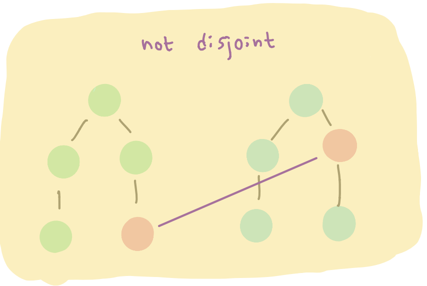
{disjoint}

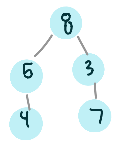
• no elements in common



• union() (set union): Combine two sets into one

• array v.s linked list: use array if you know the amount of sets beforehand

• find(): find element is in which set (returns parent)

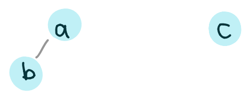


find (4) = 8
find (8) = 8

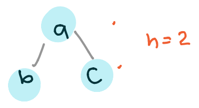
→

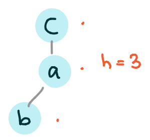
```
if (find(4) == find(2)) - different sets
    cout << "not disjoint";
else cout << "disjoint";
```

• Some union shit:



do:
a) union operation with a as parent
b) union operation with c as parent.

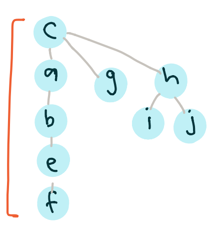
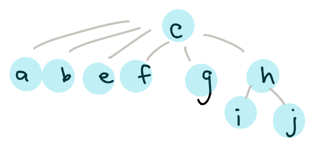
a) 

b) 

because of this structure, it will take longer to find b so this can be slower.

• union by ________:
  • size: larger tree = parent
  • height: tree w/ larger height = parent = better in every case!!
  • height: tree w/ "estimated" larger height = parent
    ◦ estimated being updating height after every union

• path compression

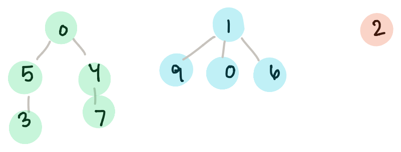
 =  = find on operations are faster

```
Given tree:

               0
           /   |    \
          1    2     3
        / |  \
      4   5    6
    / | \
   7  8  9

find(7)
    7 -> 4 -> 1 -> 0

as we go, make each point to the root

new tree:

          0
  /   /   |    \   \
  7   4   1     2   3
     / \  / \
     8 9  5 6
```

another example

• def: reset all nodes touched to root during a find operation
• why: minimize height of tree.

- Array implementation



= parent

0	1	2	3	4	5	6	7	8	9
1	-1	-1	8	5	8	1	3	-1	1

- -1 defines parent node.
- does not store value, only indexes.

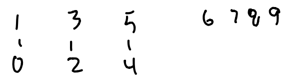
- union by rank

	0	1	2	3	4	5	6	7	8	9
links	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ranks	1	1	1	1	1	1	1	1	1	1

```
setUnion(int r1, int r2)
    //range check

    if (setFind(r1) != -1)
        r1 = setFind(r1);
    if (setFind(R2) != -1
        r2 = setFind(r2);
    if (ranks[r1] <= ranks[r2])
        p = r2      parent
        c = r1;     child
    else
        p = r1;
        c = r2;
    links[c] = parent;
    if (ranks[r1] == ranks[r2])
        ranks[p]++;
```

ASS 9 SHIT:

rank: rough estimate of height



	0	1	2	3	4	5	6	7	8	9
links	1	-1	3	-1	5	-1	-1	-1	-1	-1

-1 doesn't have parent

```
setfind (m, x)
    // if x out of range
    // return -1

    if (links[x] < 0)
        return x
    else
        return links[x] =
                  setfind (links[x])
```

$\underbrace{\text{setfind (links[x])}}_{\text{Compresses}}$

need to include path compression