Union-Find

Union-find stores a bunch of disjoint sets. Supports:

- from a member, find its owning set
- union two sets

Basic idea:

- Each set is a directed tree. Members are vertices.
- Use root to stand for the set.
- Edges go from children to parents. (So, only parent pointers.)

Union-Find: MakeSet

Initially, sets begin as single-element sets.

makeSet(a): create a set that has just one element a.

Union-Find: Union

To union two sets: have smaller set's root point to larger set's root.

union(b, g):

This requires storing a "size" field with each root, and updating at each union.

Union-Find: Find

To find set from element: follow pointers until root.

find(x) returns u.

Moreover, path compression: have every node along the path point to root directly.

One More Point on Union

What happens to union(x,y), if x or y is not a root?

Option 1: Disallow.

Option 2: Implicitly convert to union(find(x), find(y)).

\(n\) operations become at most \(3\cdot n\) basic operations.

Size and Rank

Each root has a size field = tree size.

What if a root becomes non-root after union? Imagine keeping it around, don’t change it anymore.

Note that due to path compression, non-roots lose descendents gradually.

So each non-root has a size field \(\geq\) subtree size.

Define \(rank(v) = \lfloor \log(v.size) \rfloor\). 
rank(parent) > rank(child)

\[ \text{rank}(v) = \lceil \log(v\cdot\text{size}) \rceil. \]

If \( v \) is non-root and \( v \)'s parent is \( w \):
\( w \) became \( v \)'s parent by a union in the past.
We always merge smaller trees into larger trees.

\[
\text{rank}(w) = \log(w\cdot\text{size}) \\
\geq \log(\text{old}\ w\cdot\text{size}) + v\cdot\text{size} \\
\geq \log(2v\cdot\text{size}) \\
\geq 1 + \log(v\cdot\text{size}) \\
> \text{rank}(v)
\]

How Many Nodes Have Same Rank

Parent rank > child rank. Ancestor rank > descendent rank.

(Assume \( v \neq w \))
If \( v, w \) have the same rank \( s \), then their descendents are disjoint.
\( v\cdot\text{size} \) and \( w\cdot\text{size} \) talk about two disjoint groups.

\[
\log(v\cdot\text{size}) \geq s \\
v\cdot\text{size} \geq 2^s
\]

Globally \( n \) elements, divide them into groups of at least \( 2^s \), you get at most \( n/2^s \) groups.

Each node of rank \( s \) claims one group.
You can have at most \( n/2^s \) such nodes.

Tower of 2 and Iterated Log

Define

\[
tower(0) = 1 \\
tower(i + 1) = 2^{tower(i)}
\]

Inverse function in a sense:

\[
\log^*(n) = \min\{i \in \mathbb{N} : n \leq tower(i)\}
\]

\( \log^*(n) \leq i \) iff \( n \leq tower(i) \)

E.g., \( \log^*(n) \leq 5 \) iff \( n \leq tower(5) = 2^{65536} \)

"practically \( O(5) \)"

Rank Group

Define: \( v \) and \( w \) are in the same "rank group" iff:

\[
\log^*(\text{rank}(v)) = \log^*(\text{rank}(w))
\]

Use \( \log^*(\text{rank}(v)) \) as the name of \( v \)'s rank group.

If globally at most \( n \) elements, largest possible rank group name:

\[
\log^*\log(n) \leq \log^*(n) - 1
\]

Rank group \( g > 0 \) has \( tower(g) - tower(g - 1) \) ranks.

How Many Nodes in Same Rank Group

Assume \( g > 0 \).

\[
\# \text{ of nodes in rank group } g \\
\leq \sum_{x = \text{tower}(g - 1) + 1}^{\text{tower}(g)} n/2^x \\
< \sum_{x = \text{tower}(g - 1) + 1}^{\text{tower}(g)} n/2^x \\
= 2n/2^{\text{tower}(g - 1) + 1} \\
= n/\text{tower}(g)
\]
Amortized Time Analysis: Outline

- union(root1, root2) is $O(1)$. Just need to figure out find.
- A mix of the accounting method and an aggregate argument.
- If find(x) goes through $k$ edges, it receives $k$ dollars and spends it all. But...
- Classify those $k$ dollars into two categories.
- For one category, get an upper bound per find.
- For another category, get a total upper bound over all operations. (Aggregate argument.)

Dollars for Find

$1 for each edge find(x) goes through. But there are two cases.

Let $(v, w)$ be an edge. ($w$ parent, $v$ child.)

- $w$ is root or $w$'s rank group $\neq v$'s
  At most $\log^{*}(n)$ rank groups.
  At most $\log^{*}(n) - 1$ edges crossing rank groups.
  At most $\log^{*}(n)$ edges in this category.
- $w$ is not root, and $v$ and $w$ same rank group
  Call this $\$1 charged to node $v$
  Will use an aggregate argument for dollars in this category.

Dollars Charged to A Node, Ever

$1 charged to $v$ iff $v$'s parent is not root and they are in the same rank group.

Every time $1$ is charged to $v$, find(x) performs path compression, $v$ gets new parent, higher rank than (ancestor of) old parent.

Eventually $v$ gets a parent in a different rank group. Then no more dollars charged to $v$.

If $v$ is in rank group $g > 0$, then at most $\text{tower}(g) - \text{tower}(g - 1)$ dollars charged to $v$, ever.

(If $v$ is in rank group $0$, then parent is root or parent is in different rank group.)

Dollars Charged to All Nodes, Ever

Upper bound:

$$
\sum_{g=1}^{\log^{*}(n)-1} \left( \# \text{nodes in rank group } g \right) \left( \text{tower}(g) - \text{tower}(g - 1) \right)
\leq \sum_{g=1}^{\log^{*}(n)-1} \left( \frac{n}{\text{tower}(g)} \right) \left( \text{tower}(g) - \text{tower}(g - 1) \right)
\leq \sum_{g=1}^{\log^{*}(n)-1} \left( \frac{n}{\text{tower}(g)} \right) \left( \text{tower}(g) \right) = \sum_{g=1}^{\log^{*}(n)-1} n \leq n \log^{*}(n)
$$

Amortized Time: Conclusion

In a sequence of $n$ operations:

- at most $n$ elements globally
- makeSet takes $O(1)$ time
- union(root1, root2) takes $O(1)$ time
- find(x) first category takes $O(\log^{*}(n))$ time each
- find(x) second category takes $O(n \cdot \log^{*}(n))$ time total over all $n$ operations

Total time $O(n \cdot \log^{*}(n))$.

Amortized time $O(\log^{*}(n))$. "Practically $O(5)$."