

## Chapter 21: Data Structures for Disjoint Sets

Here we are thinking of a situation in which we need to maintain a family of sets of elements, where each set has a unique representative, in which we need perform the following fundamental operations on the sets:

1. **Make-Set** Given an element  $x$ , create a new set consisting solely of  $x$ .
2. **Find-Set** Given an element  $x$ , find the representative of the set to which  $x$  belongs.
3. **Union** Given two representatives  $x$  and  $y$ , join the sets represented by  $x$  and  $y$ . There is no rule about which element will be the representative of the union.

## The Key Issue

We do not need to be able to have efficient enumeration of all the representatives. Given that the three operations that need to be performed, it is easy to keep track of the changes in the number of sets that are maintained.

Since we have the concept of representatives, in each set we need to have a link between the representative and each of its members.

We assume that we maintain a list of elements, each of which is equipped with a pointer whose value is determined by a certain rule, and that the pointers are used to look for representatives.

*Think of an implementation.*

*How efficiently the three operations can be done with your implementation?*

## *The Disjoint Forest Implementation*

Each set is a tree with pointers to the parents. The root is the representative. The parent pointer of the root points to itself.

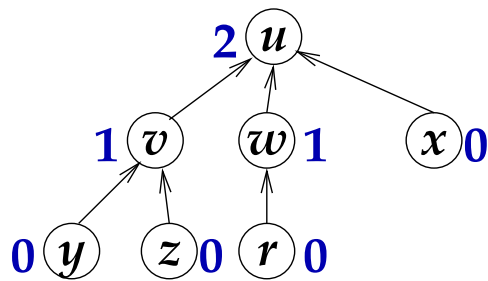
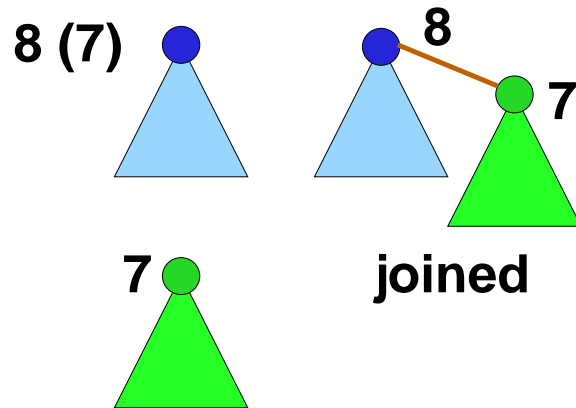
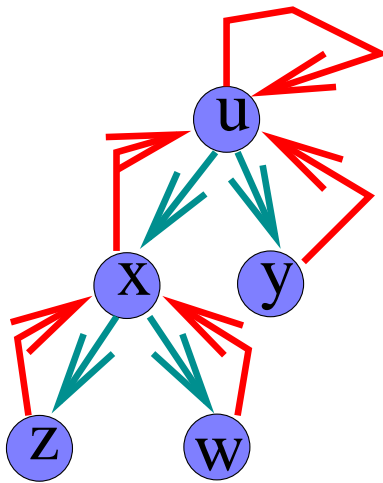
*What is the cost of*  
**Make-Set?**

## The Union-by-rank Heuristic

The heuristic is used for both union and find.


Here each node  $x$  keeps a value  $rank[x]$ . When a single-node tree is created by **Make-Set**, the value of  $rank$  is set to 0 for the single node.

For **Union**, make the one with the smaller root rank will become a child of the root of the other. A tie can be broken arbitrarily, but the rank of the root is **incremented by one**.



The root rank is the tree height and is a lower bound on the  $\lg$  of the tree size.

**Theorem A** For a sequence of  $m$  operations  $n$  out of which are **Make-Set**, the cost with the union-by-rank heuristic is  $O(m \lg n)$ .

**Proof** Since there are only  $n$  elements, the root rank is at most  $\lg n$ , so, the height is at most  $\lg n$ . The cost of **Make-Set** and that of **Union** are both  $O(1)$  and the cost of **Find** is a “big- $O$ ” of the maximum height, the total cost is  $O(m \lg n)$ . 

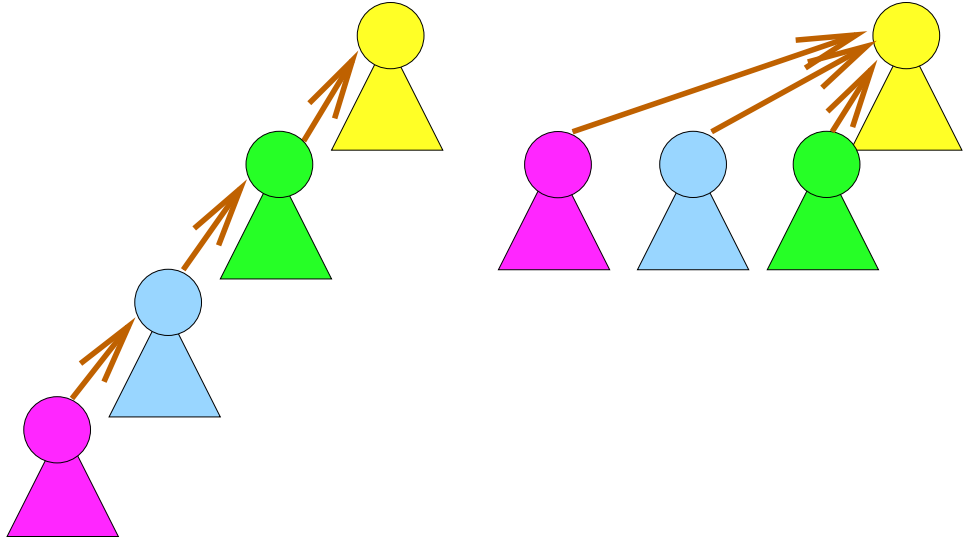
The bound is tight.

## The path-compression heuristic

When a node  $x$  is visited during **Find-Set**, redirect  $p[x]$  to the root without modifying ranks.

The root rank is an upper bound on the tree height.

*Is the root rank still an upper bound on the lg of the tree size?*





With the path-compression heuristic,  $n$  **Make-Set** operations plus  $f$  **FindSet** operations cost

$$\begin{aligned} \Theta(f \log_{1+f} n) & \text{ if } f \geq n \text{ and} \\ \Theta(n + f \lg n) & \text{ if } f < n. \end{aligned}$$

With the two heuristics,  $m$  basic operations  $n$  of which being **Make-Set** cost  $O(m \lg^* n)$ , where  $\lg^* n$  is smallest number  $k$  that the  $k$  stacks of 2 is greater than or equal to  $n$ .