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.

Computing Connected Components

Given a graph $G = (V, E)$ compute the connected components of G.

```
CONNECTED-COMPONENTS(G)
for  each vertex  v ∈ V[G]
do MAKE-SET(v)
for  each edge  (u, v) ∈ E[G]
do if FIND-SET(u) ≠ FIND-SET(v)
      then UNION(u, v)
```

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 $\lg$s of n required to get a value less than or equal to 1:

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

$$\lg^* n = \min \left\{ i \ge 0 : \lg^{(i)} n \le 1 \right\}$$

It is a very slow growing function:

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

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.

Pseudocode

MAKE-SET(x)

$p[x] \leftarrow x$
$rank[x] \leftarrow 0$

UNION(x, y)

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

LINK(x, y)

if $rank[x] > rank[y]$
    then $p[y] \leftarrow x$
    else $p[x] \leftarrow y$
            if $rank[x] = rank[y]$
                then $rank[y] \leftarrow rank[y] + 1$

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 $x \neq p[x]$
  then $p[x] \leftarrow$ FIND-SET($p[x]$)
return $p[x]$

Simple Lemmas on rank

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

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

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. □

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

Main Result

Theorem 1. *A sequence of m* MAKE-SET, UNION, *and* FIND-SET *operations, n of which are* MAKE-SET *operations, con 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

Proof. 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 $r = 0, 1, \ldots, \lfloor \lg n \rfloor$. Define $B(j)$ as follows:

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

The jth block consists of the set of ranks

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

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

Find-Set charges

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

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

For each $j = 0, 1, \ldots, \lg^* n - 1$, 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 4. *Once a node, other than x_l and x_{l-1}, is assessed block charges, it will never again be assessed path charges.*

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.

Path charges

Observations:

1. If a node x_i is assessed a path charge, then $p[x_i] \neq x_l$.
   $\Longrightarrow x_i$ must be assigned a new parent during path compression.

2. x_i's new parent must have higher rank than its old parent.

Lemma 5. *A node can be assessed at most* $B(j) - B(j-1) - 1$ *path charges while its rank is in block* j.

Counting path charges

We can bound the path charges using $N(j)$, the number of nodes with rank in block j:

$$N(j) \leq \sum_{r=B(j-1)+1}^{B(j)} \frac{n}{2^r}$$

for $j = 0$:

$$\begin{aligned} N(j) &= n/2^0 + n/2^1 \\ &= 3n/2 \\ &= 3n/2B(0) \end{aligned}$$

for $j \geq 1$,

$$
\begin{aligned}
N(j) &\leq \frac{n}{2^{B(j-1)+1}} \sum_{r=0}^{B(j)-(B(j-1)+1)} \frac{1}{2^r} \\
&< \frac{n}{2^{B(j-1)+1}} \sum_{r=0}^{\infty} \frac{1}{2^r} \\
&= \frac{n}{2^{B(j-1)}} \\
&= \frac{n}{B(j)} \leq 3n/2B(j)
\end{aligned}
$$

So, for any $j \geq 0$, we have $N(j) \leq 3n/2B(0)$.

Summing over all blocks to get $P(n)$, the overall number of path charges,

$$\begin{aligned}
P(n) &\leq \sum_{j=0}^{\lg^* n - 1} \frac{3n}{2B(j)} (B(j) - B(j-1) - 1) \\
&\leq \sum_{j=0}^{\lg^* n - 1} \frac{3n}{2B(j)} B(j) \\
&= \frac{3}{2} n \lg^* n
\end{aligned}$$

Total runtime

Thus the total number of charges incurred by FIND-SET operations is

$$O(\text{block charges} + \text{path charges}) = O(m(\lg^* n+1)+n(\lg^* n))$$

which is $O(m \lg^* n)$ since $m \geq n$.

Since there are $O(n)$ MAKE-SET and LINK operations, each with 1 charge, the total time is

$$O(m \lg^* n + n) = O(m \lg^* n)$$

□