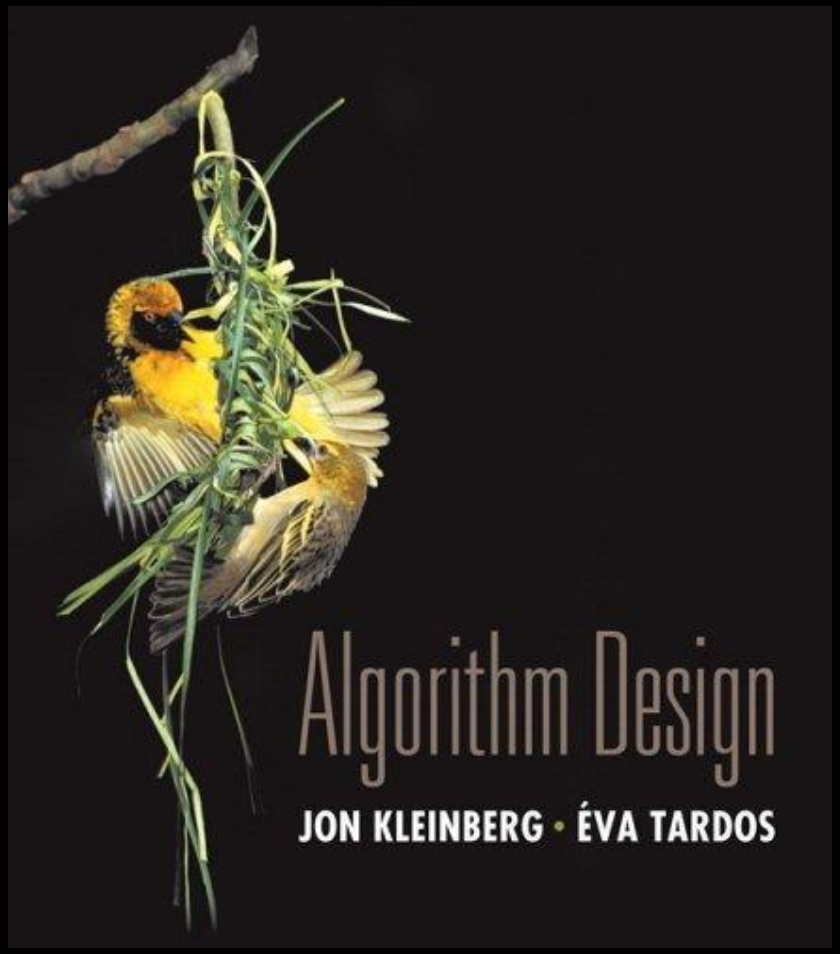


Chapter 4

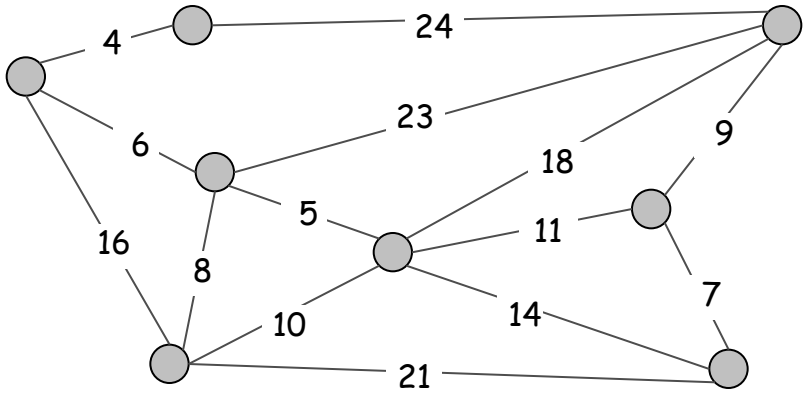
Greedy Algorithms

4.5 Minimum Spanning Tree

Minimum Spanning Tree

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge **positive** weights c_e, find a subset $E' \subseteq E$ such that

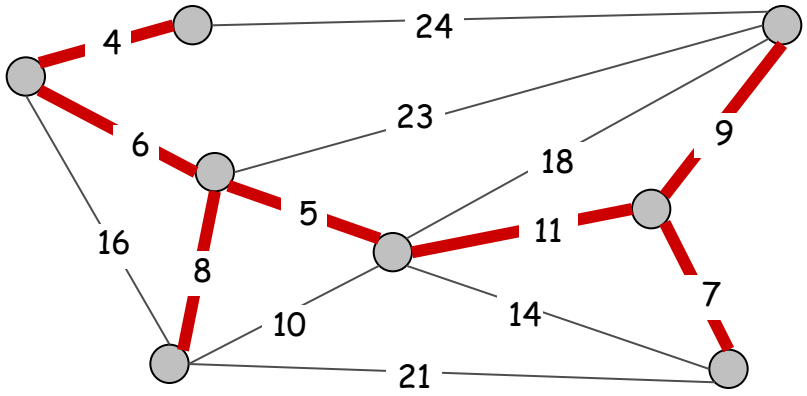
(i) the graph $G'=(V,E')$ is connected
(ii) smallest possible cost



Minimum Spanning Tree

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge **positive** weights c_e, find a subset $E' \subseteq E$ such that

(i) the graph $G'=(V,E')$ is connected
(ii) smallest possible cost



Key Observation. The optimal solution does not contain cycles => it is a tree

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.

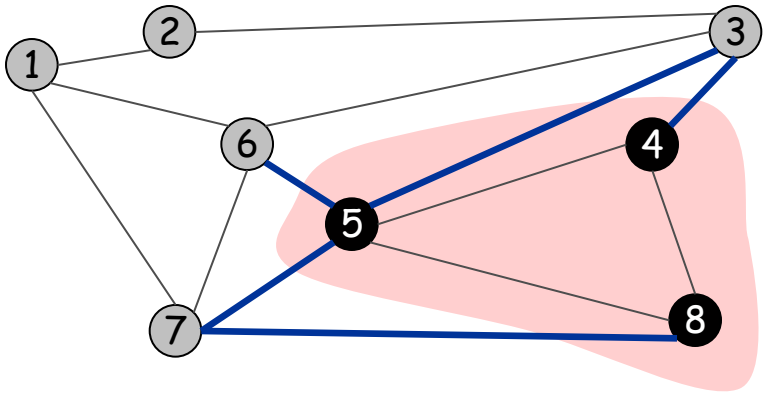
Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

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.

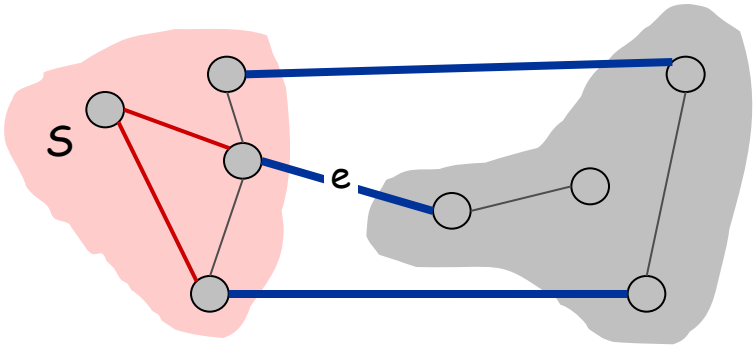


Cut S = { 4, 5, 8 }
Crossing edges = 5-6, 5-7, 3-4, 3-5, 7-8

- An edge **e** crosses a cut S if **e** has an edpoint in S and the other one in V-S

Greedy Algorithms

Cut property. Suppose set of edges X belongs to a MST for G. Pick a cut S containing all edges in X. If **e** is the lightest edge that crosses S then $X \cup e$ also belongs to a MST



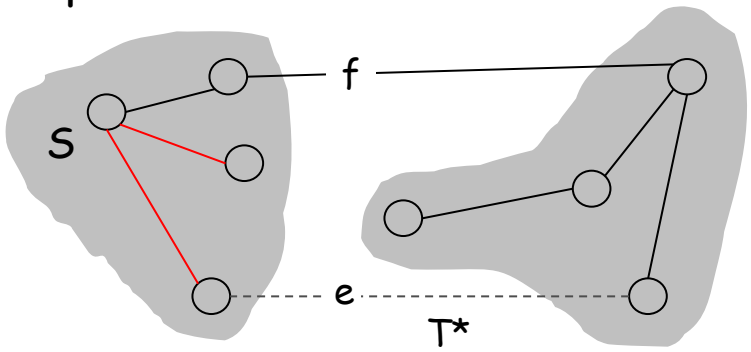
X = red edges

Greedy Algorithms

Cut property. Suppose set of edges X belongs to a MST for G. Pick a cut S containing all edges in X. If **e** is the lightest edge that crosses S then $X \cup e$ also belongs to a MST

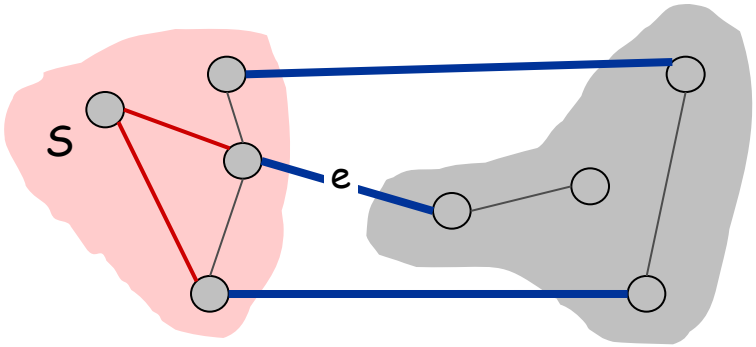
Pf. (exchange argument)

- Consider an MST T* containing X
- Suppose lightest edge **e** does not belong to T* => there is another edge **f** connecting cut to outside of the cut
- Adding **e** to T* creates a cycle containing **f**
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e <= c_f$, the new tree T' has cost at most cost of T*
- Then new tree T' is also a MST and $X \cup e$ is part of it ▪



Greedy Algorithms

Cut property. Suppose set of edges X belongs to a MST for G. Pick a cut S containing all edges in X. If **e** is the lightest edge that crosses S then $X \cup e$ also belongs to a MST



X = red edges

Greedy Algorithms

Cut property.

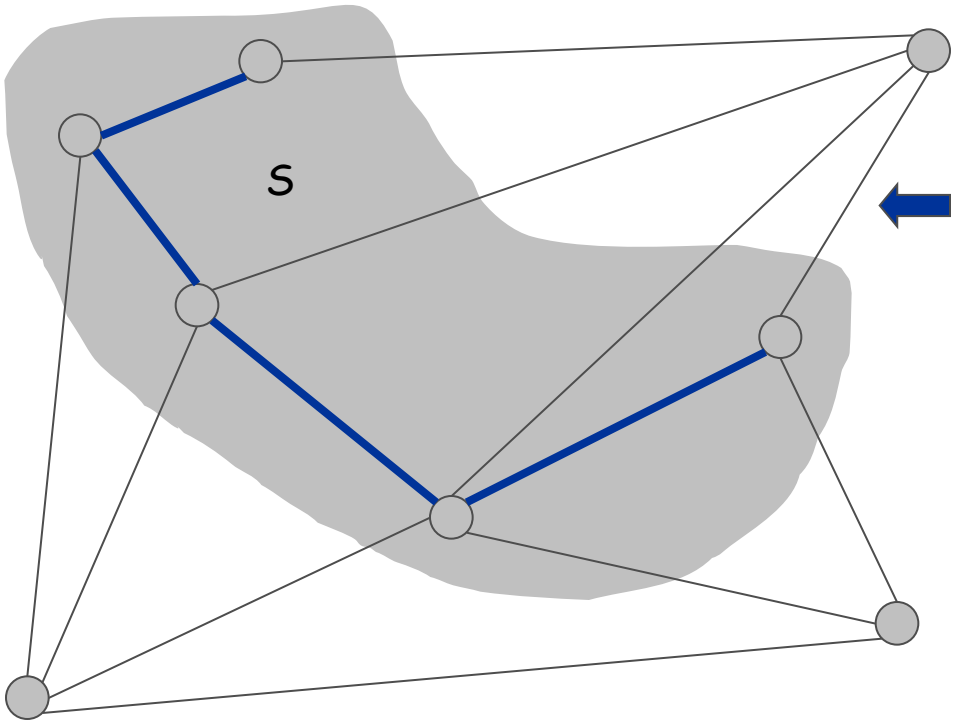
- Different applications of Cut property lead to different algorithms for constructing Minimal Spanning Trees.

- Prim and Kruskal algorithm construct a MST applying the Cut property n-1 times.

Prim's Algorithm

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

- Initialize S = any node, tree T = empty
- 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.

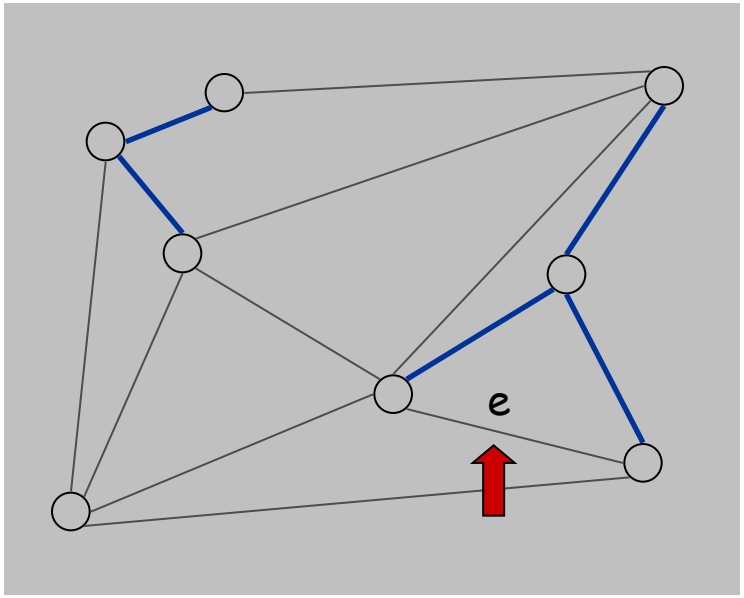
```
Prim(G, c) {
    foreach (v ∈ V) a[v] ← ∞
    Initialize an empty priority queue Q
    foreach (v ∈ V) insert v onto Q
    Initialize set of explored nodes S ← ϕ

    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
}
```

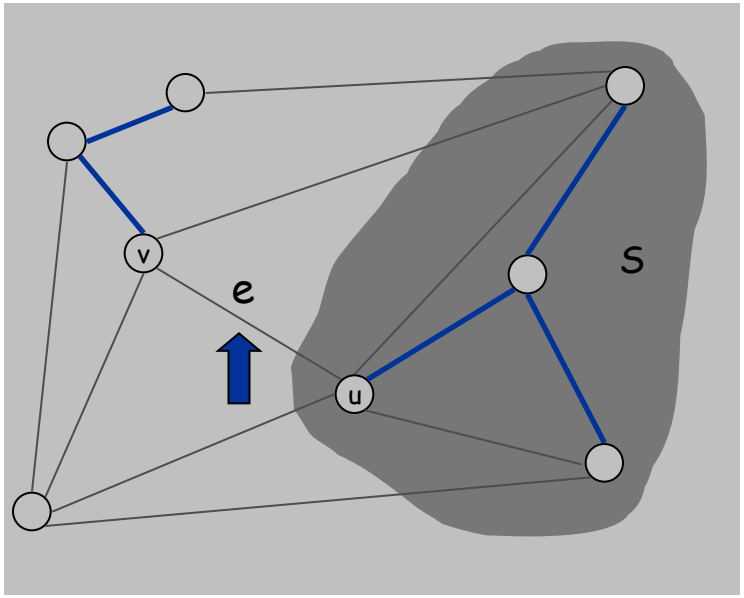
Kruskal's Algorithm

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

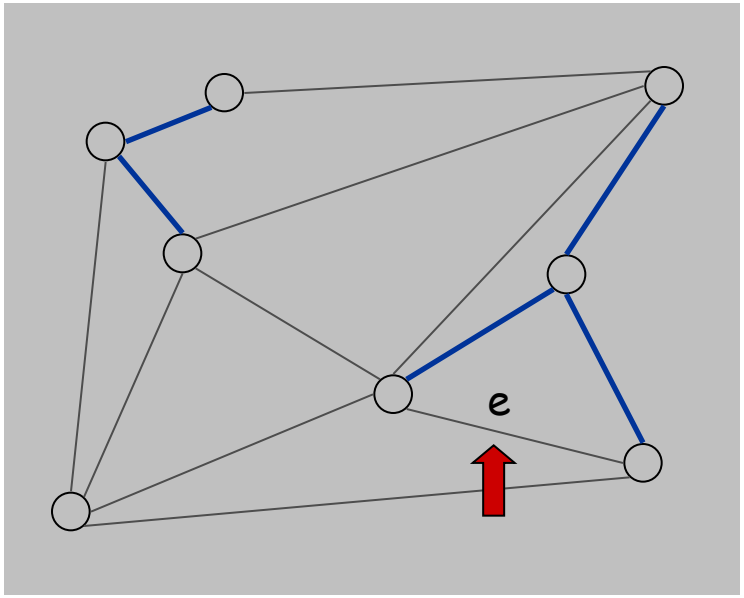


Case 2

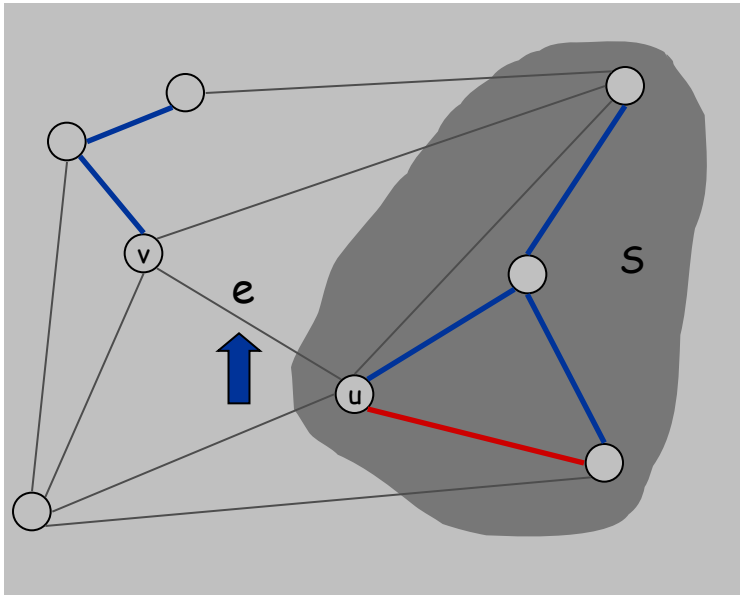
Kruskal's Algorithm: Proof of correctness

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 $O(m \, \alpha(m, n))$ for union-find.

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

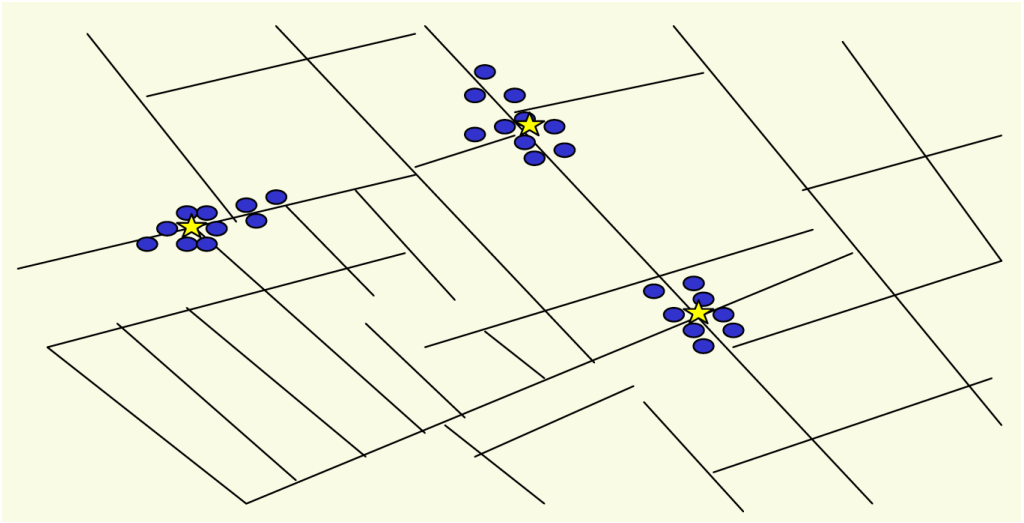
essentially a constant

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

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

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

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.

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.

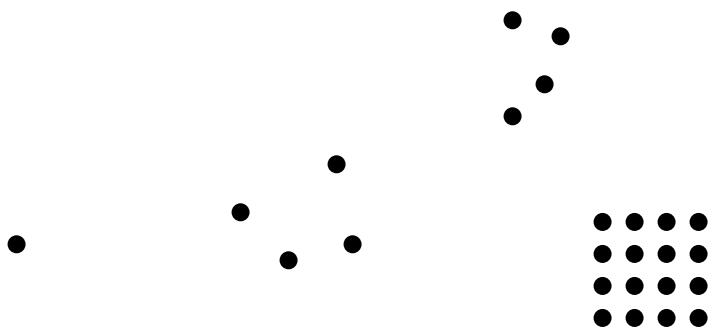
Clustering

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.

number of corresponding pixels whose intensities differ by some threshold

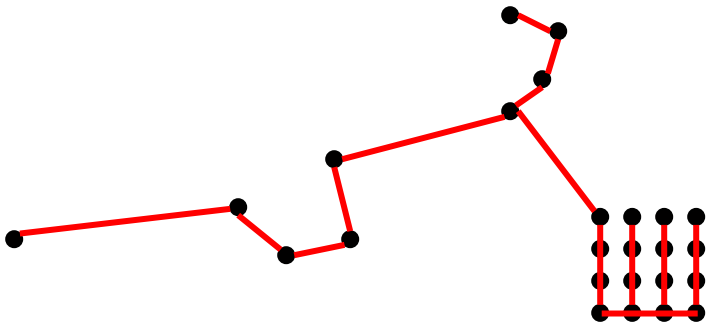


Clustering of Maximum Spacing

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

Q: Can we use an MST to perform k-clustering?

A: Start with MST, keep removing heaviest edge until we get k connected components



k = 4

Clustering of Maximum Spacing

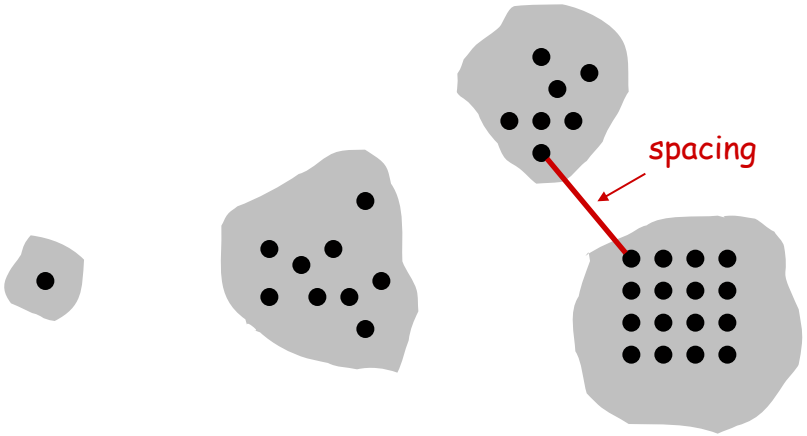
k-clustering. Divide objects into k non-empty groups.

Q: Can we use an MST to perform k-clustering?

A: Start with MST, keep removing heaviest edge until we get k connected components

Guarantees:

1. This algorithm gives cheapest way of forming k connected components (generalizes MST, which gives cheapest 1 conn. comp)
2. Maximizes spacing: minimum space between different classes



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]

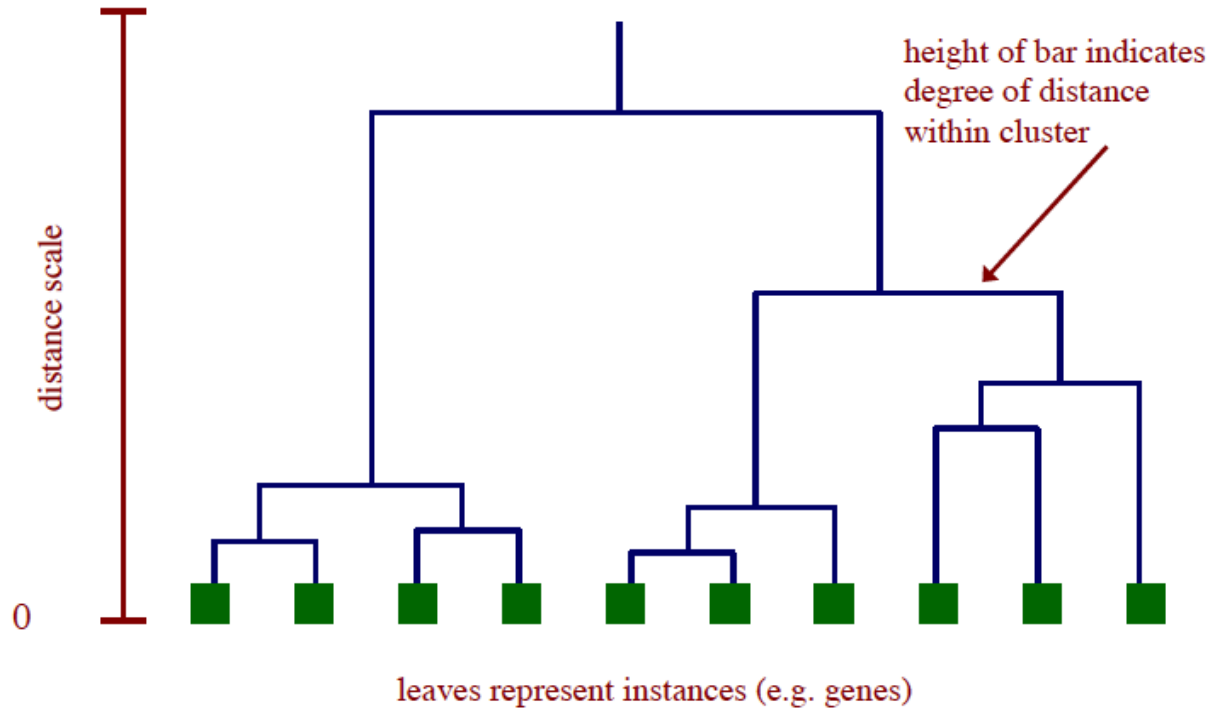
Euclidean.

- 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.

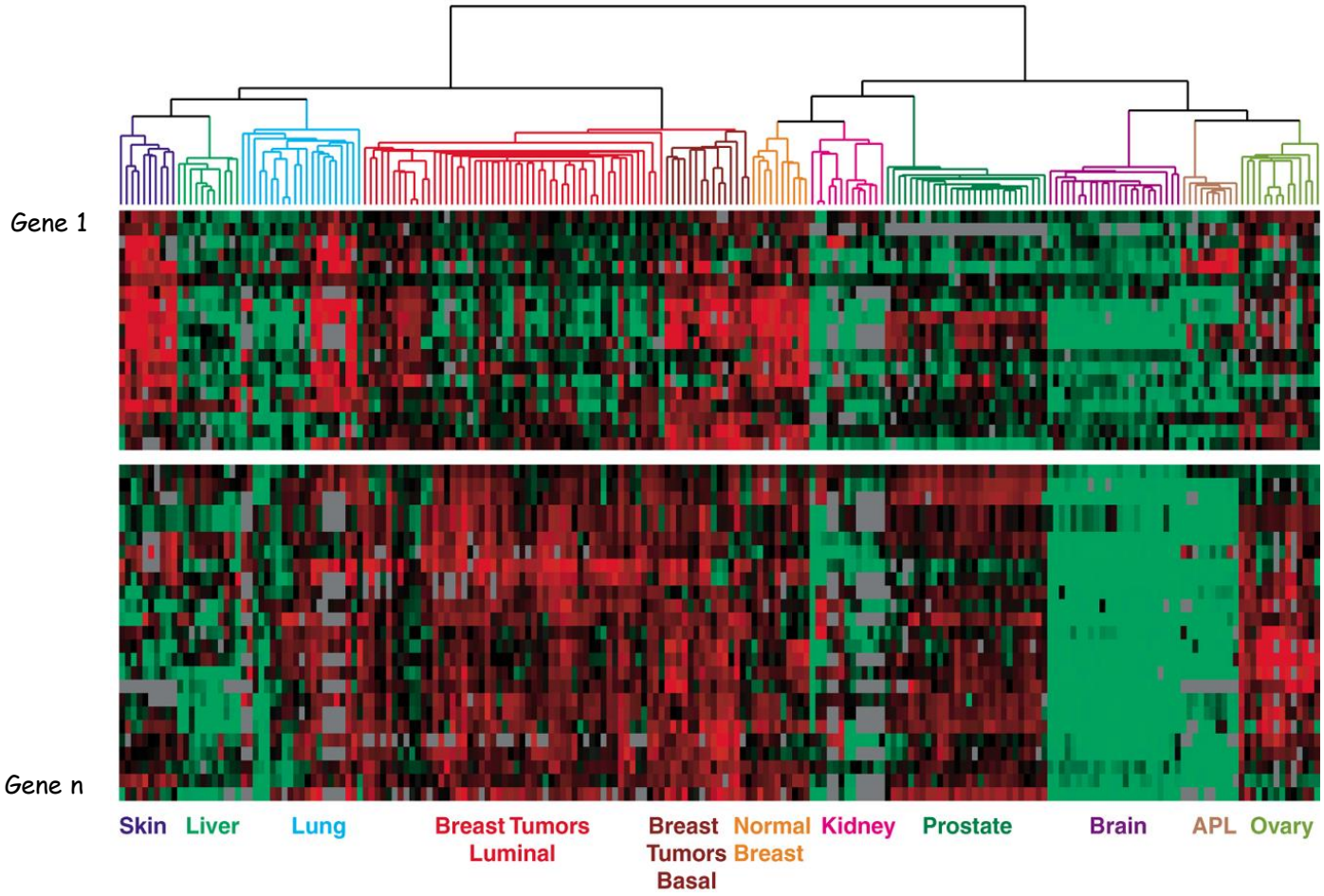
- Leaves = genes.
- Internal nodes = hypothetical ancestors.



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

Union-Find on disjoint sets

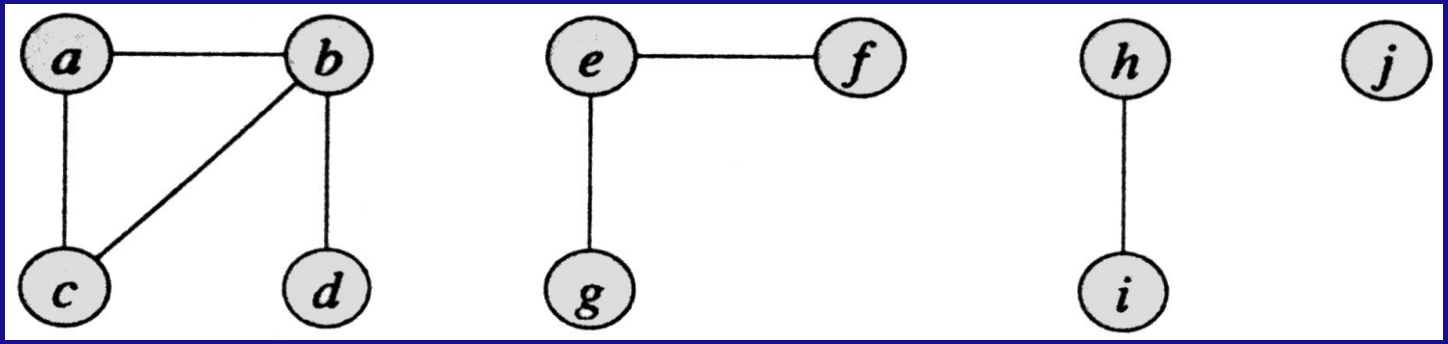
Motivation

- 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* **S** = {{*a*}, {*b*}, {*c*}, {*d*}, {*e*}, {*f*}, {*g*}, {*h*}, {*i*}, {*j*}}
- (*b*,*d*) **S** = {{*a*}, {*b*,*d*}, {*c*}, {*e*}, {*f*}, {*g*}, {*h*}, {*i*}, {*j*}}
- (*e*,*g*) **S** = {{*a*}, {*b*,*d*}, {*c*}, {*e*,*g*}, {*f*}, {*h*}, {*i*}, {*j*}}
- (*a*,*c*) **S** = {{*a*,*c*}, {*b*,*d*}, {*e*,*g*}, {*f*}, {*h*}, {*i*}, {*j*}}
- (*h*,*i*) **S** = {{*a*,*c*}, {*b*,*d*}, {*e*,*g*}, {*f*}, {*h*,*i*}, {*j*}}
- (*a*,*b*) **S** = {{*a*,*c*,*b*,*d*}, {*e*,*g*}, {*f*}, {*h*,*i*}, {*j*}}
- (*e*,*f*) **S** = {{*a*,*c*,*b*,*d*}, {*e*,*f*,*g*}, {*h*,*i*}, {*j*}}
- (*b*,*c*) **S** = {{*a*,*c*,*b*,*d*}, {*e*,*f*,*g*}, {*h*,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.

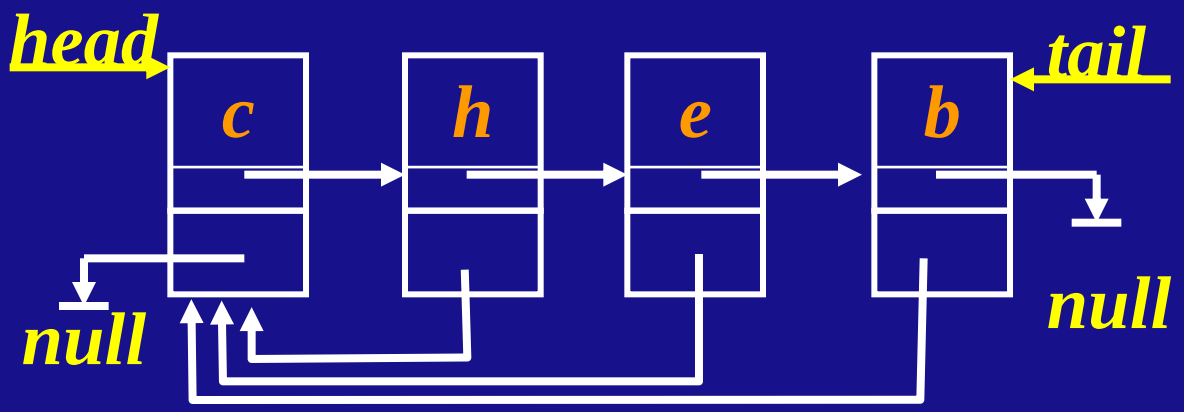
  Find-Set(x): find and return the representative of the set S_x that contains x.

Disjoint sets: linked list representation

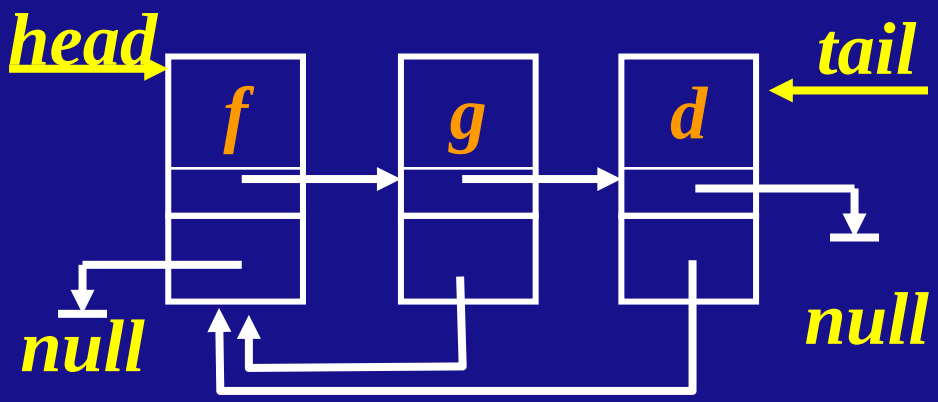
- Each set is a linked list, and the representative is the head of the list. Elements point to the successor and to the head of the list.
- Make-Set: create a new list: $O(1)$.
- Find-Set: search for an element down the list: $O(1)$.
- Union: link the tail of L_1 to the head of L_2, and make each element 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

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

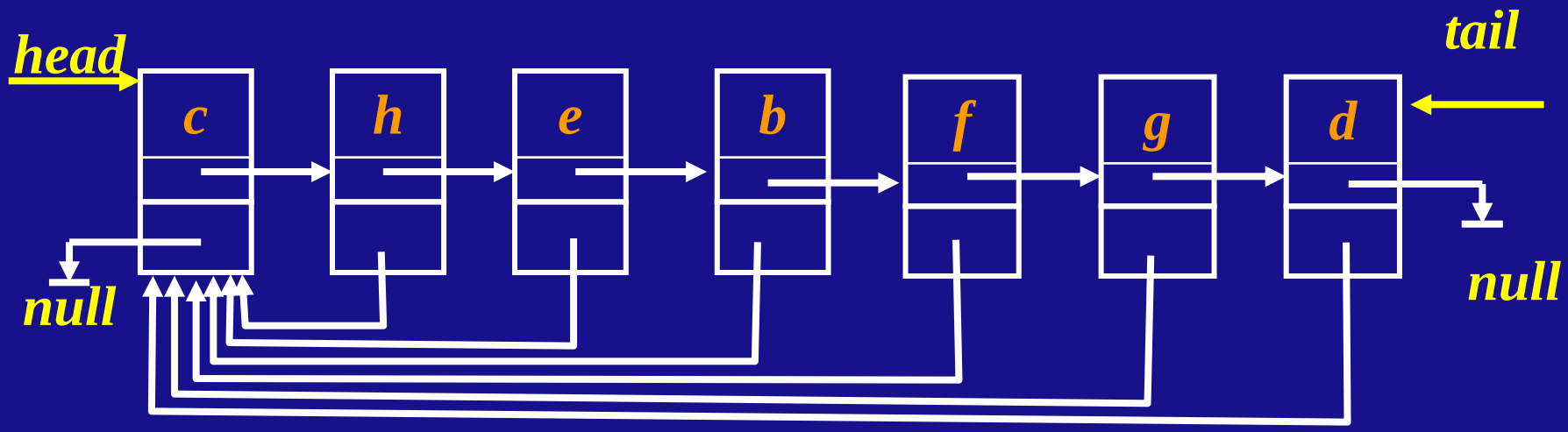


$S_2=\{f,g,d\}$



Example: linked list representation

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



Weighted-union heuristic

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

- A single operation may take $O(n)$ time.
  - Two lists of n/2 elements
  - Still $O(n^2)$ time for n Unions

Can we have n Unions of cost cn^2?

Weighted-union heuristic

A={a};B={b},C={c,d,e},D={x,f,g}, E={h,i,j}

	a	b	c	d	e	x	f	g	h	i	j	custo

Weighted-union heuristic

A={a};B={b},C={c,d,e},D={x,f,g}, E={h,i,j}

	a	b	c	d	e	x	f	g	h	i	j	custo
U(a,b)	1											1

AB={a,b}; C={c,d,e},D={x,f,g}, E={h,i,j}

Weighted-union heuristic

A={a};B={b},C={c,d,e},D={x,f,g}, E={h,i,j}

	a	b	c	d	e	x	f	g	h	i	j	custo
U(a,d)	1											1
U(b,d)	1	1										2

ABC={a,b,c,d,e},D={x,f,g}, E={h,i,j}

Weighted-union heuristic

A={a};B={b},C={c,d,e},D={x,f,g}, E={h,i,j}

	a	b	c	d	e	x	f	g	h	i	j	custo
U(a,d)	1											1
U(b,d)	1	1										2
U(x,h)						1	1	1				3

ABC={a,b,c,d,e},DE={x,f,g,h,i,j}

Weighted-union heuristic

A={a};B={b},C={c,d,e},D={x,f,g}, E={h,i,j}

	a	b	c	d	e	x	f	g	h	i	j	custo
U(a,d)	1											1
U(b,d)	1	1										2
U(x,h)						1	1	1				3
U(e,g)	1	1	1	1	1							5

ABCDE={a,b,c,d,e,x,f,g,h,i,j}

Weighted-union heuristic

A={a};B={b},C={c,d,e},D={x,f,g}, E={h,i,j}

	a	b	c	d	e	x	f	g	h	i	j	custo
U(a,d)	1											1
U(b,d)	1	1										2
U(x,h)						1	1	1				3
U(e,g)	1	1	1	1	1							5
Total	**3**	**2**	**1**	**1**	**1**	**1**	**1**	**1**				**11**

ABCDE={a,b,c,d,e,x,f,g,h,i,j}

Weighted-union heuristic

Double Counting

- S = Sum of the costs of union operations
- C= Sum of the contribution of each element
  - We count the number of times its representative is updated

$$C = S$$

Weighted-union heuristic

Double Counting

- Whenever an object x has its pointer updated, the size of the list where x lies doubles.

- Thus, an object can have its pointer updated at most log n times

Weighted-union heuristic

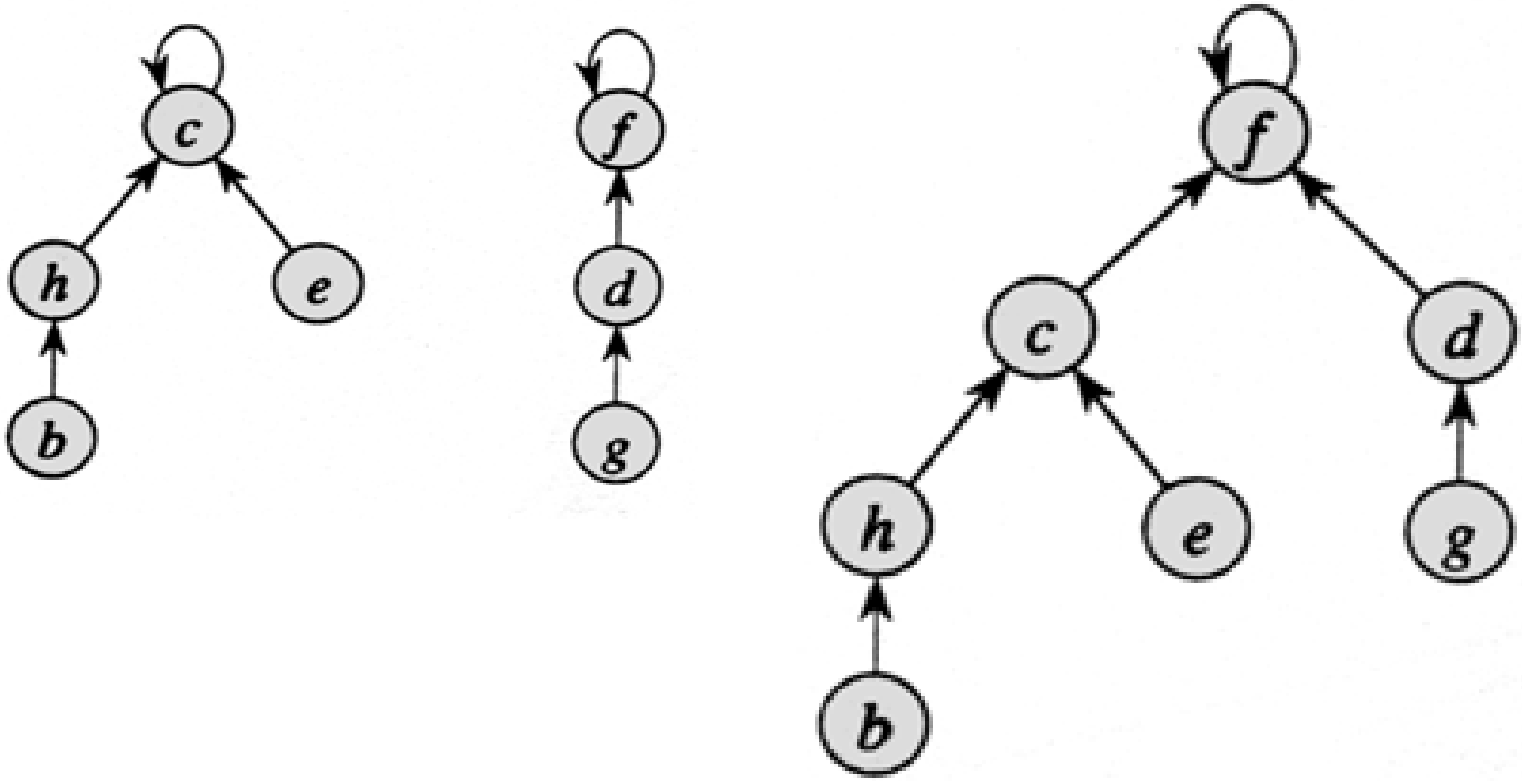
- For a sequence of $m > n$ Make-Set, Union, and Find-Set operations, of which n are Make-Set, the total time is $O(m + n\lg n)$ instead of $O(mn)$!
- Proof:
  - Each object has its pointer updated at most log n times
  - Total effort of unions is O(n log n)
  - Each find spends O(m)

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.

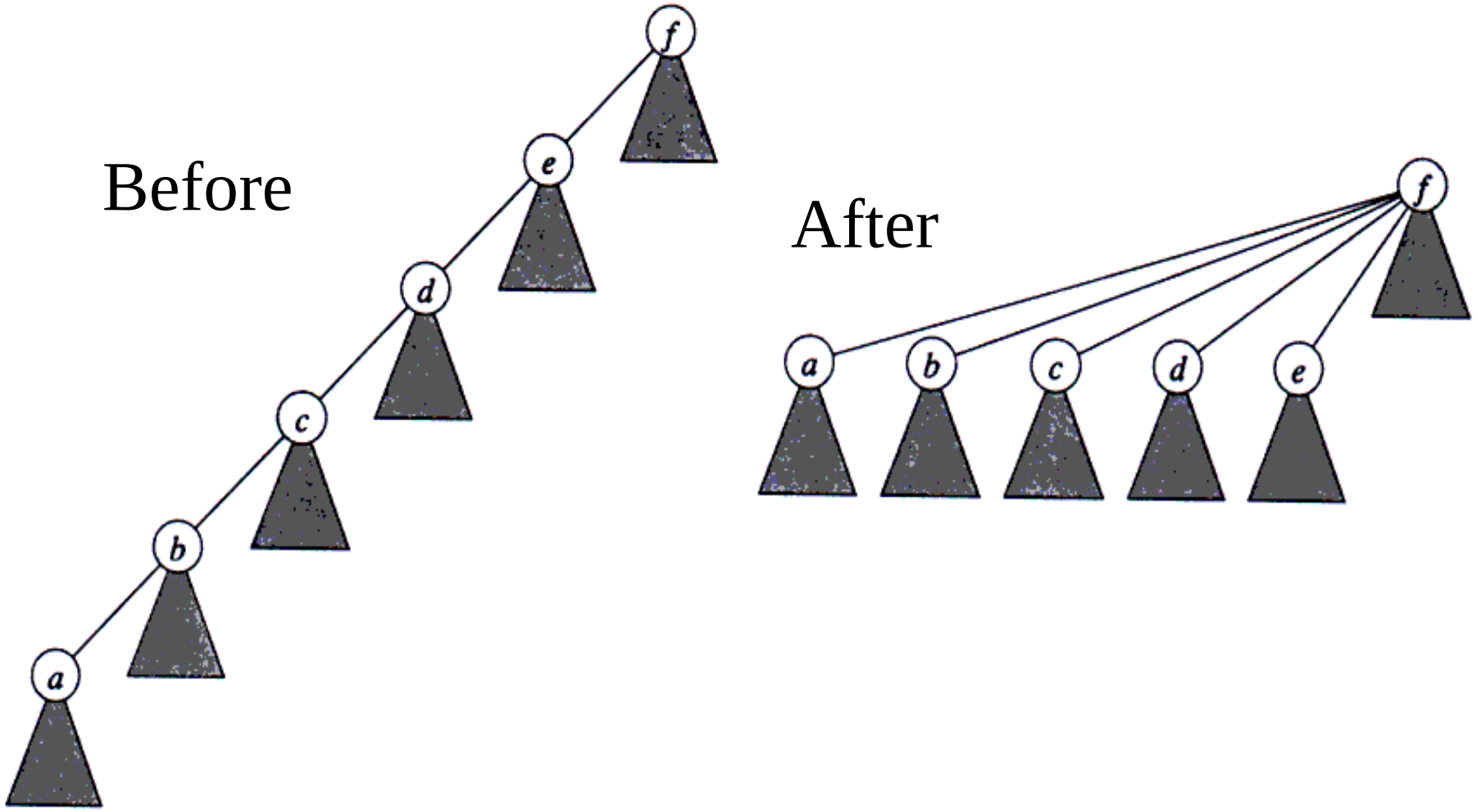
Tree representation

- 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

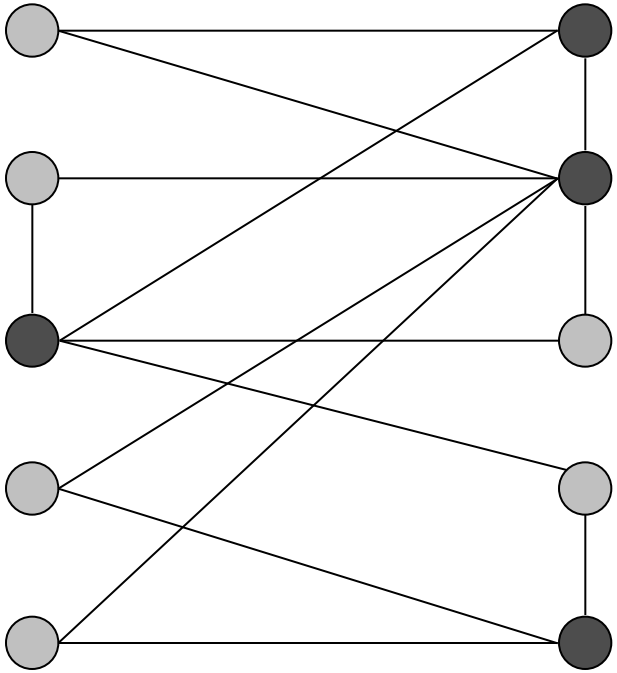
- Union-Find has many applications.
- For a sequence of $m > n$ Make-Set, Union, and Find-Set operations, of which n are Make-Set:
  - List implementation: $O(m + n\lg n)$ with weighted union heuristic.
  - Tree implementation: union by rank + path compression yields $O(m\,\alpha(n))$ complexity.

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

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?