27 Oct 2021

UNION - FIND

2 operations: 0th → create the data structure / init
- list of sets -
1st → UNION: given 2 sets, make them 1
2nd → FIND: given an "element", find which set it belongs to.

Graph: we often have "vertex" and/or "edge" objects

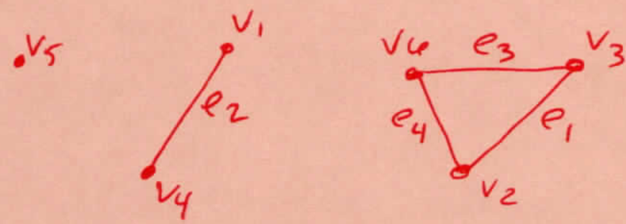
- vertex: degree
  LL of edges
  LL of vertex neighbors
  value / data
  → id
  → geometric location (x, y)
- edge: weight
  head (vertex)
  tail (vertex)
  data
  → id
  → embedding

Problem: Given a graph $G = (V, E)$, how many conn. comp. do I have?

$\boxed{\text{COUNT COMP}(G)}$

```
S.create(V) ← each vertex is its own set right now
E' ← E
while E' ≠ ∅
    e ← E'.pop()
    S1 ← S.FIND(e.head)
    S2 ← S.FIND(e.tail)
    if S1 ≠ S2
        S.UNION(S1, S2)
    endif
end while
return |S|
```

try this on a small example:



e	s_1		s_2		S
e_1	v_3	$\neq$	v_2	$\Rightarrow$	$\{v_1, v_2 = v_3, v_4, v_5, v_6\}$
e_2	v_1	$\neq$	v_4	$\Rightarrow$	$\{v_1 = v_4, v_2 = v_3, v_5, v_6\}$
e_3	v_3	$\neq$	v_6	$\Rightarrow$	$\{v_1 = v_4, v_2 = v_3 = v_6, v_5\}$
e_4	v_6	$=$	v_2	$\Rightarrow$	NO change

$|S| = 3.$

How do we know & how long does it take to figure out?

$\boxed{\text{Lemma}}$ Adding an edge to a graph has one of two effects:

① combines 2 distinct connected components "negative" / death

② creates a loop "pos" / birth

How do we store this?

<u>Try 1</u>: We have an array, indexed by the vertices that <u>stores</u> the ~~id~~ <u>/label</u> of the conn. comp (init labels are vertex ids)

<u>PICTURE</u>

init:

1	2	3	4	5	6
1	2	3	4	5	6

6 conn. comp

↓ add e_1

1	2	3	4	5	6
1	2	2	4	5	6

5 conn. comp.

these are my 2 verts. one of their labels must change

↓ add e_2

1	2	3	4	5	6
4	2	2	4	5	6

4 conn. comp.

FIND(v_1) FIND(v_4

↓ add e_3

1	2	3	4	5	6
4	6	6	4	5	6

3 conn. comp.

↓ add e_4

4	6	6	4	5	6

3 conn. comp. (+ 1 loop)

Each ~~set~~ (.) is its own set

FIND: $\Theta(1)$ "quick find"

UNION: $\Theta(n)$, assuming we've done the find

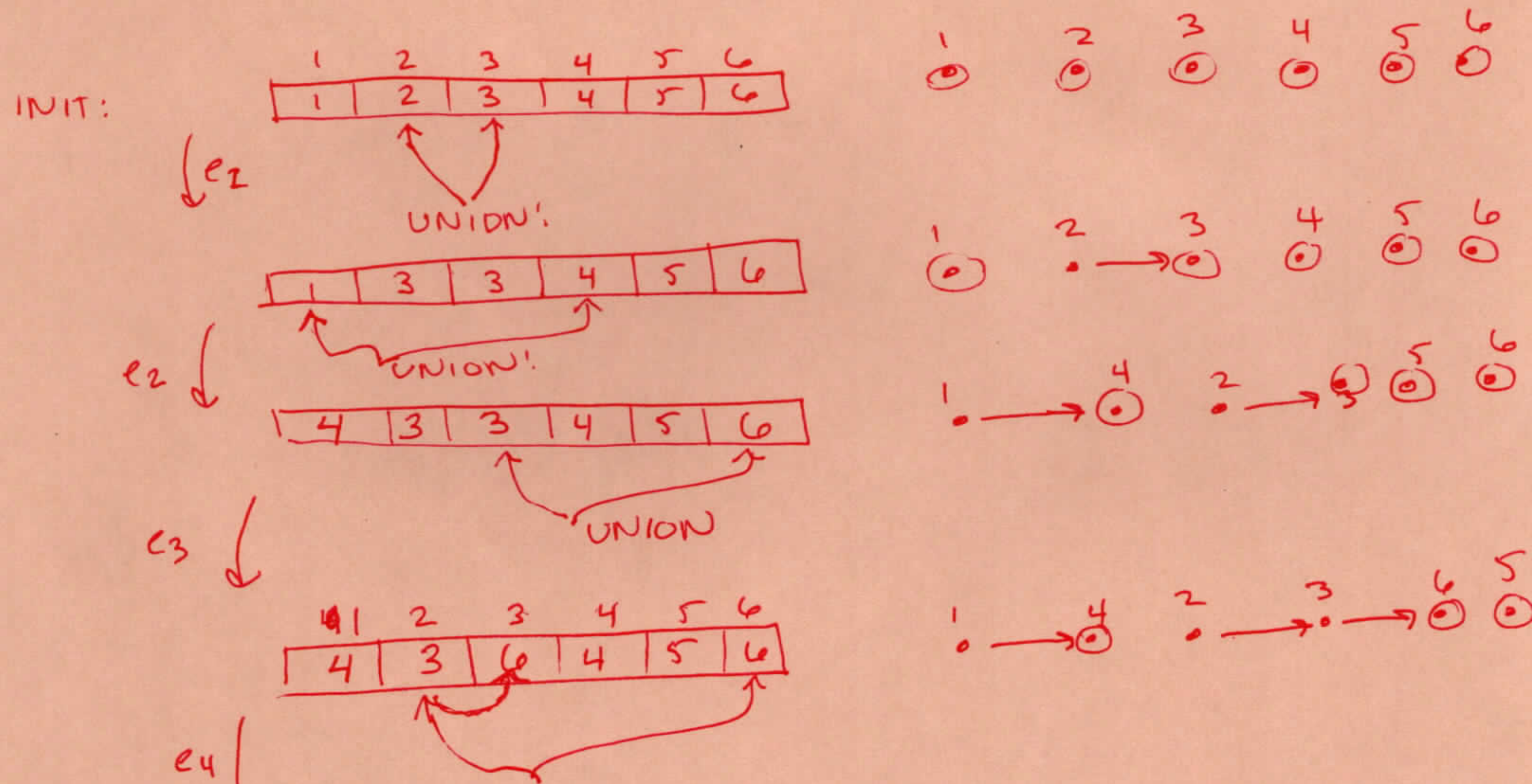
M operations $\Rightarrow O(Mn)$ time

Try 2 "quick union"

think of ~~this~~ each conn. comp. as a rooted tree. We store an array, indexed by vertices, that stores the parent of each node. (Root pts to itself)

UNION: Given 2 diff roots, make one point to the other.

FIND: Goes up the tree.



NO change

UNION: $\Theta(1)$

FIND: $\Theta(n)$ or $\Theta(\min(n,m))$

M operations $\Rightarrow O(Mn)$

$\boxed{m = O(n^2)}$

hunch: balance tree $O(M \log n)$ w/ amortization.