Data Structure for Disjoint Sets

CS 535 Fall 2025

The Disjoint Sets Operations

Maintain a collection of disjoint sets (with a representative element each) and suporting:

1. MAKE-SET(x) creates a new set whose only member is x
2. UNION(x, y) unites the dynamic sets that contain x and y, say S_x and S_y , into a new set that is the union of these two sets.
3. FIND-SET(x) returns a pointer to the representative of the (unique) set containing x.

We have n elements and m operations.

Example where it useful

Connected components in a graph. Let us look at the textbook. One can use BFS or DFS instead.

But for some Minimum Spanning Trees algorithms, we do need Union and Find operations.

The Data Structure

In a **disjoint-set forest**, each member points only to its parent. The root of each tree contains the representative.

Example from

`https://www.cs.usfca.edu/~galles/visualization/DisjointSets.html`.

Here the root of the tree contains a negative number whose absolute value is the *rank* of the tree. For us, the rank will be at least $1+$ the tree height.

Pseudocode for operations

Using the web site variant (book is similar):

MAKE-SET(x): $x.p \leftarrow -1$

UNION(x, y): LINK(FIND-SET(x), FIND-SET(y))

FIND-SET(x):

$y \leftarrow x$

while ($y.p \geq 0$)

$\quad y \leftarrow y.p$

return(y)

LINK(x,y): $x.p \leftarrow y$

Running time analysis

Everything but FIND-SET(x) takes constant time (UNION(x, y) uses two FIND-SET()).

FIND-SET(x) worst-case running time is $\Theta(h+1)$, where h is the height of the tree containing x. In the worst case, this is $\Theta(n)$, see the simulation on the web site.

Running time improvement

We use *rank* to store an upper bound on $1+$ the height of the tree. Store it in a separate field for every node (as in the book) or have the negative value of the rank be the parent-field of the root of the tree. When we link trees, keep the rank as low as possible. Pseudocode on the next slide.

Improved Link(x,y)

LINK(x,y):

if ($|x.p| > |y.p|$)

$y.p \leftarrow x$ // x has the higher rank, remains root

else if ($|x.p| < |y.p|$)

$x.p \leftarrow y$ // y has the higher rank, remains root

else // $|x.p| = |y.p|$

$x.p \leftarrow y$ // y remains root/representative

$y.p \leftarrow y.p - 1$ // rank goes up

Analysis with Improved Link(x,y)

First of all, the height of tree is smaller than the rank of the root.

Second, a tree of rank j contains at least 2^{j-1} nodes. Proof by induction: true for $j = 1$.

And when the rank goes from j to $j + 1$, we union two disjoint trees with at least 2^{j-1} nodes each, for a total of at least $2^j = 2^{(j+1)-1}$ nodes.

As the number of nodes in the tree is at most n, we get $2^{j-1} \leq n$, or $j \leq 1 + \log_2 n$. This makes Find(x) run in $O(\log n)$ time.

Further Improvements

Path compression double the time of Find(x), but saves over the long term. See again the simulation; pseudocode straighforward.

Worst-case, Find(x) is still $\Theta(\log n)$ each but a sequence of m operations with n elements will take at most $(m+n)\alpha(m,n)$ running time, where $\alpha(m,n)$ grows very very slowly. The analysis will be done on the whiteboard.