The Disjoint Set ADT

Subhabrata Samajder

IIIT, Delhi

Winter Semester,
26^{th} May, 2023

The Problem

- Some applications involve grouping n distinct elements into a collection of disjoint sets.

- Important operations:
  - FIND: Finding which set a given element belongs to.
  - UNION: Unite two sets.

Outline

- Operations supported by a disjoint-set data structure.
- Look at a simple application of a disjoint-set data structure.
- Present a simple linked-list implementation for disjoint sets.
- Look at a more efficient representation using rooted trees.
  - The running time using the tree representation is linear for all practical purposes.
  - But it is theoretically superlinear.

Disjoint-set Data Structure

Disjoint-set Data Structure

- Maintains a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- Each set is identified by a representative, which is some member of the set.
- In some applications, it doesn't matter which member is used as the representative.
  - **Consistent:** If the representative of a dynamic set is asked twice without modifying the set between the requests, then both times the answer should be the same.
- In other applications, there may be a prespecified rule for choosing the representative.

  For Example: Choosing the smallest member in the set (assuming, that the elements can be ordered).

Supported Operations

- MAKE-SET(x): Creates a new set whose only member (and thus representative) is x.
  - Sets are disjoint $\Rightarrow$ x cannot be in any other set.
- UNION(x, y): $S_x \cup S_y$, where x and y are representatives of S_x and S_y, respectively.
  - Prior to the operation $S_x \cap S_y = \emptyset$.
  - The representative of the resulting set can be any member of $S_x \cup S_y$.z
  - Many implementations of UNION specifically choose the representative of either S_x or S_y as the new representative.
  - The sets in the collection $\mathcal{S}$ needs to be disjoint $\Rightarrow$ the sets S_x and S_y are "destroyed" by removing them from $\mathcal{S}$.
- FIND-SET(x): Returns a pointer to the representative of the (unique) set containing x.

Parameters

- Running times are analyzed in terms of the following two parameters:
  - n: # MAKE-SET operations.
  - m: # MAKE-SET operations + # UNION operations + # FIND-SET operations.

Parameters

- Running times are analyzed in terms of the following two parameters:
  - n: # MAKE-SET operations.
  - m: # MAKE-SET operations + # UNION operations + # FIND-SET operations.
- Sets are disjoint $\Rightarrow$ each UNION operation reduces the number of sets by one.
- $\therefore$ after at most $n-1$ UNION operations, only 1 set remains.

Parameters

- Running times are analyzed in terms of the following two parameters:
  - n: # MAKE-SET operations.
  - m: # MAKE-SET operations + # UNION operations + # FIND-SET operations.
- Sets are disjoint $\Rightarrow$ each UNION operation reduces the number of sets by one.
- $\therefore$ after at most $n - 1$ UNION operations, only 1 set remains.
- $\therefore$ # of UNION operations $\leq n - 1$.

Parameters

- Running times are analyzed in terms of the following two parameters:
  - n: # MAKE-SET operations.
  - m: # MAKE-SET operations + # UNION operations + # FIND-SET operations.
- Sets are disjoint $\Rightarrow$ each UNION operation reduces the number of sets by one.
- $\therefore$ after at most $n-1$ UNION operations, only 1 set remains.
- $\therefore$ # of UNION operations $\leq n-1$.
- **Note** Since the MAKE-SET operations are included in the total number of operations m, we have $m \geq n$.

Parameters

- Running times are analyzed in terms of the following two parameters:
  - n: # Make-Set operations.
  - m: # Make-Set operations + # Union operations + # Find-Set operations.
- Sets are disjoint $\Rightarrow$ each Union operation reduces the number of sets by one.
- $\therefore$ after at most $n-1$ Union operations, only 1 set remains.
- $\therefore$ # of Union operations $\leq n-1$.
- **Note** Since the Make-Set operations are included in the total number of operations m, we have $m \geq n$.
- **Assumption:** n Make-Set operations are the first n operations performed.

A Simple Application of Disjoint-set Data Structure

Connected Components of an Undirected Graph

- CONNECTED-COMPONENTS(G): Run as a preprocessing step, creates connected components of the graphs G.
- SAME-COMPONENT(u, v): Answers queries about whether two vertices are in the same connected component.

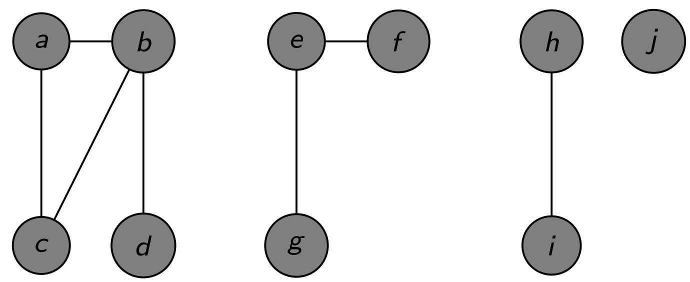
Connected Components of an Undirected Graph

- CONNECTED-COMPONENTS(G): Run as a preprocessing step, creates connected components of the graphs G.
- SAME-COMPONENT(u, v): Answers queries about whether two vertices are in the same connected component.

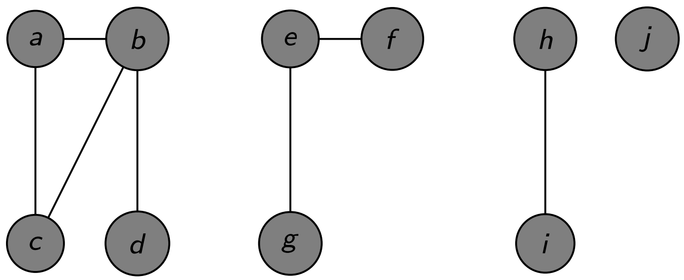
Note:

- When the edges of the graph are "static" (not changing over time) DFS computes the connected components faster.
- But sometimes the edges are added "dynamically".
- Maintain the connected components as each edge is added.
- Then implementation given here can be more efficient than running a new DFS search for each new edge added.

Connected Components: An Example



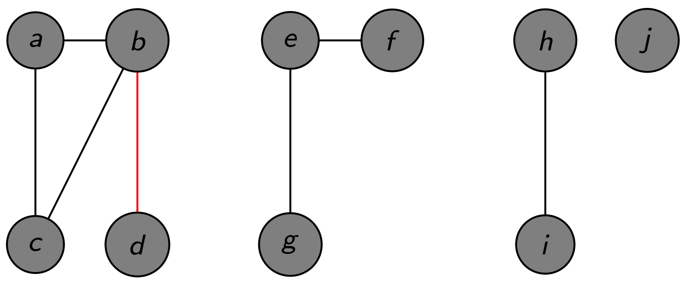
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);

Edge processed	Collection of disjoint sets									
Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$

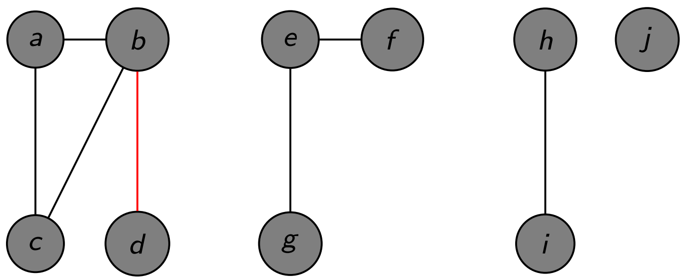
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$

Edge processed	Collection of disjoint sets									
Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b, d)	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$

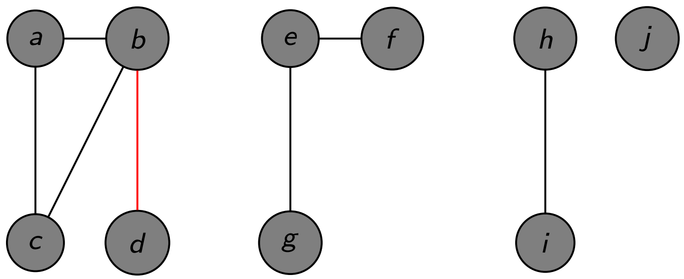
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))

Edge processed	Collection of disjoint sets									
Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b, d)	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$

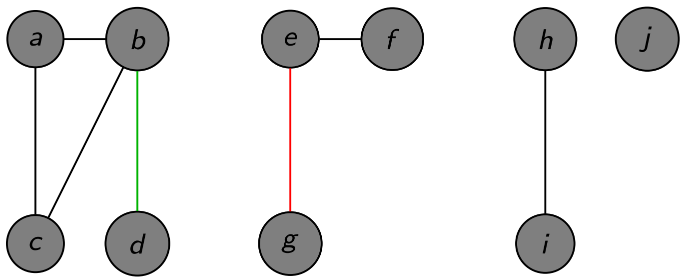
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets								
Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b, d)	$\{a\}$	$\{b, d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$

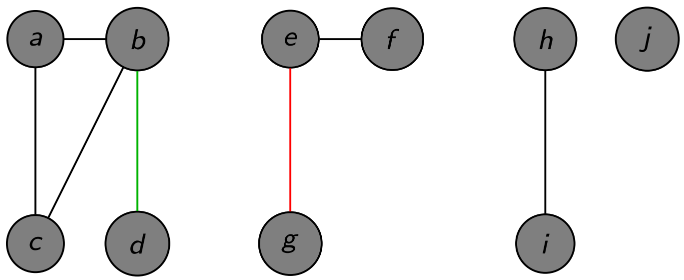
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET$(u) \neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets									
(b, d)	$\{a\}$	$\{b, d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e, g)	$\{a\}$	$\{b, d\}$	$\{c\}$		$\{b\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$

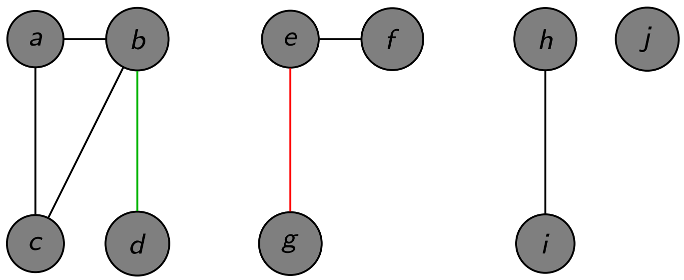
Connected Components: An Example



$\textsc{Connected-Components}(G)$
for each vertex $v \in V$
$\textsc{Make-Set}(v)$;
for each edge $(u, v) \in E$
if $(\textsc{Find-Set}(u) \neq \textsc{Find-Set}(v))$
$\textsc{Union}(u, v)$;

Edge processed	Collection of disjoint sets								
(b, d)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e, g)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$

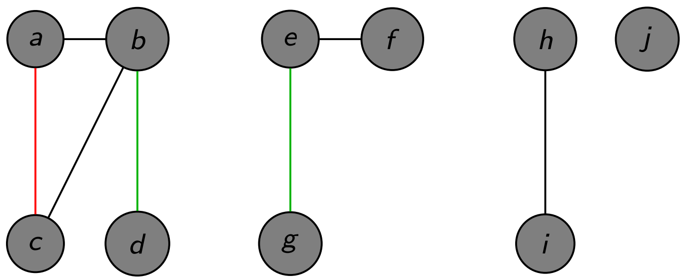
Connected Components: An Example



Connected-Components(G)
  for each vertex $v \in V$
    Make-Set(v);
  for each edge $(u, v) \in E$
    if (Find-Set(u) $\neq$ Find-Set(v))
      Union(u, v);

Edge processed	Collection of disjoint sets								
(b, d)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e, g)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e, g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$

Connected Components: An Example

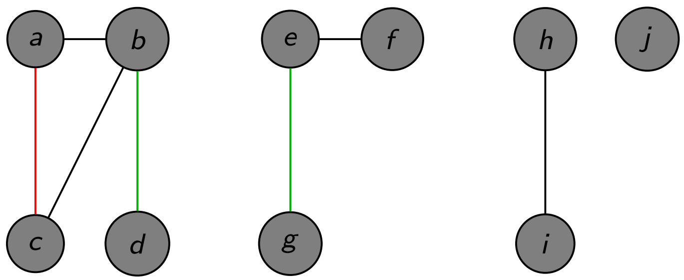


CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);

  for each edge $(u, v) \in E$
    if (FIND-SET$(u) \neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets								
(e, g)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$	
(a, c)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$	

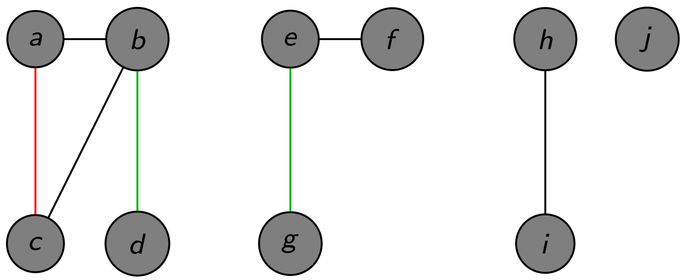
Connected Components: An Example



$\textsc{Connected-Components}(G)$
  for each vertex $v \in V$
    $\textsc{Make-Set}(v)$;
  for each edge $(u, v) \in E$
    if $(\textsc{Find-Set}(u) \neq \textsc{Find-Set}(v))$
      $\textsc{Union}(u, v)$;

Edge processed	Collection of disjoint sets							
(e, g)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$
(a, c)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$

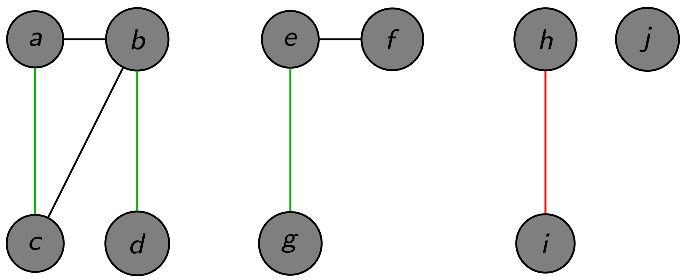
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets							
(e, g)	$\{a\}$	$\{b, d\}$	$\{c\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$
(a, c)	$\{a, c\}$	$\{b, d\}$		$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$

Connected Components: An Example

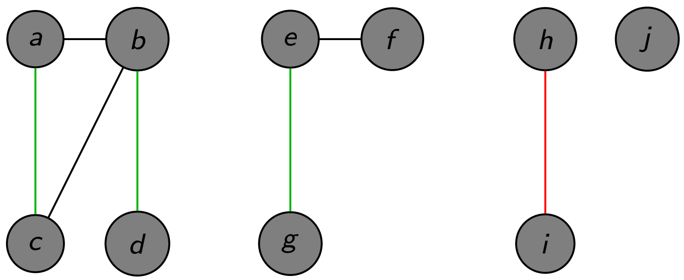


$\text{CONNECTED-COMPONENTS}(G)$
  for each vertex $v \in V$
    $\text{MAKE-SET}(v)$;

  for each edge $(u, v) \in E$
    if $(\text{FIND-SET}(u) \neq \text{FIND-SET}(v))$
      $\text{UNION}(u, v)$;

Edge processed	Collection of disjoint sets						
(a, c)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$
(h, i)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$

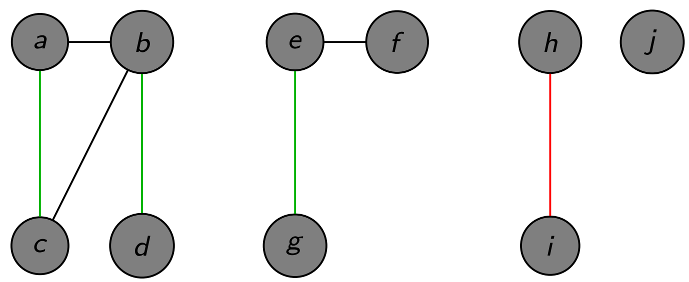
Connected Components: An Example



Connected-Components(G)
  for each vertex $v \in V$
    Make-Set(v);
  for each edge $(u, v) \in E$
    if (Find-Set(u) $\neq$ Find-Set(v))
      Union(u, v);

Edge processed	Collection of disjoint sets						
(a, c)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$
(h, i)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$

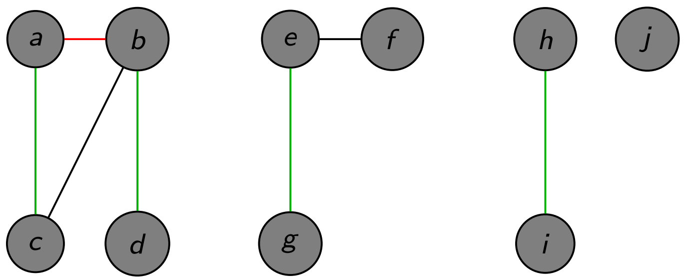
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets						
(a, c)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$
(h, i)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$		$\{j\}$

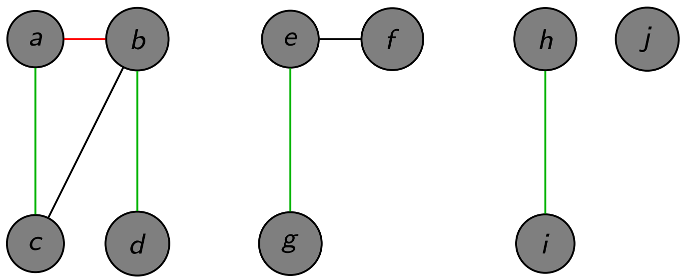
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets					
(h, i)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$
(a, b)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$

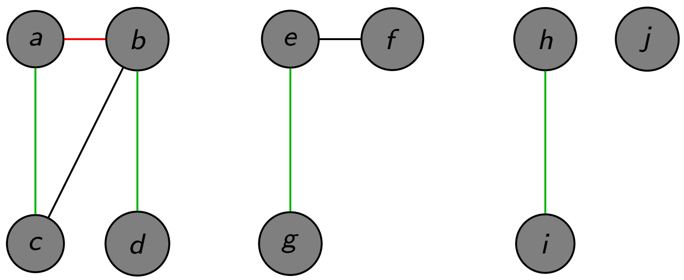
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET$(u) \neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets					
(h, i)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$
(a, b)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$

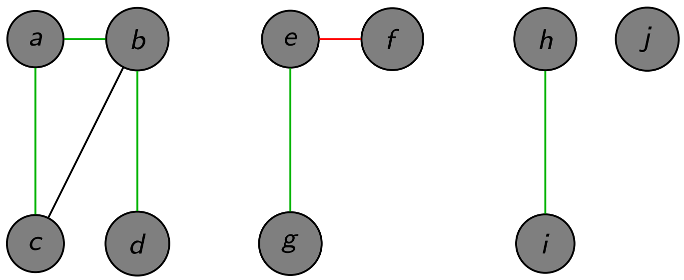
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets					
(h, i)	$\{a, c\}$	$\{b, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$
(a, b)	$\{a, b, c, d\}$		$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$

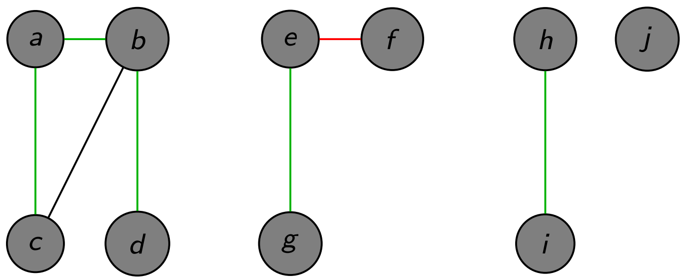
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets				
(a, b)	$\{a, b, c, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$
(e, f)	$\{a, b, c, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$

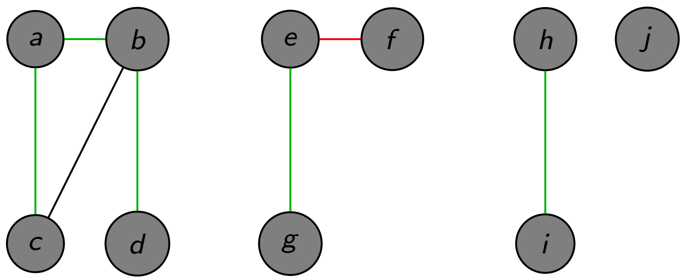
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets				
(a, b)	$\{a, b, c, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$
(e, f)	$\{a, b, c, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$

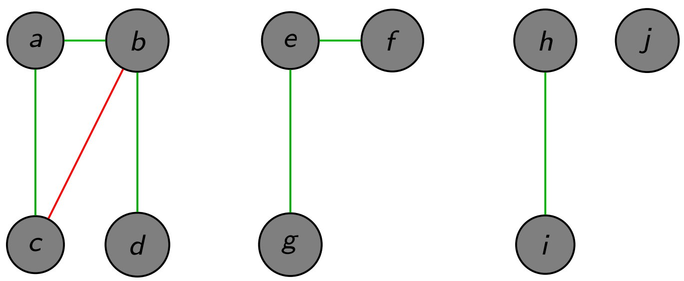
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets				
(a, b)	$\{a, b, c, d\}$	$\{e, g\}$	$\{f\}$	$\{h, i\}$	$\{j\}$
(e, f)	$\{a, b, c, d\}$	$\{e, f, g\}$		$\{h, i\}$	$\{j\}$

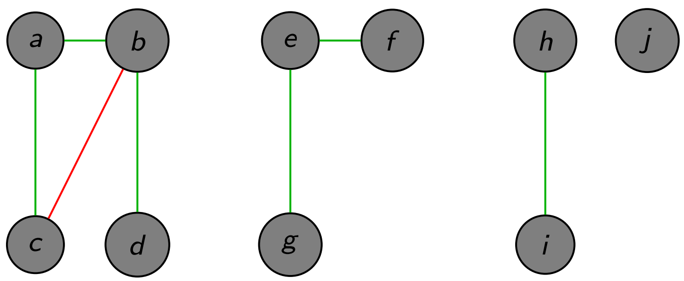
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets			
(e, f)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$
(b, c)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$

Connected Components: An Example

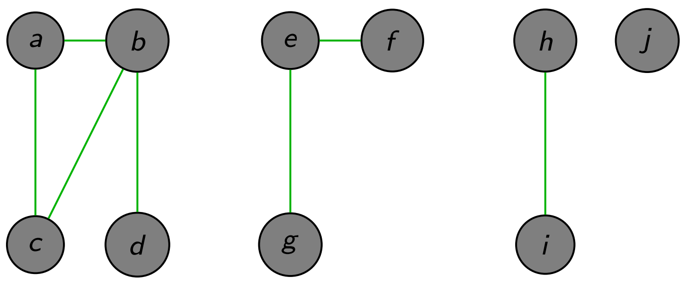


CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);

  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets			
(e, f)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$
(b, c)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$

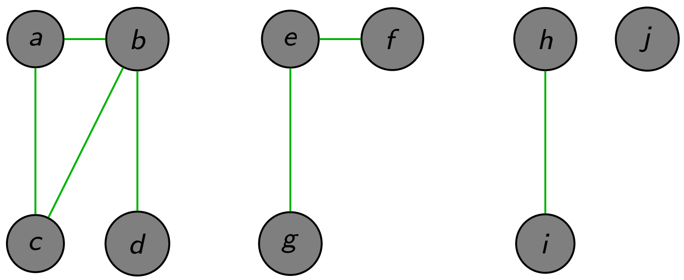
Connected Components: An Example



CONNECTED-COMPONENTS(G)
  for each vertex $v \in V$
    MAKE-SET(v);
  for each edge $(u, v) \in E$
    if (FIND-SET(u) $\neq$ FIND-SET(v))
      UNION(u, v);

Edge processed	Collection of disjoint sets			
(e, f)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$
(b, c)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$

Connected Components: An Example



$\textsc{Connected-Components}(G)$
  for each vertex $v \in V$
    $\textsc{Make-Set}(v)$;
  for each edge $(u, v) \in E$
    if $(\textsc{Find-Set}(u) \neq \textsc{Find-Set}(v))$
      $\textsc{Union}(u, v)$;

Edge processed	Collection of disjoint sets			
(b, c)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$

Same-Component

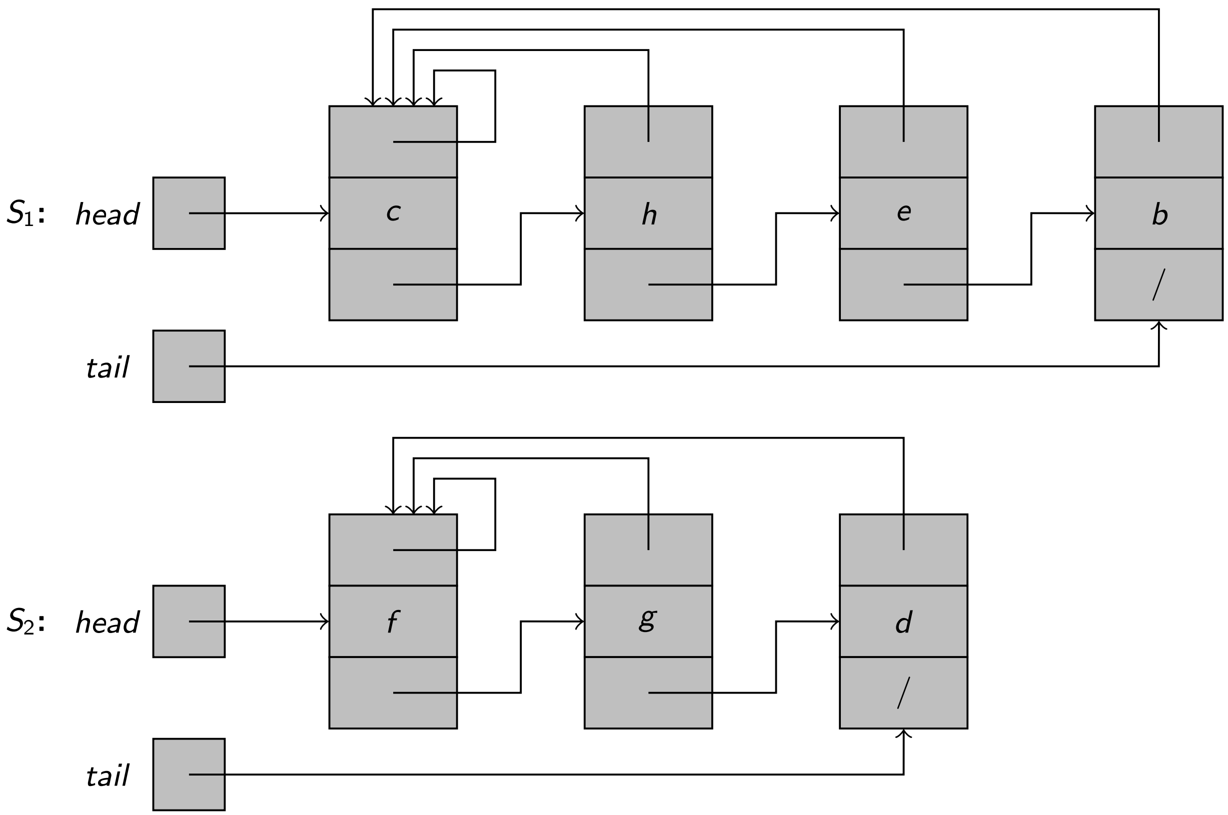
Same-Component(u, v)
  if (Find-Set(u) = Find-Set(v))
    return True;
  else
    return False;

Linked-list Representation of Disjoint Sets

Linked-list Representation

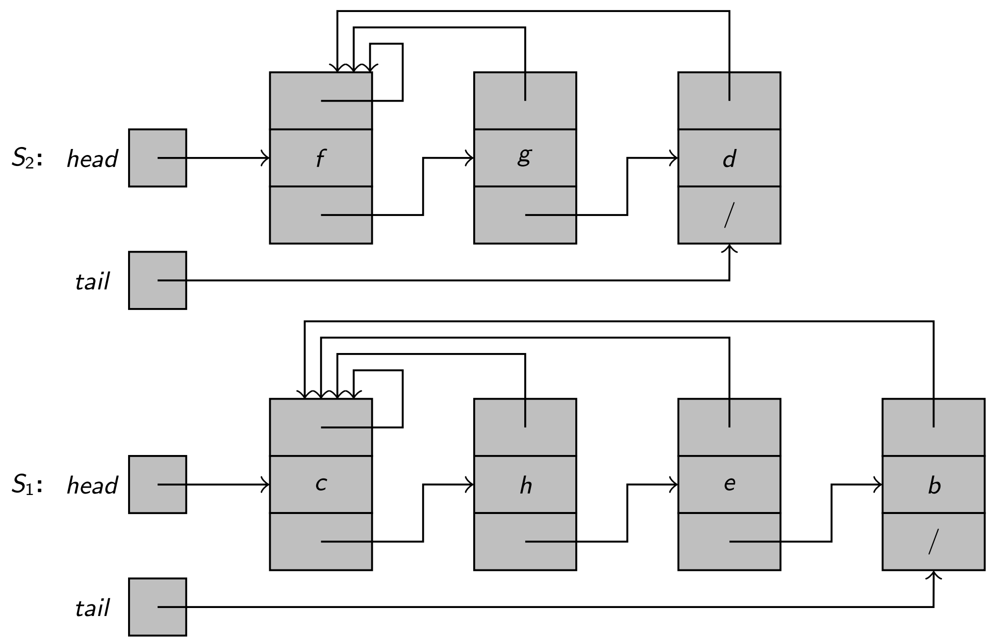
- A simple way to represent disjoint sets.
- Set representative: The first object in each linked list.
- Each object in the linked list contains
  - a set member,
  - a pointer to the object containing the next set member, and
  - a pointer back to the set representative.
- Each list maintains pointers
  - *head*: which points to the set representative, and
  - *tail*: which points to the last object in the list.
- Within each linked list, the remaining objects may appear in any order.

An Example

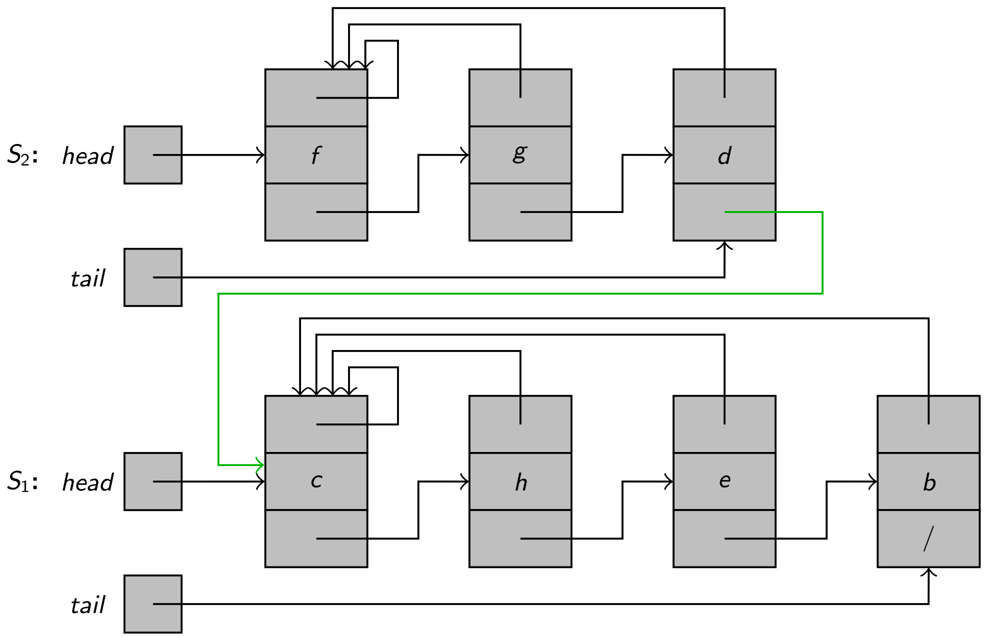


Disjoint sets $S_1 = \{c, h, e, b\}$ **and** $S_2 = \{f, g, d\}$

An Example: Union(e, g)

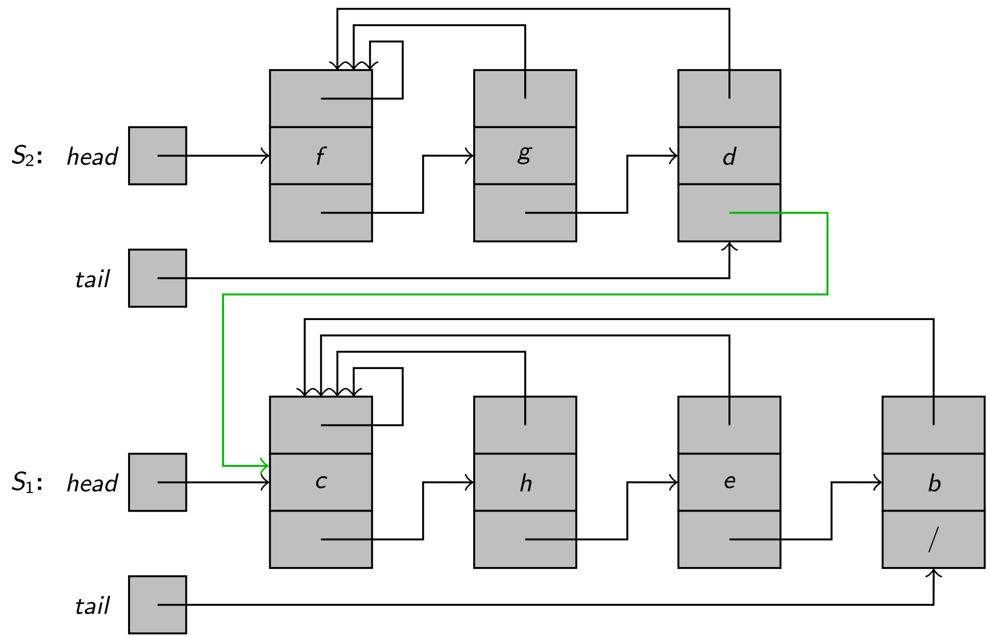


An Example: UNION(e, g)



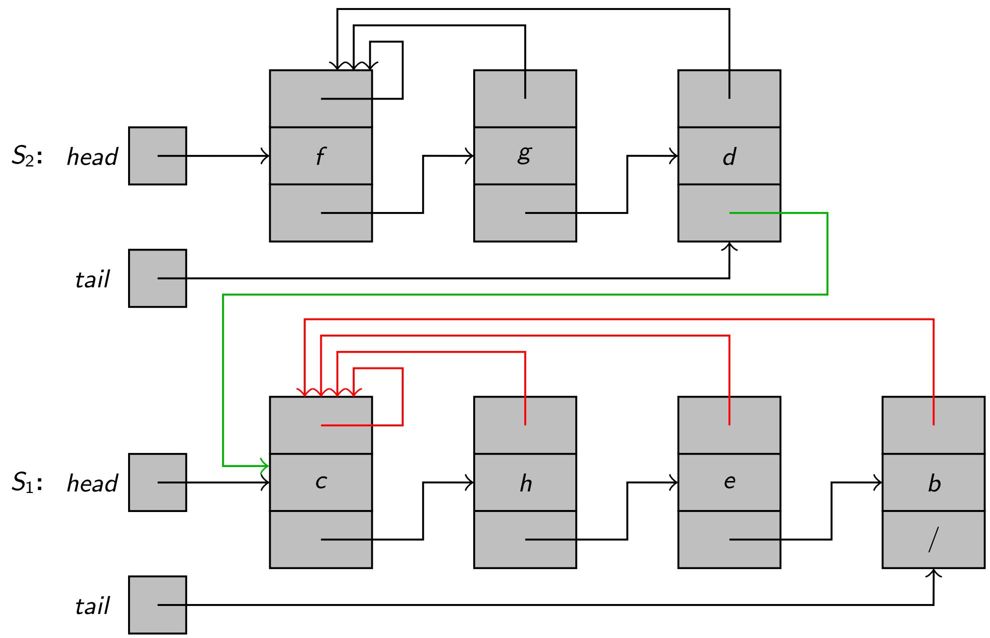
- Use the *tail* pointer for g's list to quickly find where to append e's list.

An Example: UNION(e, g)



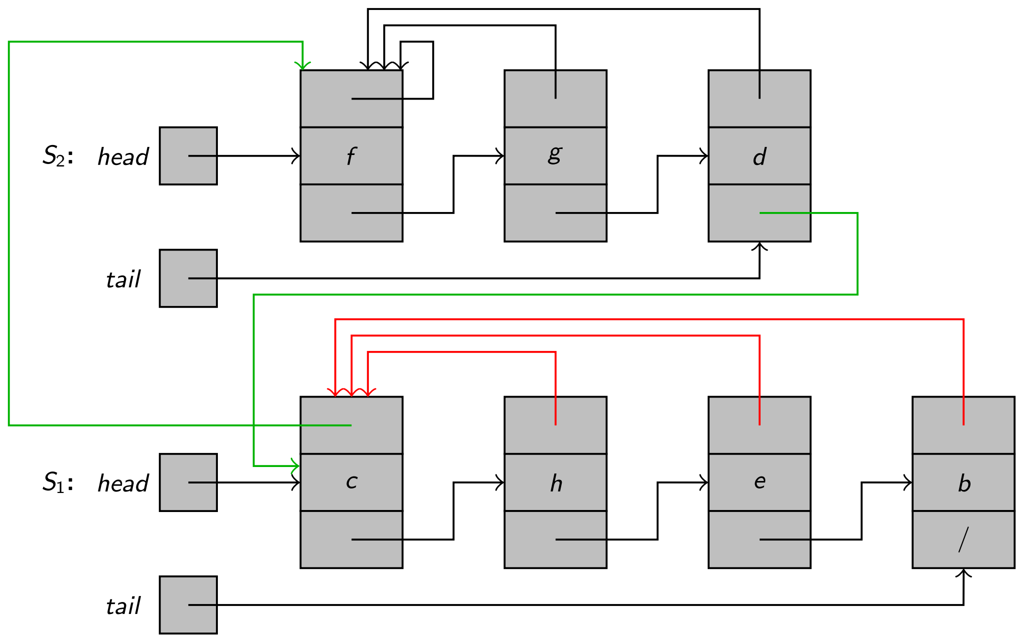
- Use the *tail* pointer for g's list to quickly find where to append e's list.
- **Note:** The representative of the new set = representative of the set containing g, i.e., f.

An Example: $\textsc{Union}(e, g)$



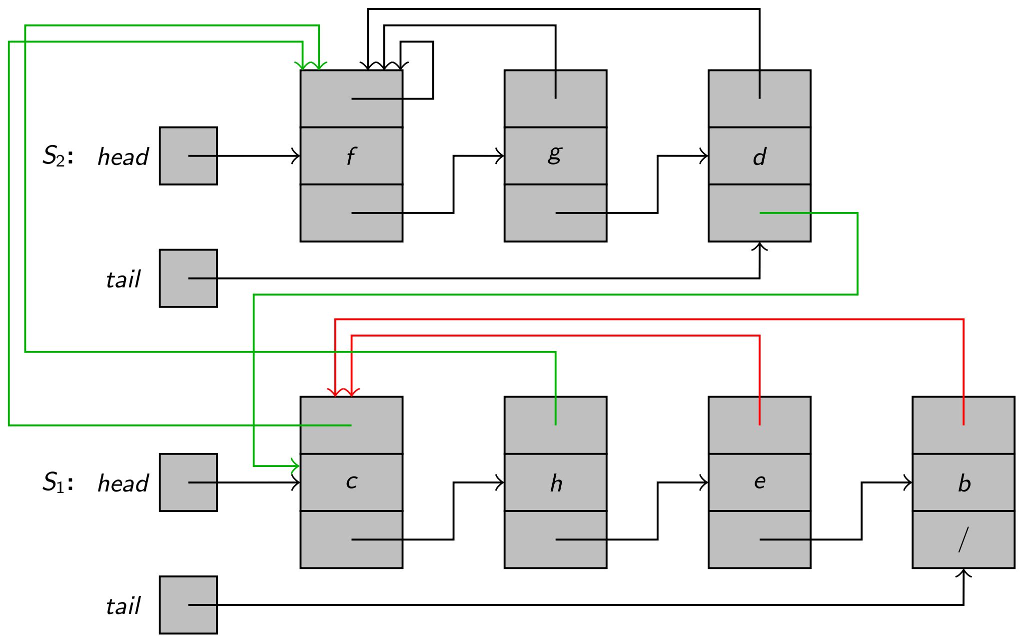
- However, we must update the pointer to the representative for each object originally on e's list.

An Example: UNION(e, g)



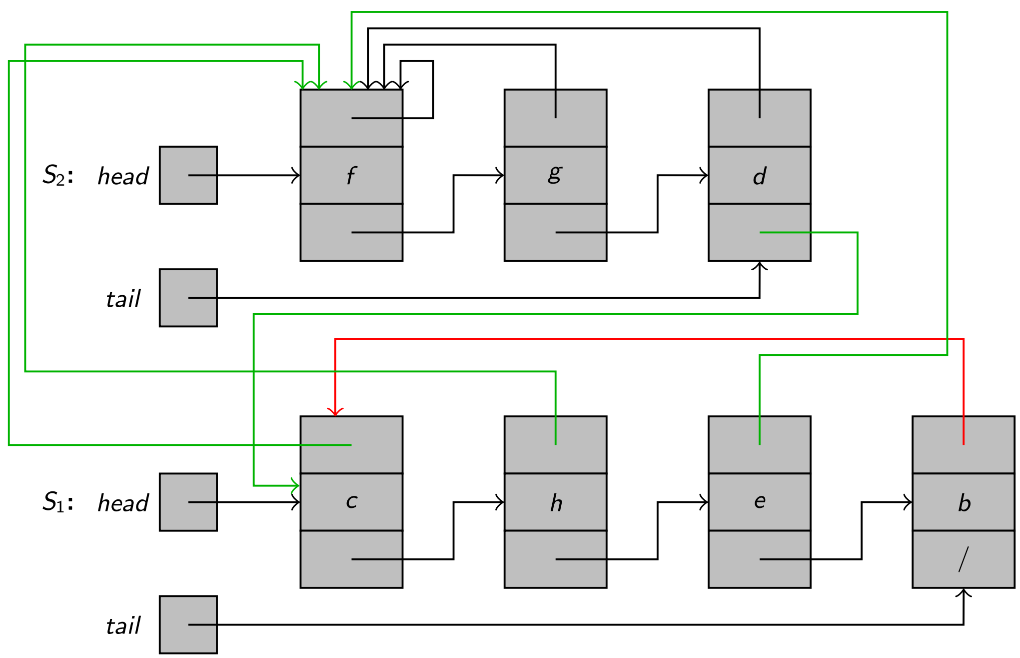
- However, we must update the pointer to the representative for each object originally on e's list.

An Example: UNION(e, g)



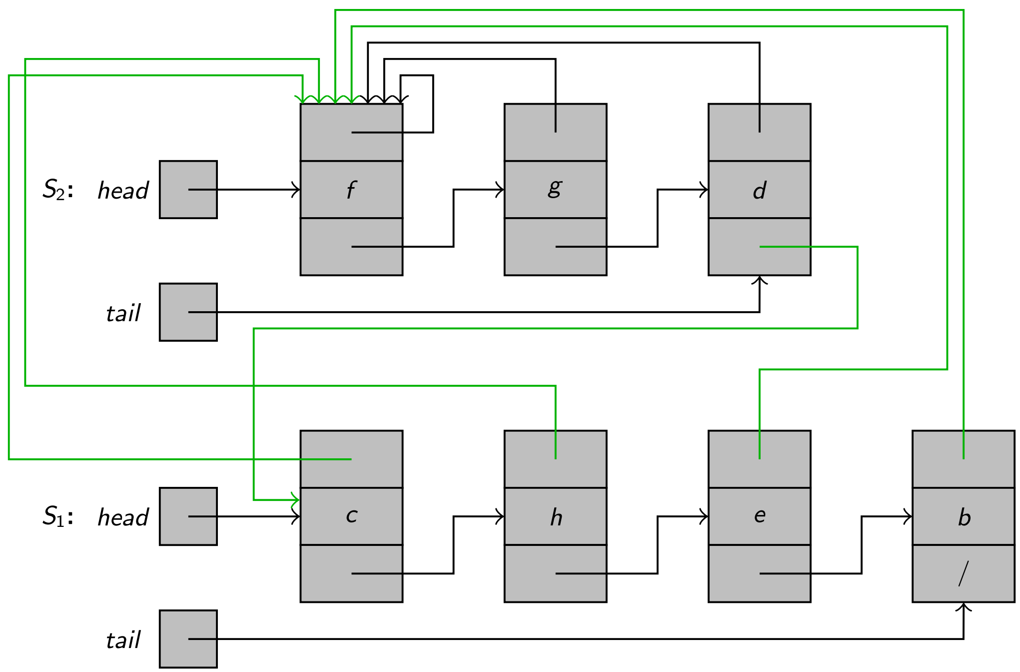
- However, we must update the pointer to the representative for each object originally on e's list.

An Example: UNION(e, g)



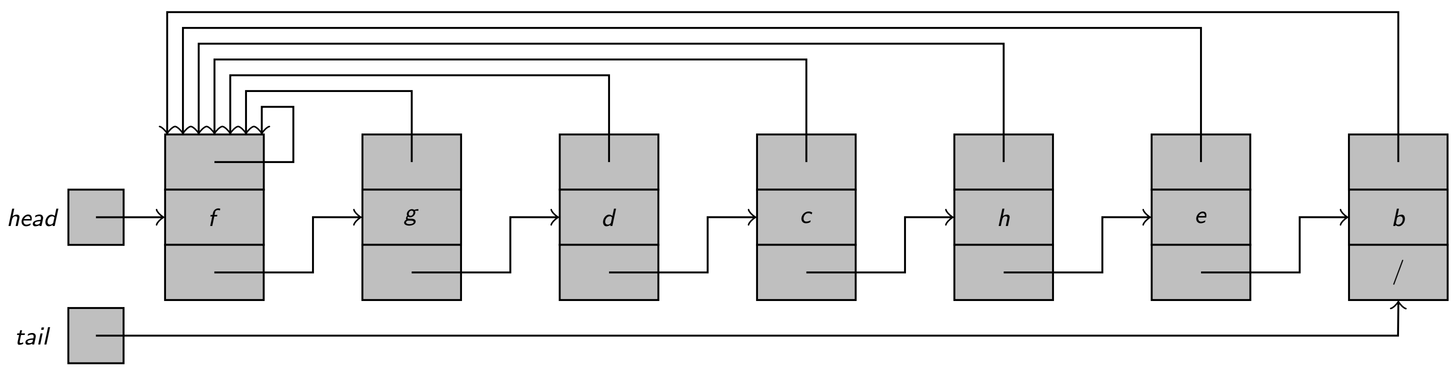
- However, we must update the pointer to the representative for each object originally on e's list.

An Example: $\textsc{Union}(e, g)$



- However, we must update the pointer to the representative for each object originally on e's list.

An Example: $\textsc{Union}(e, g)$



$$\textsc{Union}(e, g) = S_1 \cup S_2$$

Time: Linear in the length of e's list.

Worst Case Complexity

It is not difficult to come up with a sequence of m operations on n objects that requires $\Theta(n^2)$ time.

- Consider n objects $x_1, x_2, \ldots, x_n$.
- $\#$ MAKE-SET operations $= n$.
- $\#$ UNION operations $= n - 1$.
- $\therefore \; m = n + (n-1) = 2n - 1$.
- i^{th} UNION operation takes, i.e, UNION(x_i, x_{i+1}) takes i time.
- **Worst case complexity:**

$$n + \sum_{i=1}^{n} i = \Theta(n^2)$$

.

- $\because \; m = 2n - 1$, $\therefore$ on an average each operation takes $\Theta(n)$ time.

Operations	Number of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
$\vdots$	$\vdots$
MAKE-SET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
UNION(x_3, x_4)	3
$\vdots$	$\vdots$
UNION(x_{n-1}, x_n)	$n - 1$

Disjoint-set Forests

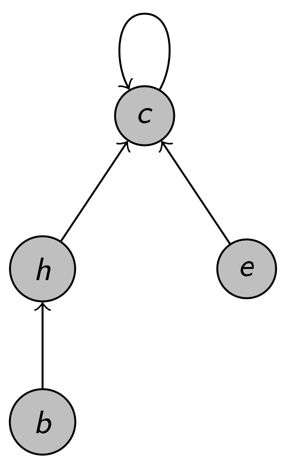
Disjoint-set Forests: A Faster Implementation

- Sets are represent by rooted trees.
- Each node contain one member.
- Each tree represents one set.
- Each member points (nodes) only to its parent.
- Set representative: The root of each tree.
  - It is it's own parent.
- **Note:** The straightforward algorithms that use this representation are not faster than ones that use the linked-list representation,
- But by introducing the following two heuristics one can achieve the asymptotically fastest disjoint-set data structure known.
  - Union by rank.
  - Path compression.

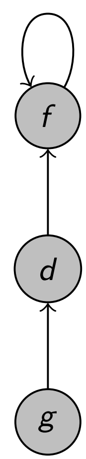
Operations

- MAKE-SET: Creates a tree with just one node.
- FIND-SET: Follows the parent pointers until the root of the tree is reached.
  - FIND-PATH: Nodes visited on this path toward the root.
- UNION: Make the root of one tree to point to the root of the other.

An Example: UNION(e, g)

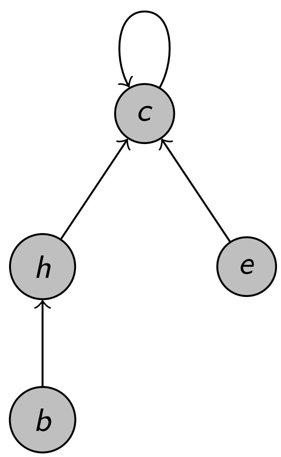


$S_1 = \{b, c, e, h\}$

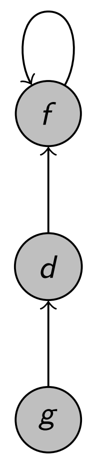


$S_2 = \{d, f, g\}$

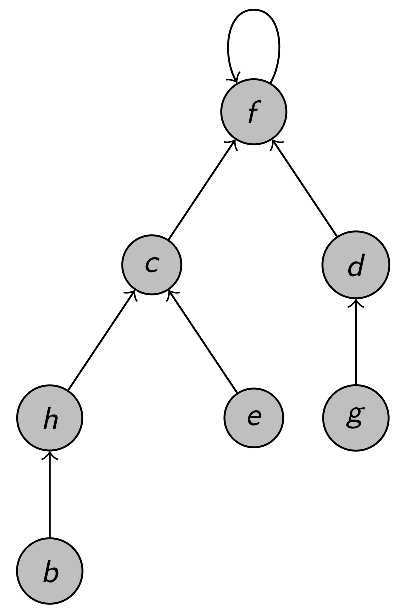
An Example: $\textsc{Union}(e, g)$



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



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



$\textsc{Union}(e, g) = S_1 \cup S_2$

Heuristics to Improve the Running Time

- So far, we have not improved on the linked-list implementation!
- A sequence of $n-1$ $\textsc{Union}$ operations may create a tree that is just a linear chain of n nodes $\Rightarrow$ $\textsc{Find-Set}$ takes $\Theta(i)$ time $\Rightarrow$ total complexity is $\Theta(n^2)$
- But by using two heuristics one can achieve a running time that is almost linear in the total number of operations m.

Heuristics 1: Union by Rank

- **Idea:** During UNION operation, make the root of the tree with fewer nodes point to the root of the tree with more nodes.

- Rather than explicitly keeping track of the size of the subtree rooted at each node, we shall use an approach that eases the analysis.

- For each node, we maintain a rank that is an upper bound on the height of the node.

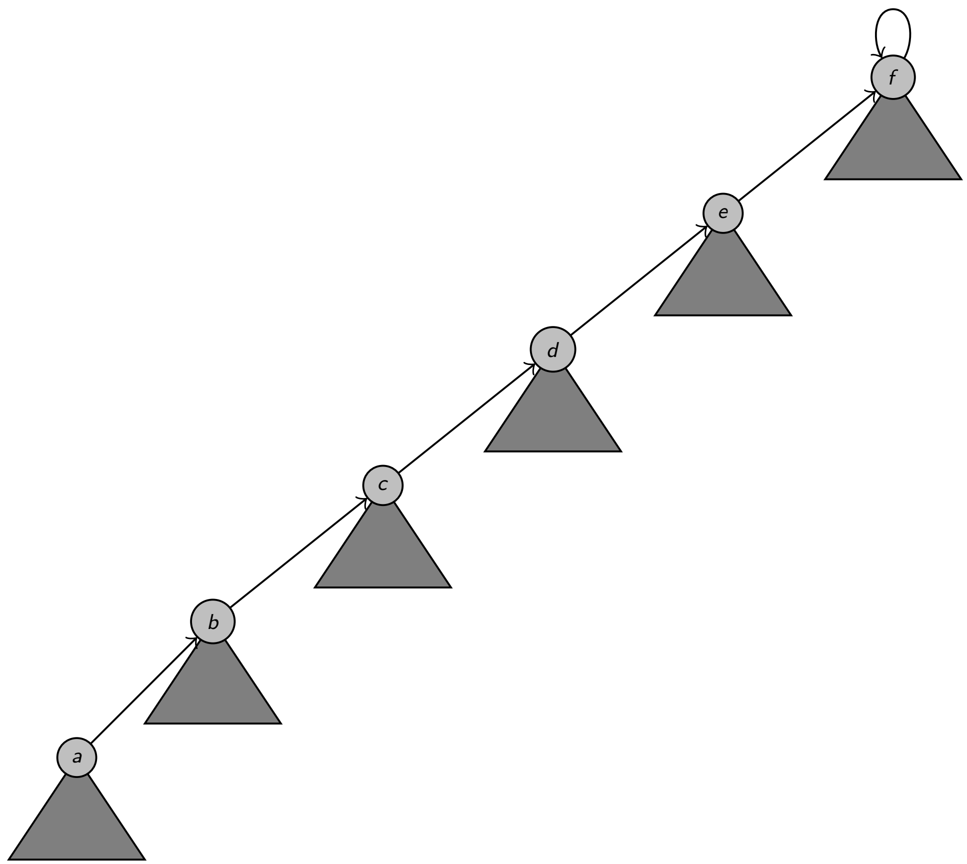
- UNION: Point the root with smaller rank to the root with larger rank.

Heuristics 2: Path Compression

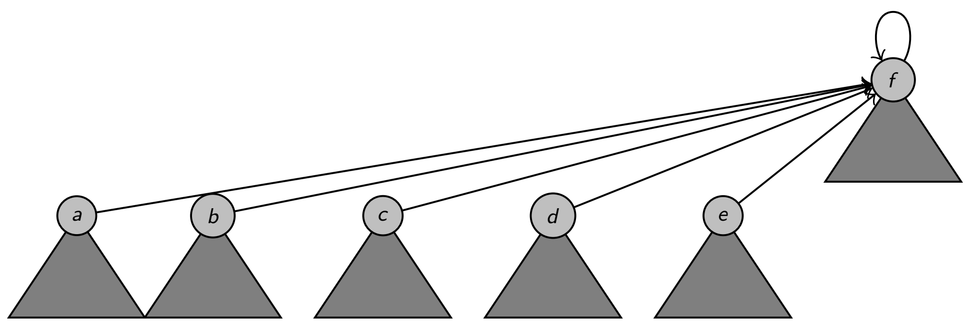
- FIND-SET: Make each node on the FIND-PATH to point directly to the root.

- Path compression does not change any ranks.

An Example: Before FIND-SET(a)



An Example: After FIND-SET(a)



Books and Other Materials Consulted

1. *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.

Thank You for your kind attention!

Questions!!