Union-Find data structures

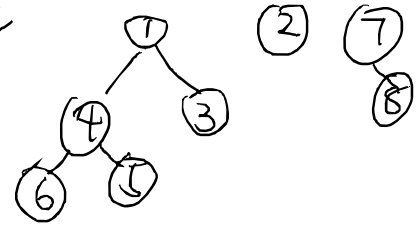
Maintain a collection of disjoint sets over $[n]$, $\{1,\dots,n\}$

supporting

- union(x,y): merge sets containing x and y
- find(x): return the representative of set containing x.

Initially, all sets are singletons $\{i\}$.

Recap: maintain a tree for each set, use the root as the representative



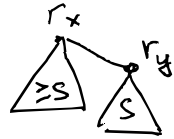
find(x):

keep going up,
return the root.

union(x,y)

r_x = find(x). r_y = find(y)
if $r_x \neq r_y$
parent[r_x] = r_y

Union-by-size:
maintain the size of the set at root,
when union two trees, set parent[smaller tree] to root of bigger tree



Union-by-size guarantees depth $\leq \log n$.

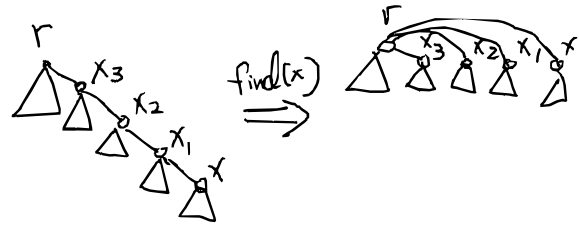
- going up one level (at least) doubles the size.

each operation takes $O(\log n)$ time

Path compression:

find(x): keep going up, set the parent of every node visited on the way to the root.

return root.



amortized $O(\log n)$

Union-by-size + path compression:

amortized $O(\alpha(n))$ time

↖ inverse Ackermann function

Ackermann function

$A(1, n) = n+2$ (+2)

$A(m, 1) = 2$

$A(m+1, n+1) = A(m, A(m+1, n))$

$A(2, n) = A(1, A(2, n-1)) = A(2, n-1) + 2 = 2 \cdot n$ (×2)

$A(3, n) = A(2, A(3, n-1)) = 2 \times A(3, n-1) = 2^n$ (power)

$A(4, n) = 2^{2^{2^{\cdots^2}}} \}n$

$A(5, n) = \underbrace{2^{2^{2^{\cdots^2}}} \}2^{2^{\cdots^2}} \}2^{2^{\cdots^2}} \}\cdots\}2^{2^{\cdots^2}}\}2}_{n}$

$A(m,n)$ is very fast growing even for $m=4$.

$\alpha(n) = \min_{k \geq 1} \{k : A(k,k) \geq n\}$. $\ll \log n$

$\ll \log^* n$

---

only path compression, no union-by-size.

Prove: Amortized time of union-find is $O(\log n)$.

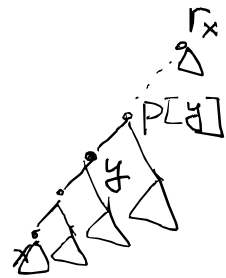
Consider a sequence of m union-find operations.

Connecting roots takes $O(1)$ time, thus, focus on Find!

Let $S(y)$ be the size of the subtree rooted at y.

Consider a find(x) operation

- visits all nodes on $x \to r_x$ path.

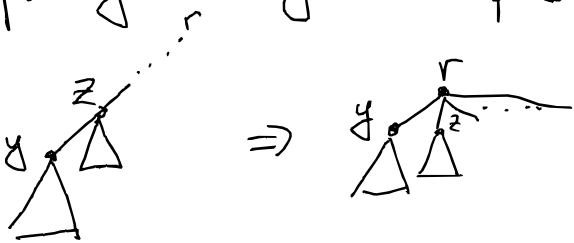


Call a node y on the path "light"

- if $S(y) \leq \frac{1}{2} S(p[y])$.

Claim: There are at most $\log n$ light y on this path.

Suppose y is heavy and $z = p[y]$ before path-compression.
(z is not the root)



In this case, $S(z)$ chop by a factor of two!

Claim: If a node z is not a root, then
z cannot become a root, and
$S(z)$ cannot increase.

Thus, every time we visit a heavy y s.t.
$z = p[y]$ is not the root,
$S(z)$ drops by a factor of two. This can happen to
z at most $\log n$ times.

During the $O(m)$ Find ops,
- visit $O(m \log n)$ light vertices
- visit $O(m \log n)$ heavy y, s.t. $p[y]$ is not the root.
- visit $O(m)$ heavy y, s.t. $p[y]$ is the root.
  only $O(m)$ nodes z can be part of a tree
  that is not a singleton.

Thus, all Find visits $O(m \log n)$ nodes.
Amortized time is $O(\log n)$.

Path compression + union-by-size.

Prove: amortized time $\le O(\log^* n)$ (weaker bound)

$\log^* n := \min_k \{k : \underbrace{\log\log\cdots\log}_{k} n \le 1\}$

$= \min_k \{A(4,k) \ge n\}$.

Consider $O(m)$ operations.

For a node y, define

$\overline{S}(y) :=$ max subtree size rooted at y during the process.

For a nonroot y, $\overline{S}(y) = S(y)$ when y becomes nonroot

$2\,\overline{S}(y) \le \overline{S}(p[y])$ union-by-size

Partition vertices into $\log^* n$ buckets based on $\overline{S}(y)$.

bucket i contains all y s.t. $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{B_i} \le \overline{S(y)} < \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{B_{i+1}=2^{B_i}}$

Claim: # nodes in bucket i is

$O\left(\frac{n}{B_i}\right)$.

Consider a node y visited during a find:

- y or $p[y]$ is root: $O(1)$ per op
- $y, p[y]$ in different buckets: $\log^* n$ per op.
- $y, p[y]$ in same bucket:

  r be root, $\overline{S}(r) \geq 2\overline{S}(p[y])$

  $p[y]$ is set to y

  thus, $\overline{S}(p[y])$ increases by $\times 2$.

  can happen $\leq \log \frac{B_{i+1}}{B_i} < B_i$ times to y.

In total, this case happens

$$\sum_{\text{bucket } i} \sum_{y \in \text{bucket } i} B_i = \sum_{i \leq \log^* n} O\left(\frac{n}{B_i} * B_i\right)$$

$$= O(n \log^* n).$$