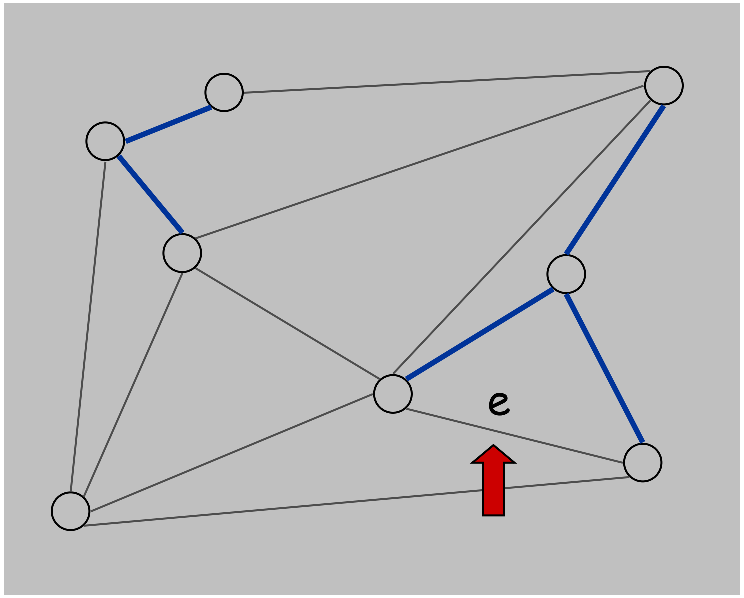
```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q
   Initialize set of explored nodes S ← φ

   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
}
```

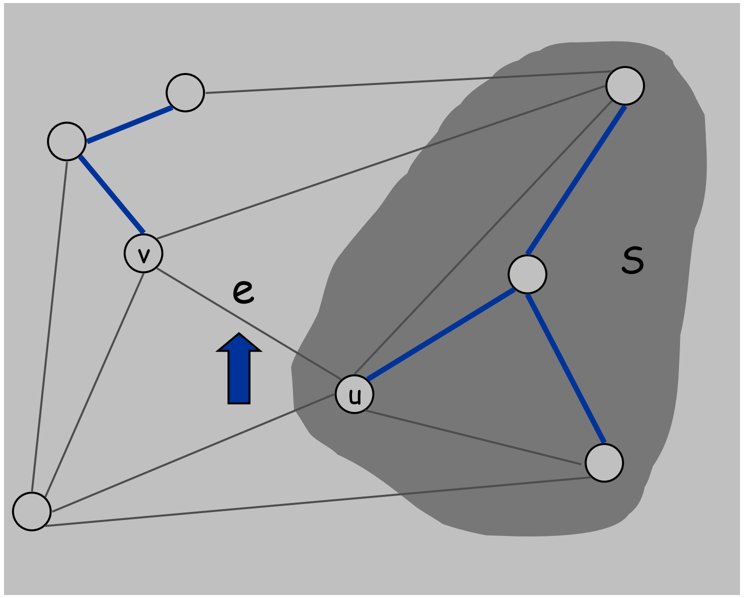
Kruskal's Algorithm

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e
- Case 2: Otherwise, insert e = (u, v) into T



Case 1

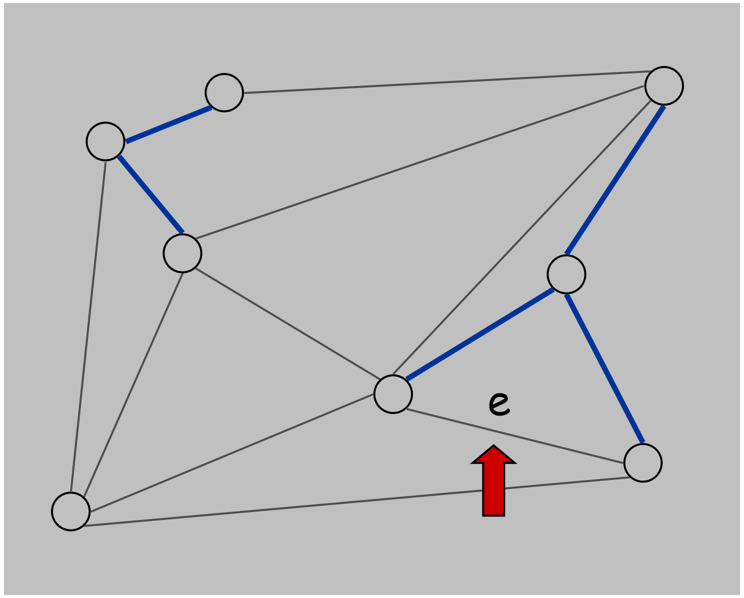


Case 2

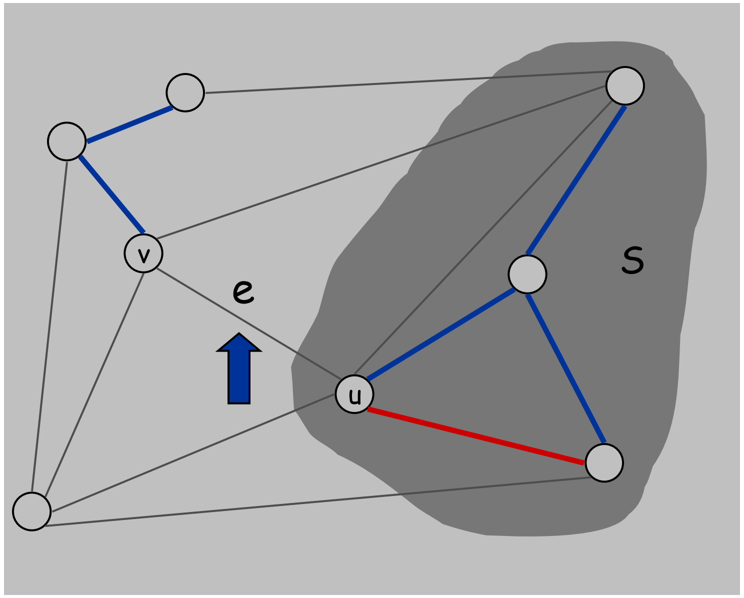
Kruskal's Algorithm: Proof of correctness

Kruskal's algorithm. [Kruskal, 1956]

- Case 1: If adding e to T creates a cycle, discard e
  - Optimal solution does not have a cycle
- Case 2: Otherwise, insert e = (u, v) into T
  - Pick the cut S as the nodes that are reachable from u in T



Case 1



Case 2

Kruskal's Algorithm: Bad Implementation

Kruskal's algorithm. [Kruskal, 1956]

- Sorting the edges O(m log m)
- Testing the existence of a cycle while considering edge e: O(n) via a DFS(BFS). Note that a tree has at most n edges.
- For all edges O(m.n)
- Total complexity O(m log m) +O(m n) = O(n.m)

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 $O(m\, \underbrace{\alpha\,(m, n)}_{\text{essentially a constant}})$ for union-find.

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

```
Kruskal(G, c) {
   Sort edges weights so that c1 ≤ c2 ≤ ... ≤ cm.
   T ← φ

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

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

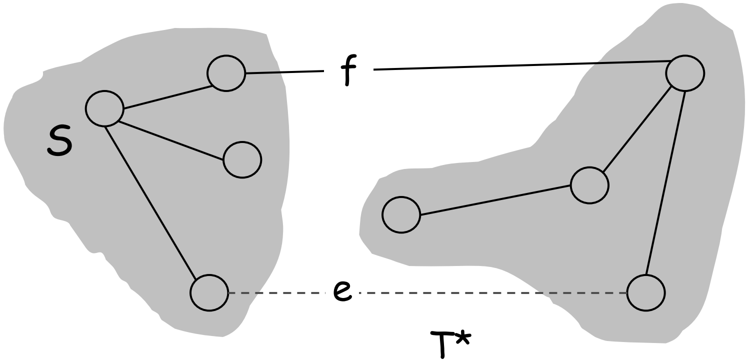
are u and v in different connected components?

merge two components

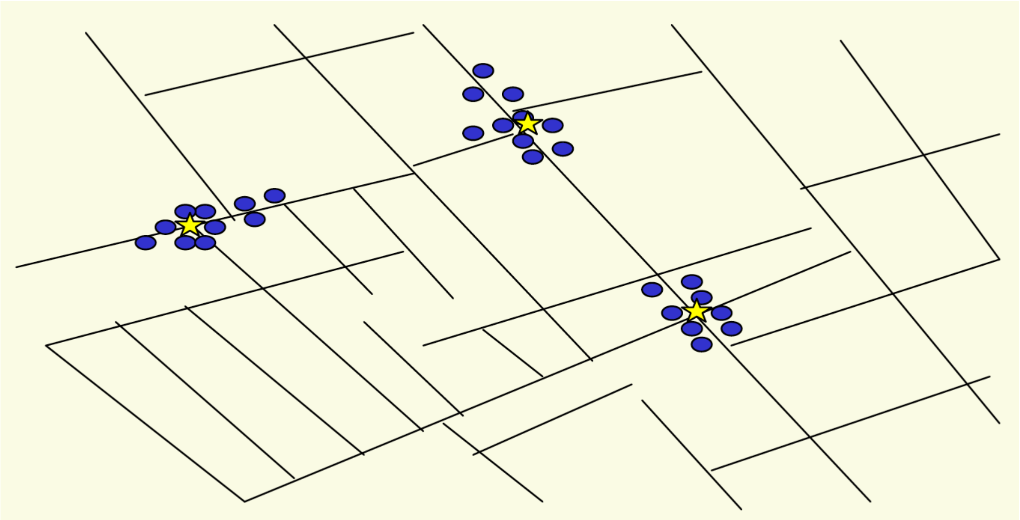
Cycle property. Let C be any cycle in G, and let f be an edge belonging to C. If cost(f)>cost(e) for every e in C, then f cannot belong to a MST.

Pf. (exchange argument)

- Suppose f belongs to a MST T*, and let's see what happens.
- Deleting f from T* creates two connected components.
- There is at least one edge in C-f, say e, that connects the two compoents. Thus,
- T' = T* $\cup$ { e } - { f } is also a spanning tree.
- Since $c_e < c_f$, cost(T') < cost(T*).
- This is a contradiction. ▪



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 $p_1, \ldots, p_n$, classify into coherent groups.

photos, documents. micro-organisms

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

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.

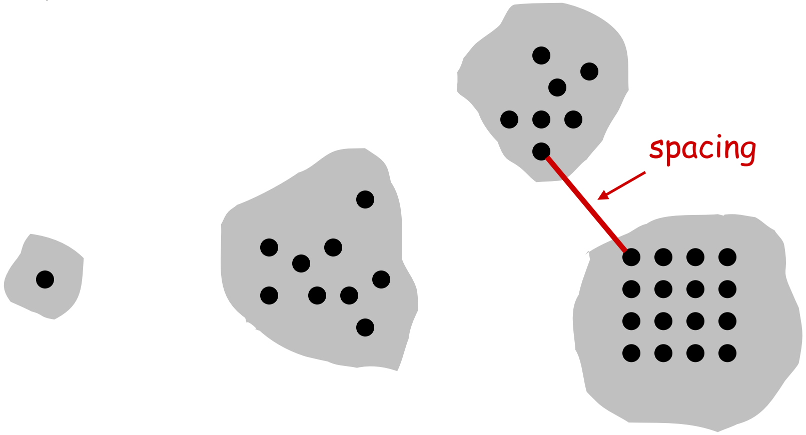
Clustering of Maximum Spacing

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)

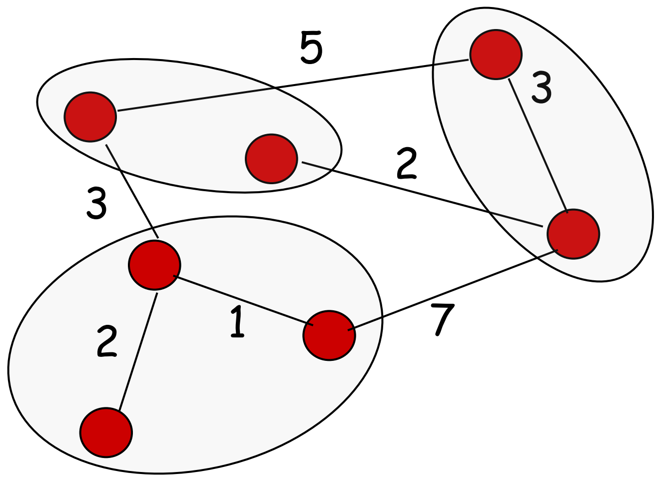
Spacing. Min distance between any pair of points in different clusters.

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



k = 4

Clustering of Maximum Spacing



K=3

Spacing = 2

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 such that each object is in a different cluster, and add an edge between them.
- Repeat n-k times until there are exactly k clusters.

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

Remark. Equivalent to finding an MST and deleting the k-1 most expensive edges.

Greedy Clustering Algorithm: Analysis

Lemma. If $C=\{C_1, \ldots, C_k\}$ and $C'=\{C'_1, \ldots, C'_k\}$ are two different clustering then there exists a pair of points pi,pj such that pi and pj are in the same cluster in C and in different clusters in C'

Pf. Homework

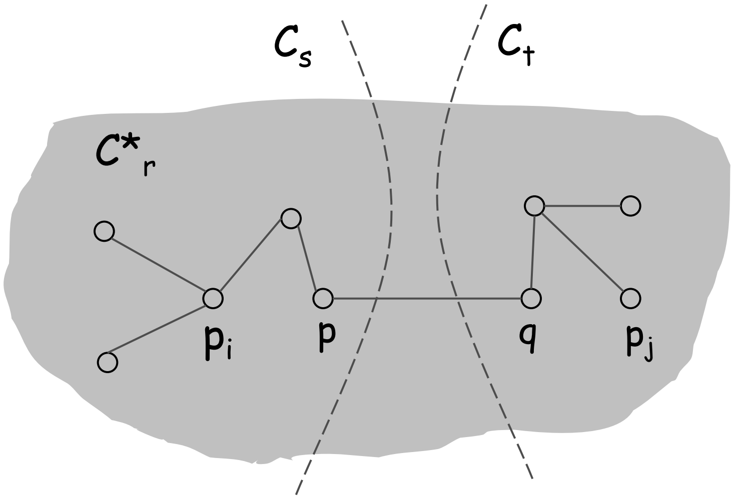
.

Greedy Clustering Algorithm: Analysis

Theorem. Let C^* denote the clustering $C^*_1, \ldots, C^*_k$ formed by the single k-link algorithm. C^* is a k-clustering of max spacing.

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

- The spacing of C^* is the length d^* of lightest edge that connects two cluster of C^*.
- Let p_i, p_j be in the same cluster in C^*, say C^*_r, but different clusters in C, say C_s and C_t.
- Some edge (p, q) on p_i-p_j path in C^*_r connects two different clusters in C.
- All edges on p_i-p_j path have length $\leq d^*$ since Kruskal chose them.
- Spacing of C is $\leq d^*$ since p and q are in different clusters. ▪



Extra Slides

MST Algorithms: Theory

Deterministic comparison based algorithms.

- $O(m \log n)$ [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- $O(m \log \log n)$. [Cheriton-Tarjan 1976, Yao 1975]
- $O(m\, \beta(m, n))$. [Fredman-Tarjan 1987]
- $O(m \log \beta(m, n))$. [Gabow-Galil-Spencer-Tarjan 1986]
- $O(m\, \alpha\, (m, n))$. [Chazelle 2000]

Holy grail. $O(m)$.

Notable.

- $O(m)$ randomized. [Karger-Klein-Tarjan 1995]
- $O(m)$ verification. [Dixon-Rauch-Tarjan 1992]

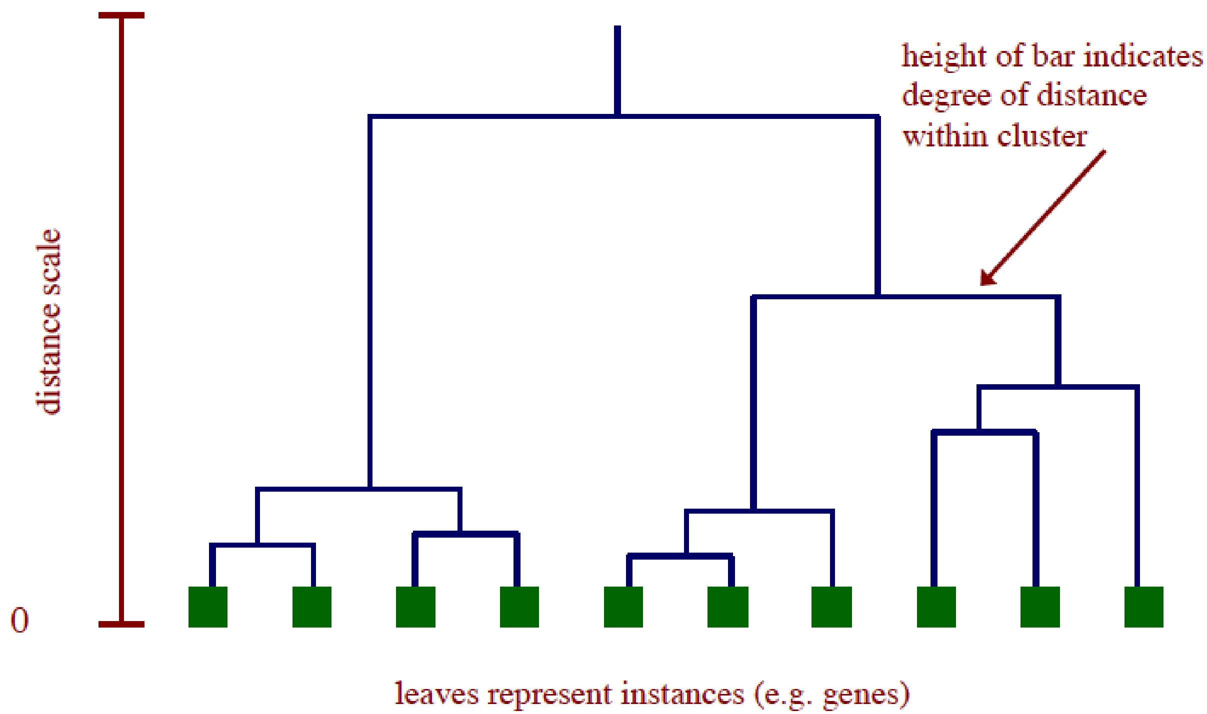
Euclidean.

- 2-d: $O(n \log n)$. compute MST of edges in Delaunay
- k-d: $O(k\, n^2)$. dense Prim

Dendrogram

Dendrogram. Scientific visualization of hypothetical sequence of evolutionary events.

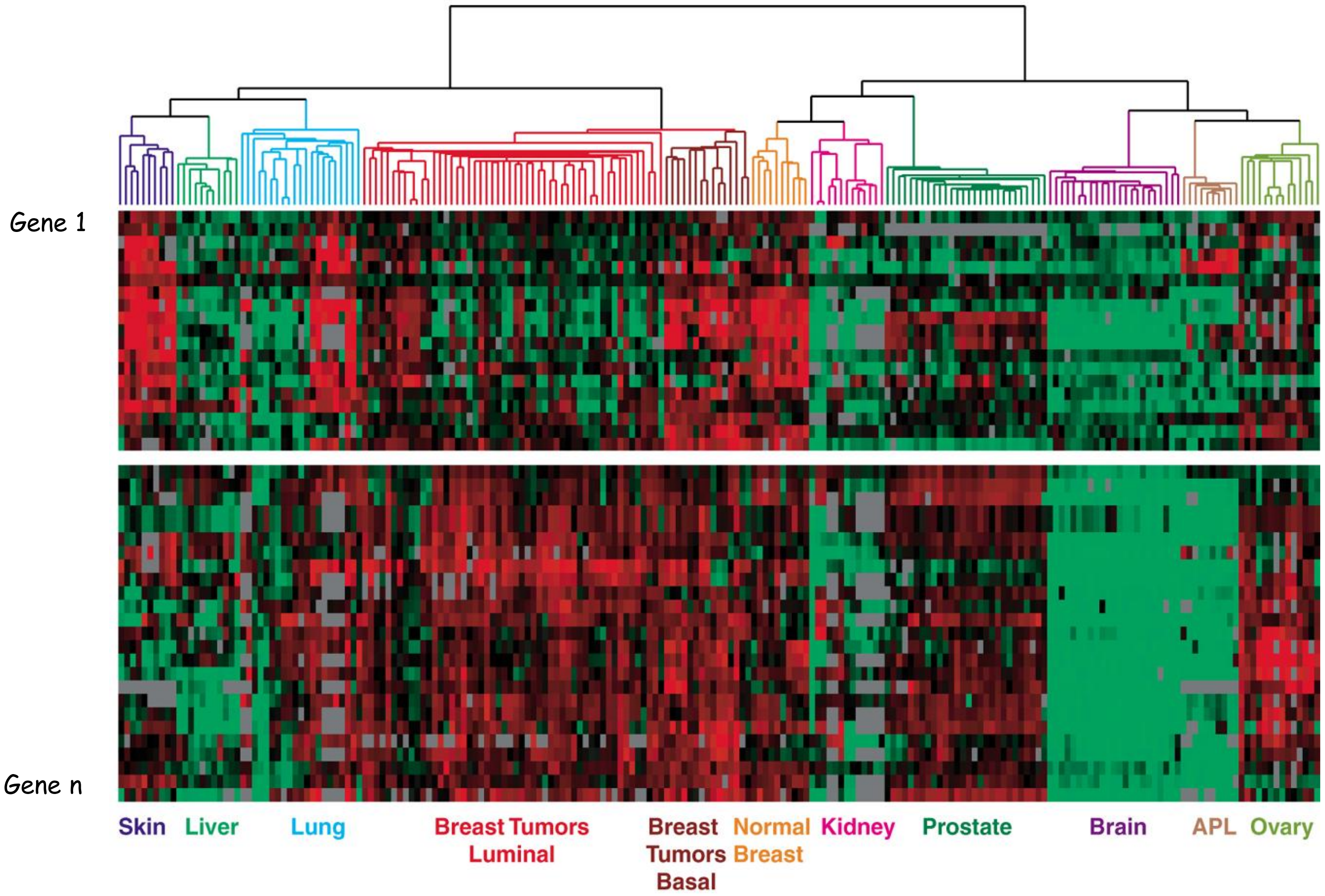
- Leaves = genes.
- Internal nodes = hypothetical ancestors.



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

gene expressed

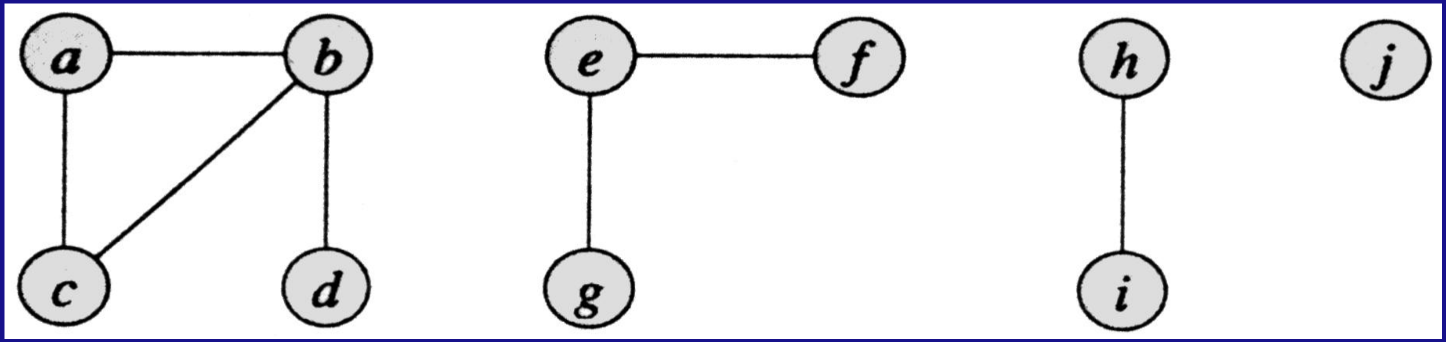
gene not expressed

Union-Find on disjoint sets

Motivation

- Perform repeated union and find operations on disjoint data sets.
- Examples:
  - Kruskal's MST algorithm
  - Connected Components
- Goal: define an ADT that supports Union-Find queries on disjoint data sets efficiently.

Example: connected components



- *Initial set* $\mathbf{S} = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}, \{j\}\}$
- (b,d) $\mathbf{S} = \{\{a\}, \{b,d\}, \{c\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}, \{j\}\}$
- (e,g) $\mathbf{S} = \{\{a\}, \{b,d\}, \{c\}, \{e,g\}, \{f\}, \{h\}, \{i\}, \{j\}\}$
- (a,c) $\mathbf{S} = \{\{a,c\}, \{b,d\}, \{e,g\}, \{f\}, \{h\}, \{i\}, \{j\}\}$
- (h,i) $\mathbf{S} = \{\{a,c\}, \{b,d\}, \{e,g\}, \{f\}, \{h,i\}, \{j\}\}$
- (a,b) $\mathbf{S} = \{\{a,c,b,d\}, \{e,g\}, \{f\}, \{h,i\}, \{j\}\}$
- (e,f) $\mathbf{S} = \{\{a,c,b,d\}, \{e,f,g\}, \{h,i\}, \{j\}\}$
- (b,c) $\mathbf{S} = \{\{a,c,b,d\}, \{e,f,g\}, \{h,\mathrm{i}\}, \{j\}\}$

Union-Find Abstract Data Type

- Let $S = \{S_1, S_2, \ldots, S_k\}$ be a dynamic collection of disjoint sets.
- Each set S_i is identified by a representative member.
- Operations:

  Make-Set(x): create a new set S_x, whose only member is x (assuming x is not already in one of the sets).

  Union(x, y): replace two disjoint sets S_x and S_y represented by x and y by their union.

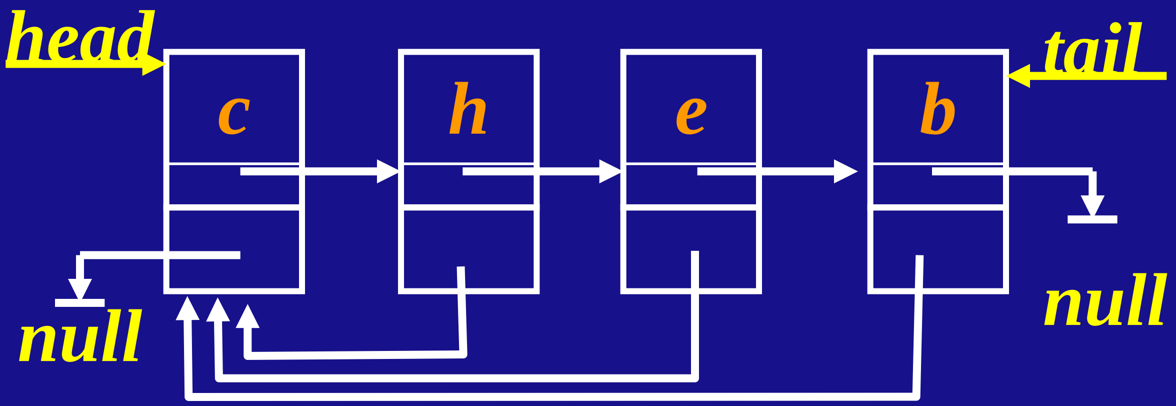
  Find-Set(x): find and return the representative of the set S_x that contains x.

Disjoint sets: linked list representation

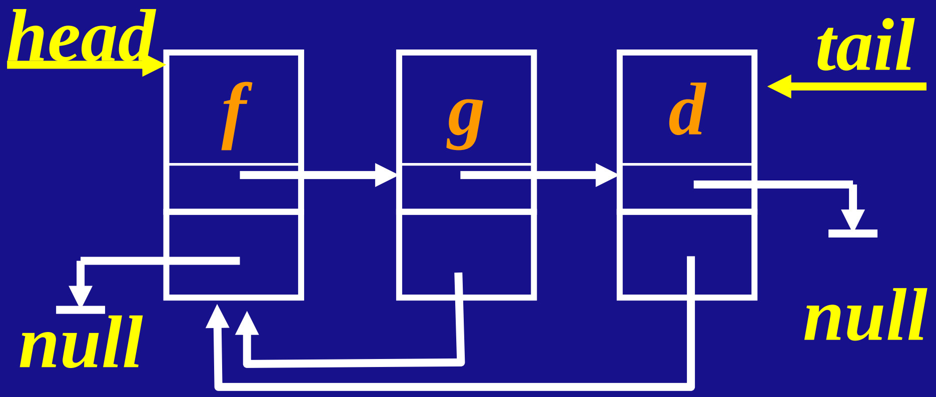
- Each set is a linked list, and the representative is the head of the list. Elements point to the successor and to the head of the list.
- Make-Set: create a new list: $O(1)$.
- Find-Set: search for an element down the list: $O(1)$.
- Union: link the tail of L_1 to the head of L_2, and make each element of L_2 point to the head of L_1: $O(|L_2|)$.
- A sequence of n Make-Set operations +(n–1) Union operations may take $n+\Sigma i = O(n^2)$ operations.

Example: linked list representation

$S_1=\{c,h,e,b\}$

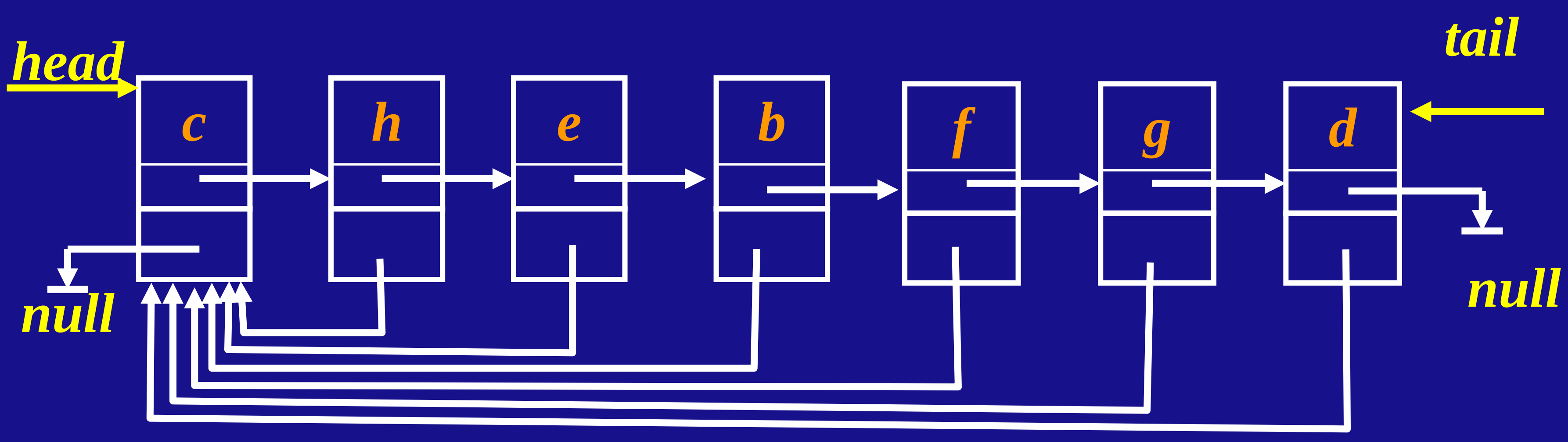


$S_2=\{f,g,d\}$



Example: linked list representation

S_1 U S_2= ={c,h,e,b}U{f,g,d}



Weighted-union heuristic

- When doing the union of two lists, append the shorter one to the longer one.

- A single operation will still take $O(n)$ time.
  - Two lists of n/2 elements

Weighted-union heuristic

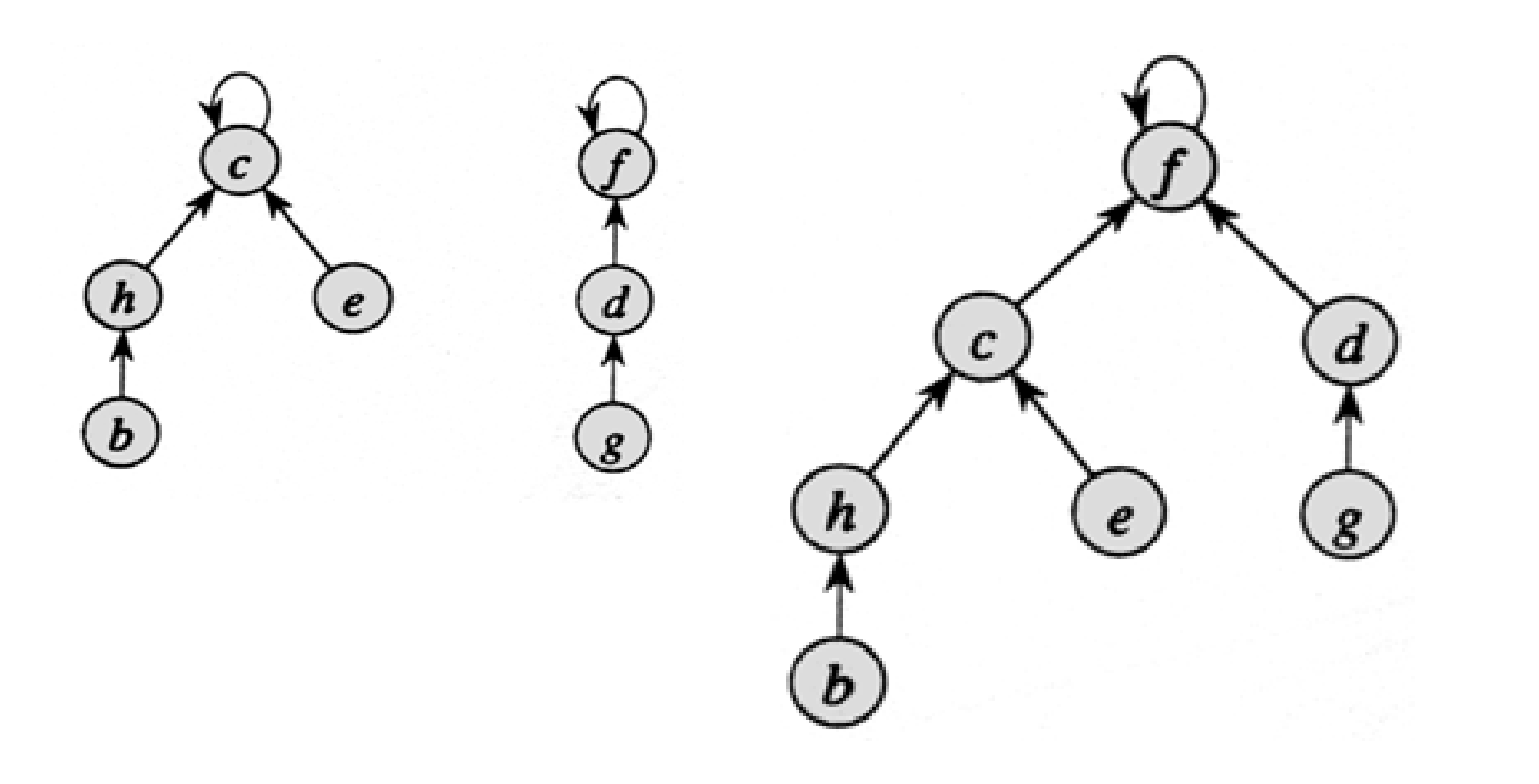
- However, for a sequence of $m>n$ Make-Set, Union, and Find-Set operations, of which n are Make-Set, the total time is $O(m + n\lg n)$ instead of $O(mn)$!
- Proof outline:
  – The total of operations equals the number of times an the objects have its pointes updates
  – Whenever an object x has its pointer updated, the size of the list where x lies doubles. Thus, an object x can only have its pointer updated log n times

Disjoint sets: tree representation

- Each set is a tree, and the representative is the root.
- Each element points to its parent in the tree. The root points to itself.
- Make-Set: takes $O(1)$.
- Find-Set: takes $O(h)$ where h is the height of the tree.
- Union is performed by finding the two roots, and choosing one of the roots, to point to the other. This takes $O(h)$.
- The complexity therefore depends on how the trees are maintained!

Example

$S=\{S_1, S_2\}$, $S_1=\{c,h,e,b\}$, $S_2=\{f,g,d\}$



Union by rank

- We want to make the trees as shallow as possible → trees must be balanced.
- When taking the union of two trees, make the root of the shorter tree become the child of the root of the longer tree.
- Keep track of height of each sub-tree: → keep the *rank* of each node.
- Every time Union is performed, update the rank of the root.

Complexity of Find-Set (1)

- Claim: A tree of height h has at least 2^h nodes
- Proof: Assume that the property holds before the k-th operation and prove that it is true after the the k-th operation.

Complexity of Find-Set (2)

- Case 1) **The tree height does not grow.** It follows that one tree was shorter than the other, in which case it is clearly true, because h didn’t grow and the number of nodes did.

- Case 2) **The height does grow to h+1.** It follows thaat both tree heights were the same. By hypothesis, each sub-tree has at least 2^h nodes, so the new tree has at least $2.2^h = 2^{h+1}$ nodes. Thus, the height of the tree grew by 1 to h+1, which proves the induction step.

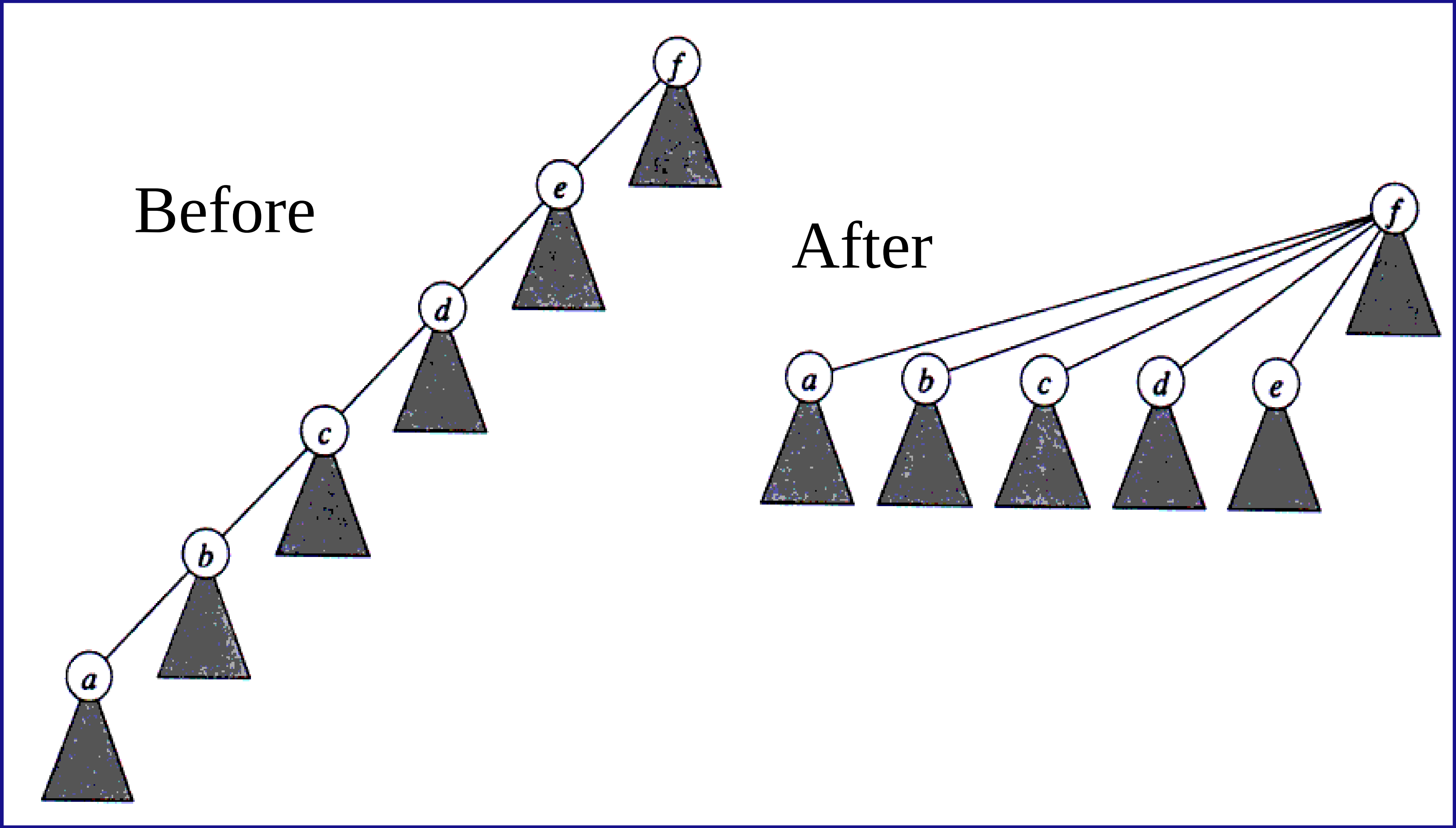
Tree representation

- For a sequence of $m>n$ Make-Set, Union, and Find-Set operations, of which n are Make-Set, the total time is $O(m \lg n)$.

Path compression

- Speed up Union-Find operations by shortening the sub-tree paths to the root.
- During a Find-Set operation, make each node on the find path point directly to the root.
- worst-case time complexity is $O(m\ \alpha(n))$ where $\alpha(n)$ is the *inverse Ackerman function*.
- The inverse Ackerman function grows so slowly that for all practical purposes $\alpha(n) \leq 4$ for very large n.

Example: path compression



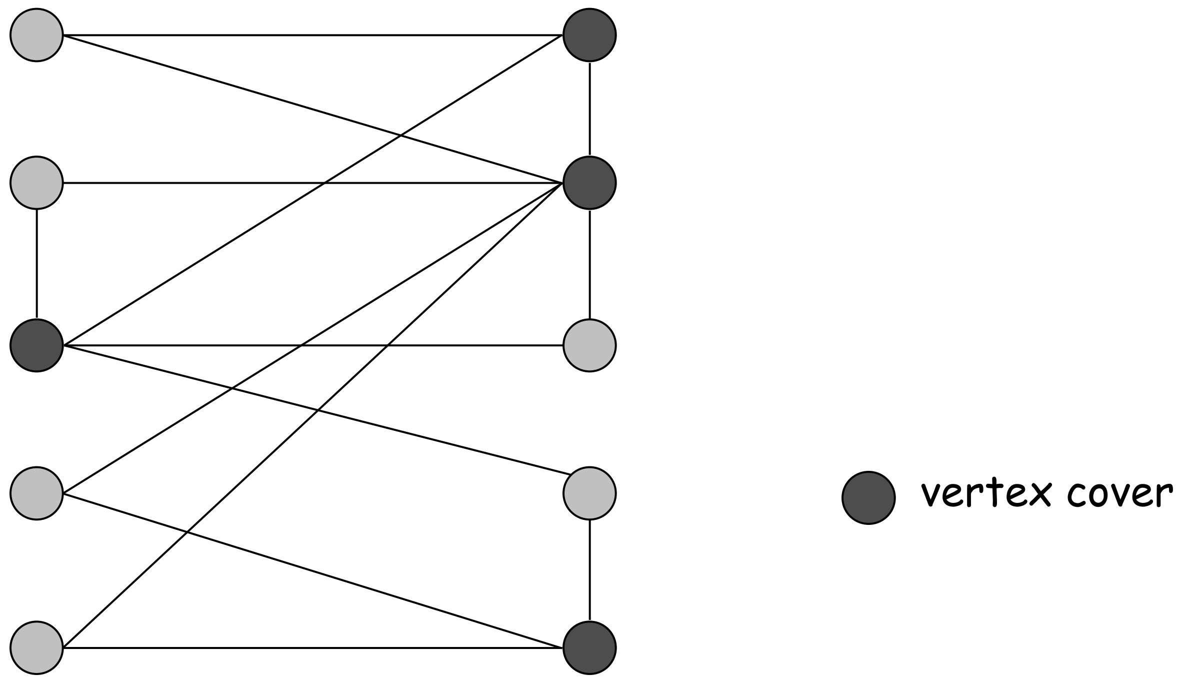
Summary

- Union-Find has many applications.
- For a sequence of $m > n$ Make-Set, Union, and Find-Set operations, of which n are Make-Set:
  - <u>List implementation</u>: $O(m + n \lg n)$ with weighted union heuristic.
  - <u>Tree implementation</u>: union by rank + path compression yields $O(m\,\alpha(n))$ complexity.

Vertex Cover

VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.
Ex. Is there a vertex cover of size ≤ 3? No.



Vertex Cover

- Considerthe Greedy Algorithm that always selects the node with minimum degree?
  - Does it always produce the optimal solution? Why?
  - How can we implement it efficiently?