Algorithms - I

Princeton University

① Union Find Algorithm.

Here, the problem is, to find connections between nodes and also to make connection between nodes.

First we will initialize the array with their index value.

arr =

~~0~~	~~1~~	~~0~~	~~3~~	~~3~~	~~5~~	~~6~~	~~7~~	
0	1							

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

Union (0,2) → Means we have to connect the nodes.
So what we will do is, we will make changes in the values where index 0 and 2 both will be represented by 0 as their root

0	1	0	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

Union $(3,4)$ $\rightarrow$

0	1	0	3	3	5	6	7	8
0	1	2	3	4	5	6	7	8

Union $(2,4)$ $\rightarrow$

0	1	0	0	3	5	6	7	8
0	1	2	3	4	5	6	7	8

Here we can see that, the $(2,4)$ already has their parent.

$2 \rightarrow 0$, $\cancel{0} \longrightarrow 4 \rightarrow 3$

We will keep 2's parent to 0.

We will keep 4's parent to 3.

But we will change 3's parent to 0.

Union $(5,4)$ $\longrightarrow$

~~0~~5	1	0	0	3	5	6	7	8
0	1	2	3	4	5	6	7	8

Here, For, 5 parent is 5 $(5 \rightarrow 5)$

For, 4 parent is 0 $(4 \rightarrow 3 \rightarrow 0)$

We will keep 4's parent to 3

But we will change 0's parent to 5

Now, check if (2,5) are connected?

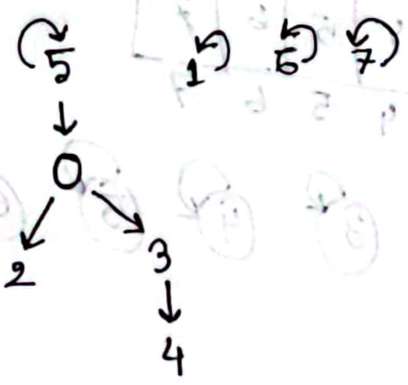
For that we will check both of their root parent. if that is same means they are connected.

For 2 → $(2 \rightarrow 0 \rightarrow 5)$: 5 is root parent

For, 5 → $(5 \rightarrow 5)$ = 5 is root parent

So, they are connected.

So, After connecting some nodes. Our tree will look like this



But for large number of input, the time complexity of this method will increase. So we have to think for an efficient approach because the larger the height of tree goes, it will take more time to find the root parent.

For optimizing the previous Union find Algorithm, we will use Disjoint sets.

Let's rethink the algorithm again →

[illegible] Disjoint set

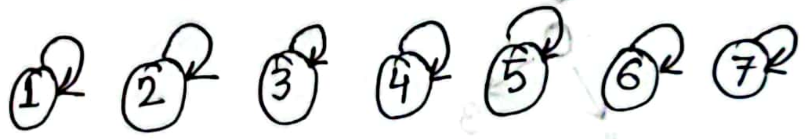
Rank →

0	0	0	0	0	0	0
1	2	3	4	5	6	7

Parent list →

1	2	3	4	5	6	7
1	2	3	4	5	6	7

Current Tree view →



Pseudocode of Union (a,b)

```
1) find ultimate parent of a, b ; Pa, Pb
                                  |   └→ Ultimate parent b
                                  └→ Ultimate parent a

2) find rank of Pa and Pb

3) Connect smaller rank to Larger Rank
   In case of equal rank, connect anyone
      to anyone.
```

Union (1,2)

→ Check P(1) → 1, check P(2) → 2

→ Now check R(1) → 0 and check R(2) = 0

As they have same rank, so you can connect anyone to anyone.

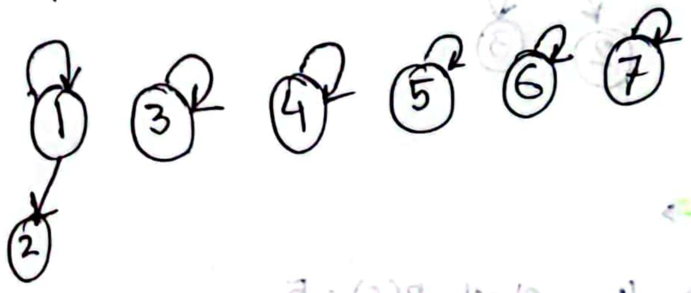
Disjoint set

~~Rank~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
	1	2	3	4	5	6	7

Rank	1	0	0	0	0	0	0
	1	2	3	4	5	6	7

Parent	1	1	3	4	5	6	7
	1	2	3	4	5	6	7

Current Tree



Union (2,3) →

check P(2) = 1 , check P(3) = 3

Check R(1) = 1 , Check R(3) = 0

connect 3 to 1 (smaller rank to Larger Rank)

Disjoint SET :

Rank

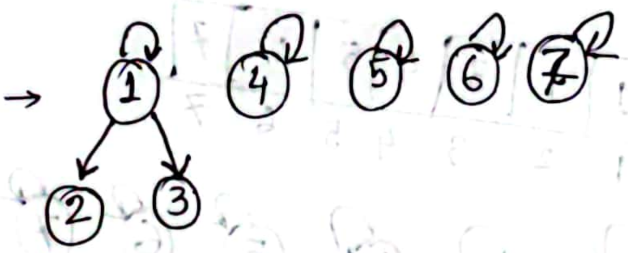
1	0	0	0	0	0	0
1	2	3	4	5	6	7

parent

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
1	2	3				

1	1	1	4	5	6	7
1	2	3	4	5	6	7

Current Tree →



Union (4,5) →

check P(4) = 4 , check P(5) = 5

Check R(4) = 0 , check R(5) = 0

So connect anyone to anyone

Disjoint set

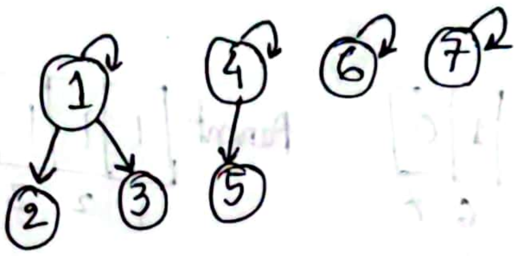
Rank

1	0	0	1	0	0	0
1	2	3	4	5	6	7

Parent

1	1	1	4	4	6	7
1	2	3	4	5	6	7

Current Tree →



Union (6,7) :

Same like previous method.

Disjoint set

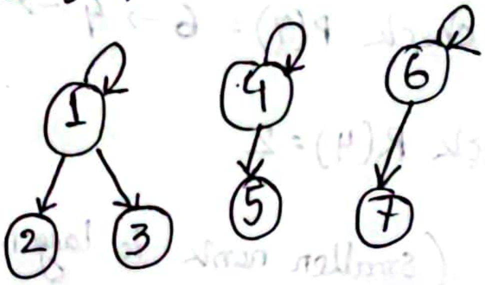
Rank

1	0	0	1	0	1	0
1	2	3	4	5	6	7

Parent

1	1	1	4	4	6	6
1	2	3	4	5	6	7

Tree →



Union (5,6)

Check P(5) = 4 → 4 , check P(6) = 6

Check R(4) = 1, check R(6) = 1

Connect anyone to anyone

Disjoint SET

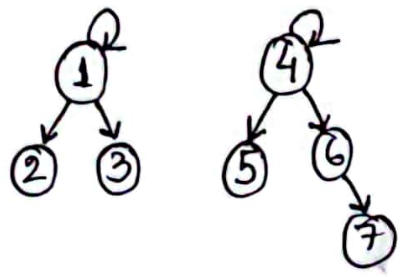
Rank

1	0	0	2	0	1	0
1	2	3	4	5	6	7

Parent

1	1	1	4	4	4	6
1	2	3	4	5	6	7

Tree →



Union (3,7):

Check P(3) = 1 → 1 , check P(7) = 6 → 4 → 4

Check R(1) = 1 , check R(4) = 2

Connect 1 to 4 (smaller rank to larger rank)

Disjoint SET

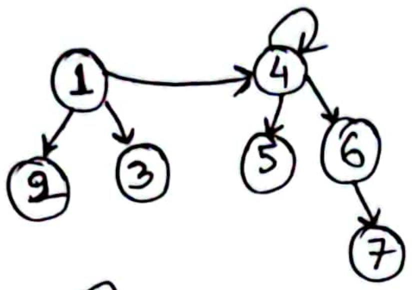
Rank

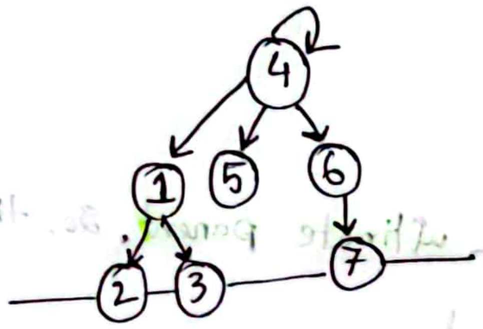
1	0	0	2	0	1	0
1	2	3	4	5	6	7

Parent

4	1	1	4	4	4	6
1	2	3	4	5	6	7

Tree →





Find (2,7) →

check P(2) = 1 → 4 → 4

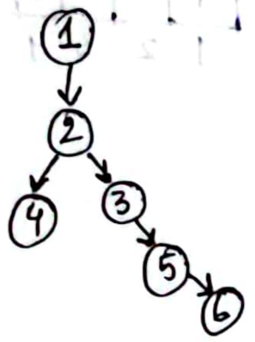
Check P(7) = 6 → 4 → 4

As both of their ultimate parent is same, they are connected

if ultimate parent is not same, then they won't be connected.

For finding any connected node, searching will take log (n) time when we will use Rank. If we want to acheive the find operation in Constant time, we have to use path compression approach.

Let us see a scenario,



if we see find (1,6) →

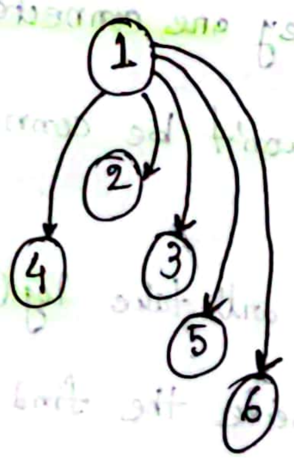
P(1) = 1 → 1

P(6) = 5 ⟶ 3 → 2 → 1

At the end we are camparing the ultimate parent. So, the othore parents are not that importan.

If we can store the parent value with ultimate paarent, then we can see the connection in constant time. O(4α)

↳ Time complexity



Remember that we have keep track of rank instead of height of a tree

Because after perfoming path compression tree height would shrink.

But For rank the values tracked, will be same.