

The Union-Find Problem

①

We look at the problem maintaining a system of sets that are pairwise disjoint. It should support two operations: (1) Find (2) Union.

C : collection of subsets of $\{1, 2, \dots, n\}$
s.t. $\bigcup_{I \in C} I = \{1, 2, \dots, n\}$ and $I \cap J = \emptyset$
if $I, J \in C$.

Find(i): determines the set $I \in C$ with $i \in I$.

Union(I, J): Joins sets I and J in C .

Often in applications we need the above two operations in the following way:

$I := \text{Find}(i); J := \text{Find}(j);$

If $I \neq J$ then Union(I, J) endif.

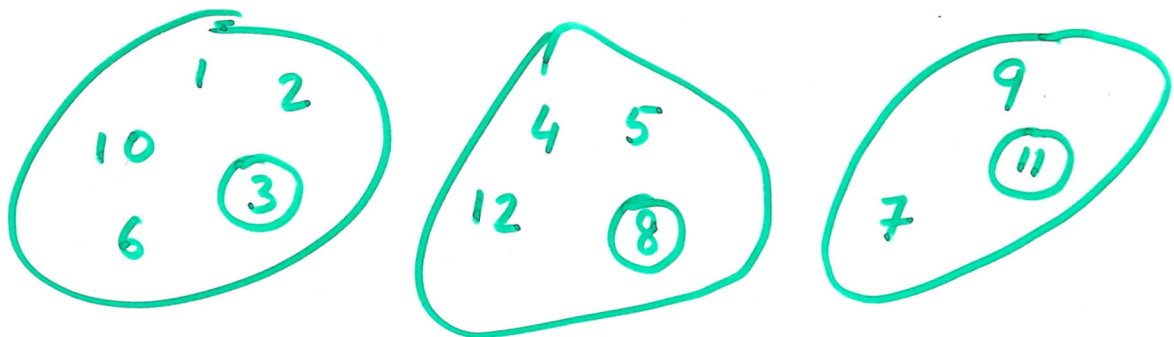
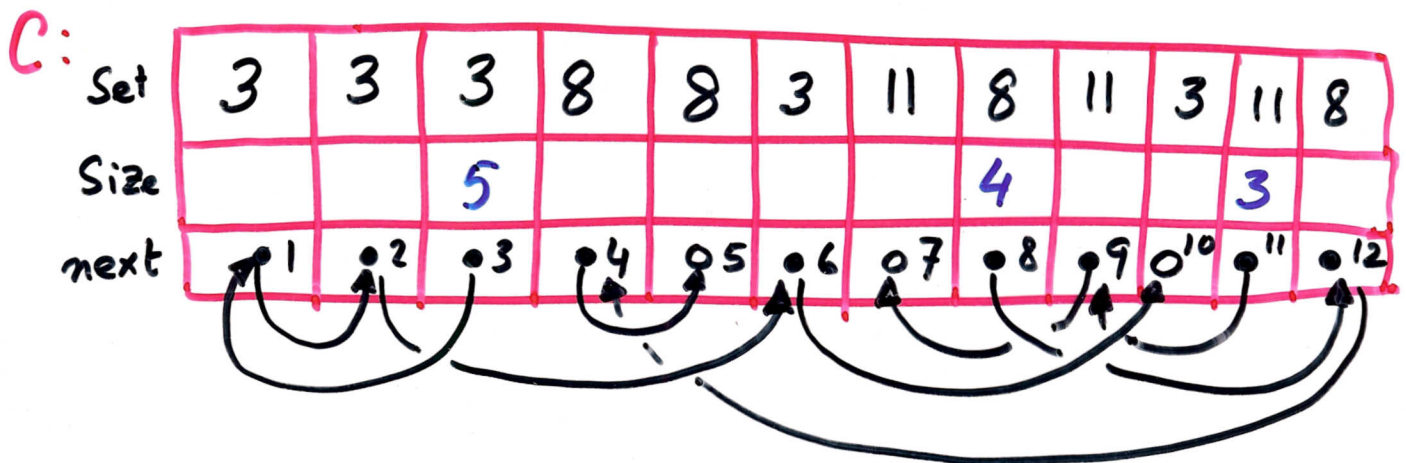
Here it does not really matter what I and J really are, except that they need to be different iff they represent different sets.

A simple solution.

(2)

C : array $[1 \dots n]$ of integers

Each set is represented by one of its elements, and $C[i]$ stores the name (the index of the representative) of the set containing i .



Finding a set takes $O(1)$, but union takes $\Theta(n)$ since the entire array needs to be scanned in the worst-case.

The previous solution can be improved (3)
by storing

- (i) the elements of a set in a linked list (next pointer)
- (ii) the size of a set at its representative

```
function Find(i)
  return C[i].set
```

```
procedure Union(I, J)
  if C[I].size < C[J].size then I ↔ J endif
  C[I].size := C[I].size + C[J].size;
  second := C[I].next; C[I].next := J;
  t := J; loop
    C[t].set := I;
    if C[t].next := 0 then
      C[t].next := second
      exit loop
    endif
    t := t.next
  endloop
```

The worst-case of a single union operation (4) is still $\Theta(n)$, as before, but now we can show a logarithmic amortized bound.

Claim $n-1$ union operations take time $O(n \log n)$

proof We consider the size of the set that contains the element i . So define

$$\sigma(i) = C[\text{Find}(i)]. \text{ size.}$$

$\sigma(i)$ changes whenever i is touched in the union operation; in this case the new $\sigma(i)$ is at least twice as large as the old one. This is because i is touched only if it belongs to the smaller of the two sets joined. Define k as the number of times element i is touched.

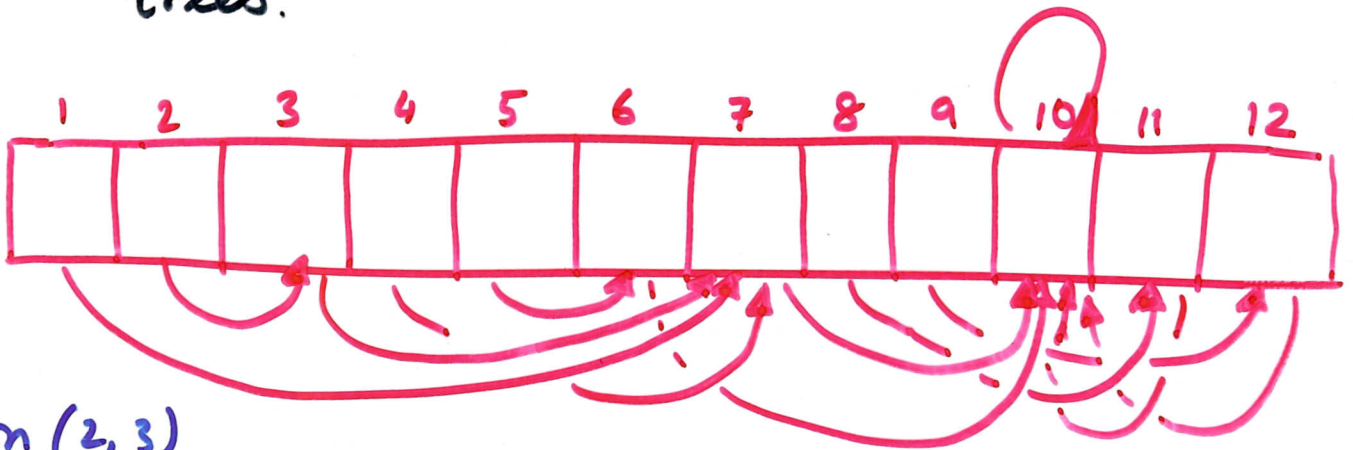
$$\text{Then } \sigma(i) \geq 2^k \implies k \leq \log n.$$

Tree Representation

⑤

We consider representing each set as a tree.

- idea - each set is represented by
- Find(i) traverses the path from i up to the root.
 - Union(I, J) links the two trees.



Ex.

Union (2, 3)

" (4, 7)

" (2, 4)

" (1, 2)

" (4, 10)

" (9, 12)

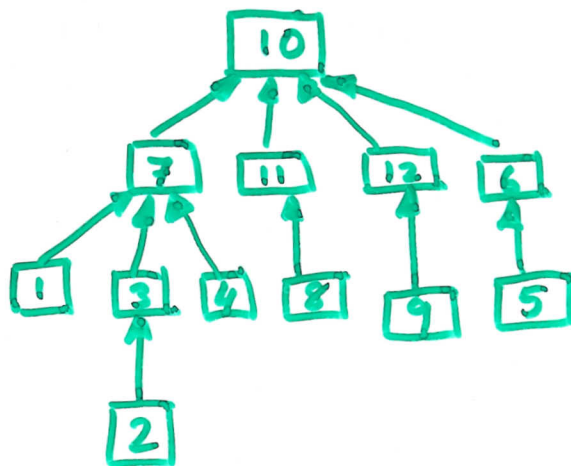
" (12, 2)

" (8, 11)

" (8, 2)

" (5, 6)

" (6, 1)



Union takes $O(1)$ time,
Find takes time proportional to
the depth of the ~~tree~~ node.

Weighted Merging. The same idea as (6) before improves time: instead of joining arbitrarily, join the smaller to the larger tree.

Assume: C has fields

p ... index of parents,
index to itself if root
 h ... height of the tree

function Find(i)

```
if  $C[i].p = i$  then return  $i$ 
else return Find( $C[i].p$ )
endif
```

procedure Union(I, J)

```
if  $C[I].h < C[J].h$  then
     $C[I].p := J$ 
else
     $C[J].p := I$ ;
    if  $C[I].h = C[J].h$  then
         $C[I].h := C[J].h + 1$ 
    endif
endif
endif
```

Claim. The height of a tree with n nodes (7) is at most $\log n$.

So, Find takes $O(\log n)$ time.

Union takes $O(1)$ time.

Path Compression

The idea is to connect all nodes visited during a Find operation directly to the root.

function Find(i)

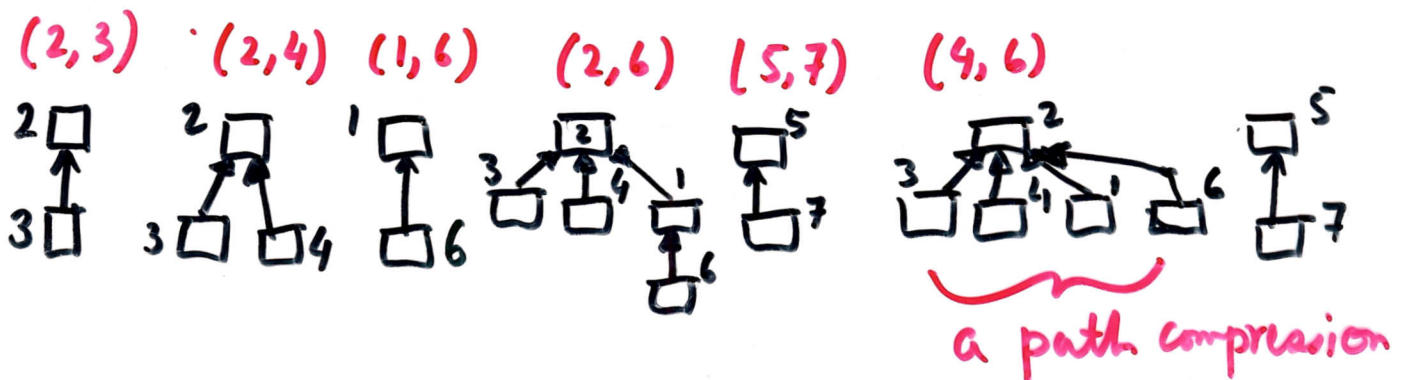
if $C[i].p \neq 0$ then $C[i].p := \text{Find}(C[i].p)$ endif

return $C[i].p$

Example (i, j) stands for

$I := \text{Find}(i); J := \text{Find}(j);$

If $I \neq J$ then Union(I, J)



Ackermann's Function

⑧

It can be shown that m find operations take $O(m \alpha(m))$ time where

$\alpha(m)$ is the slowly growing inverse Ackermann's function.

Def. $A_k(1) = 2$ for $k \geq 1$
 $A_1(n) = 2n$ for $n \geq 1$
 $A_k(n) = A_{k-1}(A_k(n-1))$ for $k, n \geq 2$ } Ackermann's function

	n=1	2	3	4	5	6
k=1	2	4	6	8	10	12
2	2	4	8	16	32	64
3	2	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$	$2^{2^{2^{2^{2^2}}}}$
4	2	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$	$2^{2^{2^{2^{2^2}}}}$
5	2	2^2	2^{\dots}	2^{\dots}	2^{\dots}	2^{\dots}

tower of height $2^{2^{2^2}}$

$$\alpha(m) = \min \{ n \mid A_n(n) \geq m \}$$

For all practical purposes $\alpha(m) \leq 4$, but $\alpha(m)$ goes to infinity as m goes to ∞ .