

Data Structures for Disjoint Sets

Maintain a **Dynamic** collection of disjoint sets.

Each set has a unique representative (an arbitrary member of the set).

Make-Set(x) - Create a new set with one member x .

Union(x, y) - Combine the two sets, represented by x and y into one set.

Find-Set(x) - Find the representative of the set containing x .

Implementation: Disjoint-set forests

Represent each set with a rooted tree where the root is the *representative* of the set.

- **MAKE-SET**: creates a tree with just one node.
- **FIND-SET**: follows parent pointers to the root, returns root.
- **UNION**: makes the root of one tree point to the root of another.

Performance

Under a naive implementation, a sequence of m operations on n elements can take $O(mn)$ time.

Using **union by rank** heuristic the above time can be reduced to $O(m \lg n)$.

Using **path compression** heuristic the time can be reduced even further to $O(m \lg^* n)$!

lg* function

Intuitively, $\lg^*(n)$, or the *iterated logarithm*, is the number of repeated lgs of n required to get a value less than or equal to 1:

$$\lg^{(i)} n = \begin{cases} n & : i = 0 \\ \lg(\lg^{(i-1)} n) & : i > 0, \lg^{(i-1)} n > 0 \\ \text{undefined} & : i > 0, \lg^{(i-1)} n \leq 0 \text{ or undefined} \end{cases}$$

$$\lg^* n = \min \{ i \geq 0 : \lg^{(i)} n \leq 1 \}$$

It is a very slow growing function:

$$\begin{aligned} \lg^* 2 &= 1 \\ \lg^* 4 &= 2 \\ \lg^* 16 &= 3 \\ \lg^* 65536 &= 4 \\ \lg^* 2^{65536} &= 5 \\ \left. \begin{array}{l} \vdots \\ \cdot^2 \end{array} \right\} n &= n \end{aligned}$$

Union by rank

When executing a UNION operation, make the root of the tree with fewer nodes point to the root of the tree with more nodes.

Maintain a *rank* for each subtree which is an upper bound on the height of the node.

Every node x then has variables $rank[x]$, the rank of x , and $p[x]$, the parent of x .

Make-Set(x)

- 1 $p[x] = x$
- 2 $rank[x] = 0$

Union(x, y)

- 1 **Link(Find-Set(x), Find-Set(y))**

Link(x, y)

- 1 **If** $rank[x] > rank[y]$
- 2 $p[y] = x$
- 3 **else**
- 4 $p[x] = y$
- 5 **If** ($rank[x] = rank[y]$)
- 6 $rank[y] = rank[y] + 1$

Path Compression

When executing a `FIND-SET` operation, make each node along the find-path point directly to the root.

We define `FIND-SET` recursively so that it updates all the pointers along a find-path:

Find-Set(x)

If ($x \neq p[x]$)

$p[x] = \mathbf{Find-Set}(p[x])$

Return $p[x]$

Lemmas on rank

Lemma 1. *For all nodes x , $\text{rank}[x] \leq \text{rank}[p[x]]$ with strict inequality if $x \neq p[x]$. The value of $\text{rank}[p[x]]$ is monotonically increasing with time.*

Lemma 2. *For all tree roots x , $\text{size}(x) \geq 2^{\text{rank}[x]}$.*

Lemma 3. *For any integer $r \geq 0$, there are at most $n/2^r$ nodes of rank r .*

Proof. We can “identify” 2^r nodes *uniquely* with each node of rank r : these are the nodes belonging to the subtree rooted at the node of rank r .

If there were more than $n/2^r$ nodes of rank r , then the graph contains more than $n/2^r \cdot 2^r = n$ nodes, a contradiction. \square

Corollary 1. *Every node has rank at most $\lfloor \lg n \rfloor$.*

Lemma 4. *Any sequence S' of m' MAKE-SET, UNION, and FIND-SET operations can be converted into a sequence S of m MAKE-SET, UNION, and FIND-SET operations by turning each UNION into two FIND-SET operations followed by a LINK. Then if sequence S runs in $O(m \lg^* n)$ time, sequence S' runs in $O(m' \lg^* n)$ time.*

Proof. Since each UNION operation in S' is converted into three operations in S , we have $m' \leq m \leq 3m'$. Since $m = O(m')$, by proving an $O(m \lg^* n)$ bound for S implies an $O(m' \lg^* n)$ bound for the original sequence S' . \square

Main Result

Theorem 1. *A sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, can be performed on a disjoint-set forest with union by rank and path compression in worst-case time $O(m \lg^* n)$.*

Proof by Amortized Analysis

Assign a charge of 1 to each MAKE-SET and LINK operation.

Partition node ranks into *blocks* by putting rank r into block $\lg^* r$ for $r = 0, 1, \dots, \lfloor \lg n \rfloor$. Define $B(j)$ as follows:

$$B(j) = \begin{cases} -1 & \text{if } j = -1 \\ 1 & \text{if } j = 0 \\ 2 & \text{if } j = 1 \\ \left. \begin{matrix} \vdots \\ 2 \end{matrix} \right\}^j & \text{if } j \geq 2 \end{cases}$$

The j th block consists of the set of ranks

$$\{B(j-1) + 1, B(j-1) + 2, \dots, B(j)\}$$

for $j = 0, 1, \dots, \lg^* n - 1$.

Find-Set charges

We assign *two* types of charges for a FIND-SET operation.

block charge: Suppose the find-path is x_0, x_1, \dots, x_l where x_l be the root.

For each $j = 0, 1, \dots, \lg^* n - 1$, we assess one **block charge** to the *last* node with rank in block j on that path.

One **block charge** is also assessed to x_{l-1} .

path charge: Each node which does not receive a block charge receives a **path charge**.

Counting block charges

Lemma 5. *Once a node, other than x_l and x_{l-1} , is assessed block charges, it will never again be assessed path charges.*

Each time path compression occurs, for all nodes x_i (except x_{l-1}), the new parent of x_i has rank strictly greater than the old parent. Thus, the difference between $\lg^* \text{rank}[x_i]$ and $\lg^* \text{rank}[p[x_i]]$ can only increase with time. Once the two have ranks in different blocks, they will always have ranks in different blocks, so x_i will always be the last node in its block.

There is at most one block charge assessed for each block number on the given find path, plus one block charge for the child of the root, x_{l-1} .

Since block numbers range from 0 to $\lg^* n - 1$, there are at most $\lg^* n + 1$ block charges assessed for each FIND-SET operation.

Thus, there at most $m(\lg^* n + 1)$ block charges assessed over all FIND-SET operations.