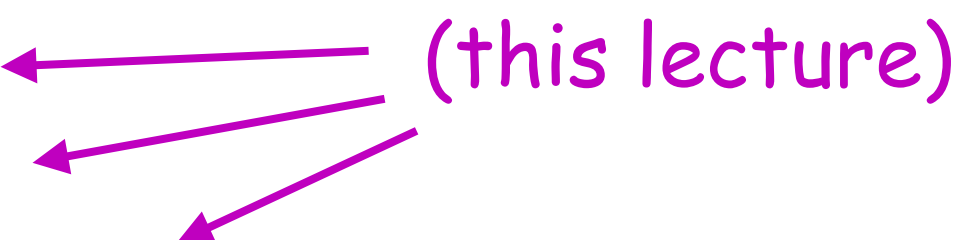# CS4311
Design and Analysis of Algorithms

Lecture 21:
Data Structures for Disjoint Sets II

About this lecture

- Data Structure for Disjoint Sets
  - Support Union and Find operations

- Various Methods:
  1. Linked List
  2. Union by Size ⟵ (this lecture)
  3. Union by Rank ⟵
  4. Union by Rank + Path Compression

2

Disjoint-Set Forest

- Another popular method to maintain disjoint sets is by a forest

  - Each set ⇔ a separate rooted tree
  - Representative ⇔ root of tree

- Unlike the linked lists implementation, each element now points only to its parent (and does not directly point to the representative)

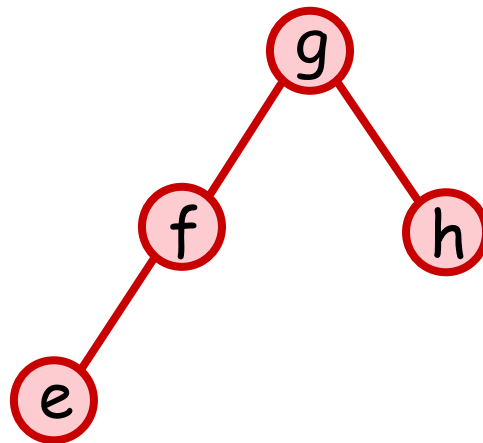# Example

Current dynamic sets : $\{ \{a,b,c,d\}, \{e,f,g\}, \{h\} \}$
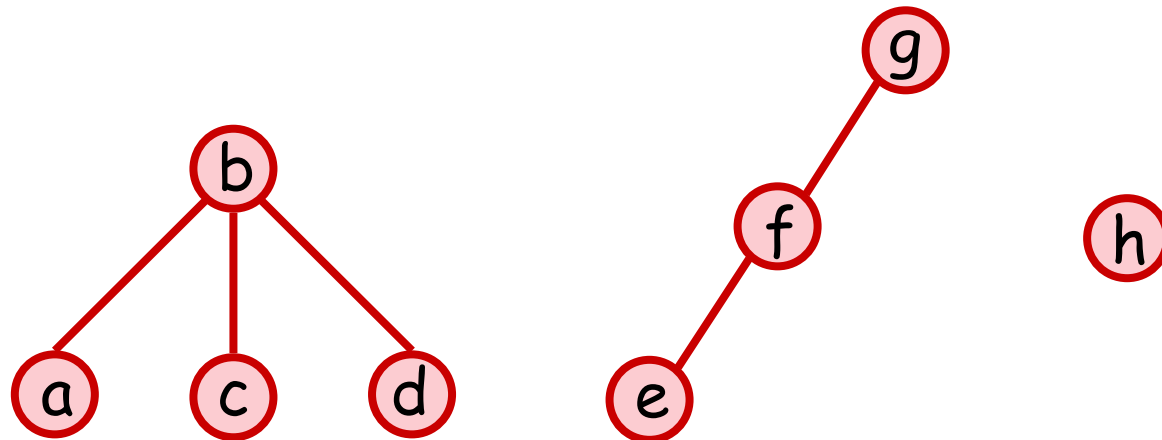
Disjoint-Set Forest

- To perform Union(x,y), we join the trees containing x and containing y, by linking their roots

- E.g. Union(f,h) in previous example gives:

Disjoint-Set Forest

- Let H_{max} = max height of all trees
- In the worst-case:

  Make-Set : $\Theta(1)$ time

  Find or Union : $O(H_{max})$ time

➔   m operations on n elements :

  worst-case $\Theta(mn)$ time

Union By Size

- Let us apply a union-by-size heuristic :

To perform Union, we link root of the smaller tree to root of the larger tree

➔ $H_{max} = O(\log n)$ (how to prove??)

➔ m operations : $\Theta(m \log n)$ time

Union By Rank

- A similar heuristic is called union-by-rank
- Each node keeps track of its rank – an upper bound on the height of the node
  - In a single-node tree (created by Make-Set)

rank of root = 0

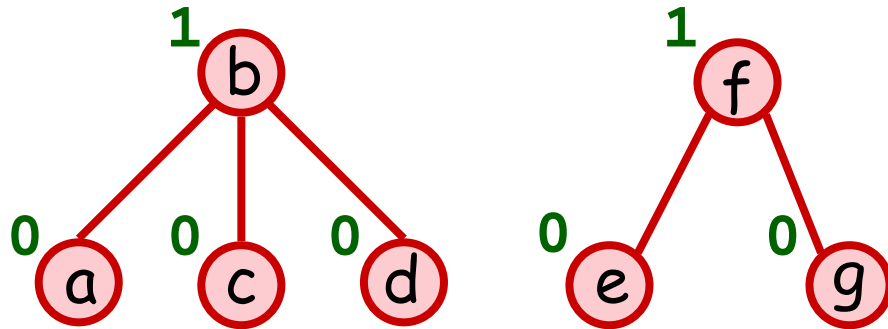To perform Union, we link root with smaller rank to root with larger rank

Union By Rank

- Rank needs not be very accurate
  - as long as it always gives an upper bound of height is enough

- When Union is performed, only the rank of the roots may change :
  - If both roots have same rank
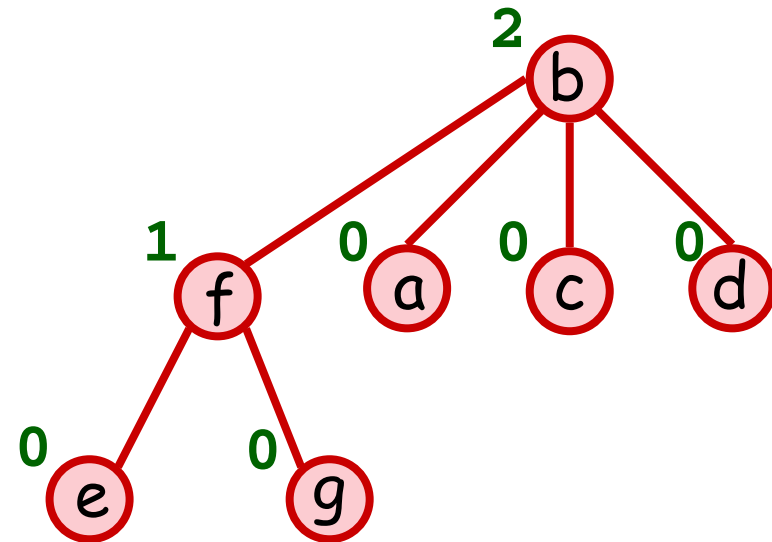   ➔ rank of new root increases by 1
  - Else, no change

Example

Before Union

After Union(c,f)

Union By Rank

- Let H_{max} = max height of all trees

  ➔ H_{max} = $O(\log n)$ (how to prove??)

  ➔ m operations : $\Theta(m \log n)$ time

- So, union by rank is no better than union by size, but …

Path Compression

- The closer a node to its root, the faster the Find or Union operation

- When we perform Find(x), we will need to find the root of the tree containing x

  ➔ will access every ancestor of x

- why don't we make all these ancestors of x closer to the root now?

   (Because no increase in asymptotic performance !!!)

Example

Before Find(x)
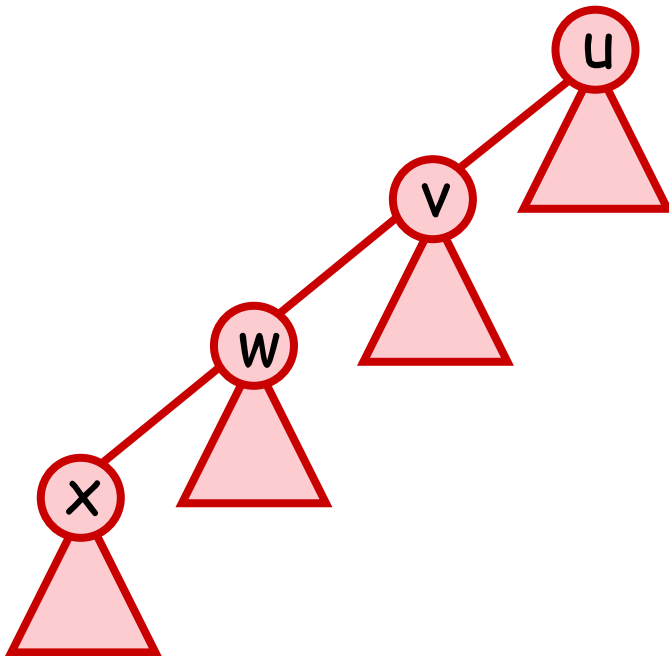
After Find(x)
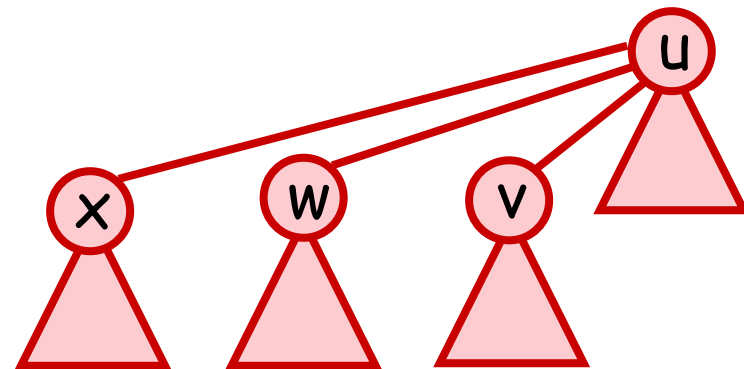
Union by Rank + Path Compression

- If Union(x,y) is always performed by first Find(x), Find(y), and then linking the roots,

  then by combining union-by-rank (at Union) and path compression (at Find and Union) :

  m operations: $\Theta(m\,\alpha(n))$ time

  Inverse Ackermann
  (in practice, at most 4)

Finding Connected Components

- Recall: To find connected components of a graph G with n vertices and m edges
  - there are n Make-Set and m Find or Union operations

- Which scheme for dynamic disjoint sets gives the best running time (theoretically)?

  Ans. Depends on m (why?)