Disjoint Set Union-Find I

Equivalence Relation:

Binary relation over set S such that $\forall a,b,c \in S$:

- **reflexive**: $a \equiv a$
- **symmetric**: $a \equiv b \Rightarrow b \equiv a$
- **transitive**: $a \equiv b + b \equiv c \Rightarrow a \equiv c$

Any equivalence relation defines a **partition** over S.

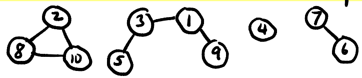
Examples:

- Given prime p, $a \equiv b \bmod p$

  E.g. $p=5$; **Partition**: $\{0,5,10,\ldots\}, \{1,6,11,\ldots\}, \{2,7,12,\ldots\} \ldots$

- Given graph G, vertices u,v,

  $u \equiv v$ if in same connected component



Partition: $\{2,8,10\}, \{1,3,5,9\}, \{4\}, \{6,7\}$

Union-Find:

Given set $S = \{1,2,\ldots,n\}$ maintain a partition supporting ops:

- **Init**: Each element in its own set $\{1\}, \{2\}, \ldots, \{n\}$
- **Union(s,t)**: Merge two sets $s+t$ and replace with their union
- **Find(x)**: Return the set containing x

Example:

Suppose: $\underbrace{\{1,5\}}_{s_1}, \underbrace{\{2,6,8\}}_{s_2}, \underbrace{\{3,4,7\}}_{s_3}$

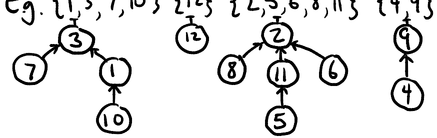
$\text{Union}(s_1, s_3) \rightarrow \{1,3,4,5,7\}, \{2,6,8\}$

$\text{Find}(5) \rightarrow s_1 \quad \text{Find}(8) \rightarrow s_2$

Inverted-Tree Approach:

- Store elements of each set in tree with links to **parent**
- Root node is **set identifier**

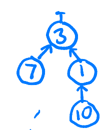
Eg. $\{1,3,7,10\}$ $\{12\}$ $\{2,5,6,8,11\}$ $\{4,9\}$



Array-Based Implementation:

Assume: $S = \{1,2,\ldots,n\}$

parent[1..n], where **parent[i]** is i's parent index or **0 = null if root**



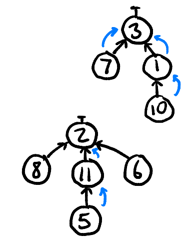
1	2	3	4	5	6	7	8	9	10	11	12
3	0	0	9	11	2	3	2	0	1	2	0

Set Identifiers are indices of root nodes

E.g. Find(7) = 3

Find(10) = 3

Find(5) = 2



Two items in same set iff Find(x) = Find(y)

A simple approach to finding is to trace the path to the root –

Set ≡ Element ≡ int

$1 \le x \le n$

```
Set Simple-Find(Element x) {
    while (parent[x] ≠ null)
        x ← parent[x]
    return x
```

Disjoint Set Union-Find II

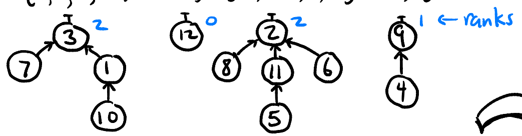
```
Set Union(Set s, Set t) {
  if (rank[s] > rank[t])
    swap s & t
  parent[s] ← t
  rank[t] ← max(rank[t], 1+ rank[s])
  return t
```

Recall: These are just array indices of roots

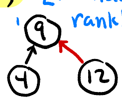
Init: All ranks ← 0

Example:

{1,3,7,10}   {12}   {2,5,6,8,11}   {4,9}

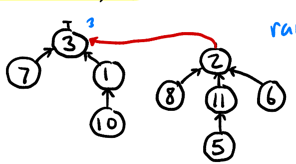


Union(9,12) [12 has lower rank]



rank[9] = max(rank[9], 1+ rank[12])
= max(1,1)
= 1

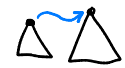
Union(2,3) [Both have same rank]



rank[3] = max(rank[3], 1+ rank[2])
= max(2,3)
= 3

How to Union?

- Just link one tree under the other



- How to maintain low heights?
- **Rank:** Based on height of tree. Link lower rank as child

Running Time?

Init: $O(n)$ - set a parents to null & ranks to 0

Union: $O(1)$ - constant time

Find: $O(\text{tree height})$

What is worst case?

We'll show tree height $= O(\log n)$

$\Rightarrow$ Find takes $O(\log n)$ time

Lemma: Assuming rank-based merging a tree of height h has at least 2^h nodes.

Proof: By induction on num. of unions

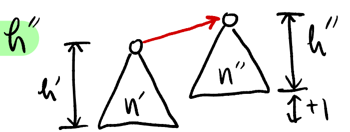
Basis: Single node. $h=0$, $2^0=1$ nodes

Step: Consider the last of series of unions. Let T' & T'' be trees to merge: Heights: h' & h''

Sizes: n' & n''

By induction: $n' \geq 2^{h'}$ $n'' \geq 2^{h''}$

Cases: $h' = h''$



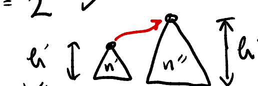
Final tree **height**: $h = h'+1 = h''+1$

Final **size**:

$n = n'+n'' \geq 2^{h'} + 2^{h''} = 2^{h-1} + 2^{h-1}$

$= 2 \cdot 2^{h-1} = 2^h$ ✓

Case 2: $h' < h''$



Final **height**: $h = h''$

Final **size**:

$n = n'+n'' \geq 2^{h'} + 2^{h''} \geq 2^{h''} = 2^h$ ✓

Case 3: $h' > h''$ (symmetrical) □

Disjoint Set Union-Find III

Simple Union-Find performs a sequence of m Unions + Finds on set of size n in $O(m \log m)$ time.

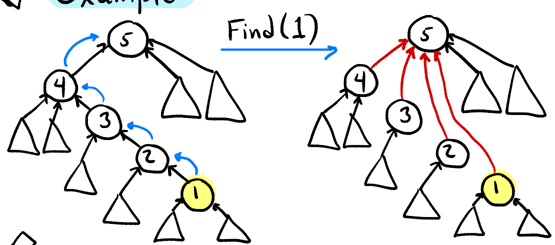
⇒ **Amortized time** (average per op) is $O(\log m)$

- Not bad - But can we do better?

Path Compression:

- Whenever we perform find, "short-cut" the links so they point directly to root
- This does not increase running by more than constant, but can speed up later finds

Example:



Does this little trick improve running times?

→ **Worst case** - No. Find may take $O(\log n)$ time

→ **Amortized** - Yes! Huge improvement! (But hard to prove)

Digression: Ackerman's Function (1926)

for $i, j \geq 0$

$$A(i,j) = \begin{cases} j+1 & \text{if } i=0 \\ A(i-1,1) & \text{if } i>0, j=0 \\ A(i-1, A(i,j-1)) & \text{o.w.} \end{cases}$$

Looks innocent, but it's a monster!

i \ j	0	1	2	3	4	5	...	
0	1	2	3	4	5	6		$A(0,j) = j+1$
1	2	3	4	5	6	7		$A(1,j) = j+2$
2	3	5	7	9	11	13		$A(2,j) = 2j+3$
3	5	13	29	...				$A(3,j) = 2^{j+3}-3$
4	13	BIG!	REAL BIG!	...				$A(4,j) = \underbrace{2^{2^{2^{\cdot^{\cdot^{2}}}}}}_{j+3} - 3$
⋮								

More than atoms in universe

Obs: $\alpha(m,n) \leq 4$ for any imaginable values of m, n $(m \geq n)$

From super big to super small

Inverse of Ackerman:

$$\boxed{\alpha(m,n) = \min\left\{ i \geq 1 \;\middle|\; A(i, \lfloor m/n \rfloor) > \log m \right\}}$$

Theorem: (Tarjan 1975) After init. any seq of m Union-Finds (with path compression) takes total time $O(m \cdot \alpha(m,n))$.

⇒ **Amortized time** $= O(\alpha(m,n))$

[For all practical purposes, this is constant time.]