USA Tutor

Disjoint Sets Union

Slideshow by David Yang and Nguyen Xuan Tung(Neko_nyaa)

What is DSU?

- Disjoint Sets Union, or Union Find
- Used for Connected Component problems, to check for connectivity
- You can do DFS, but DSU is a very clean implementation
- It uses recursion (call yourself)

We need to support 2 operations
Union(node1, node2)
Find(node)

-> Start off the DSU by initializing all of the elements with a[i]=i
This means that the current node is its own parent.

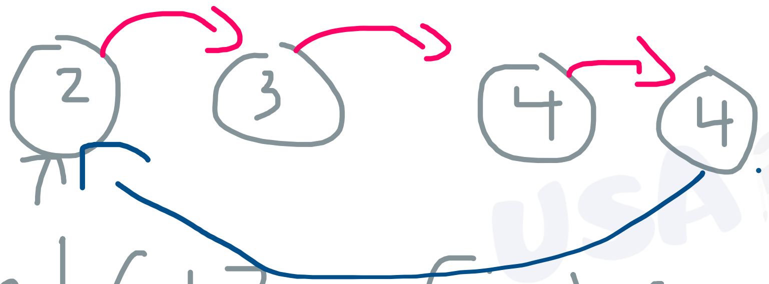
Initializing the DSU

- How do we start it off? Assign a[i]= i.

```
public djset(int n) {
    parent = new int[n];
    N = n;
    for (int i = 0; i < n; i++)
        parent[i] = i;
}
```

Find Operation

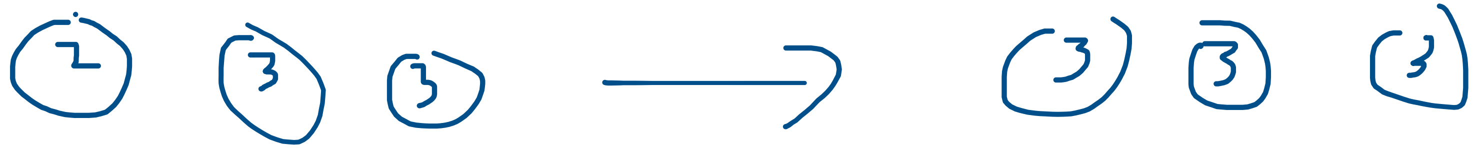
- Find(a) = returns the parent of a



find(1) = find(2) = find(3) = find(4) = $\boxed{4}$

How to Implement Find?

- We recursively call find until we reach the parent.
- We know the parent of the current node(a[i]) is i. If this is satisfied, then we can return our current node.
- To save some runtime for future queries, we point our current node to the result of our recursive statement.



$f(1) \Rightarrow f(2) \Rightarrow f(3)$

$a[1] = a[2] = \swarrow$

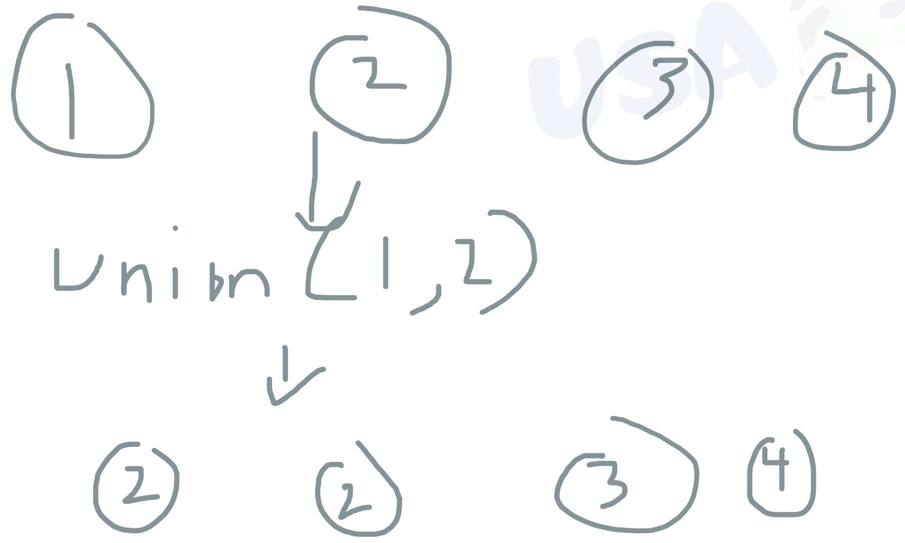
Code Snippet for Find

- The FULL code will also be posted on the LMS

```
public int find(int v) {

   // I am the root of the tree
   if (parent[v] == v) return v;

   // Find my parent's root.
   int res = find(parent[v]);

   // Attach me directly to my previous root.
   parent[v] = res;
   return res;
}
```

The Union Operation

- Union(A,B) = Set the parent of A to the parent of B
- Runtime is in Log2 (N) time



How to Implement Union?

- Call find for A and B
- Let pA = find(A), and pB = find(B)
- We can just set our current node to the parent of B.
- parent[pA] = pB or parent[pB]= pA (no difference)

```
public void union(int v1, int v2) {

    // Find respective roots.
    int rootv1 = find(v1);
    int rootv2 = find(v2);

    parent[rootv2] = rootv1;
}
```

Check if two nodes are in the same CC

```
• public boolean sameSet(int v1, int v2) {

      // Find respective roots.
      int rootv1 = find(v1);
      int rootv2 = find(v2);

      return rootv1 == rootv2;
  }
```

USACO 2016 US Open Closing the Farm

- Idea: Instead of starting with full graph and remove nodes, we start with empty graph, and add node.

Sample test case:

1 2

2 3

3 4

Removal order: 3 4 1 2

--> Add order: 2 1 4 3 (reverse removal)

USACO 2016 US Open Closing the Farm

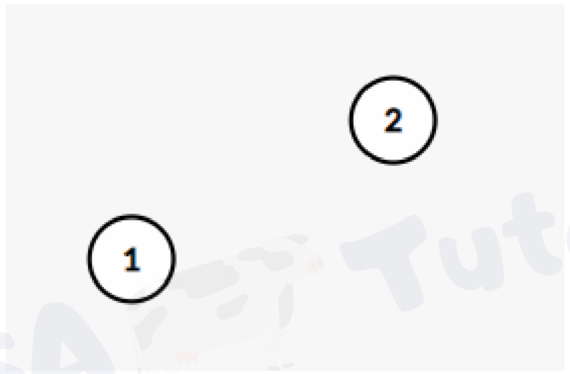
- Each time we add a node, number of CCs increase by one
- Each time we successfully unite two nodes (i.e. it is an actual union, meaning they don't have the same root to begin with), then the number of CCs decrease by one

--> YES if the number of CCs is exactly one.

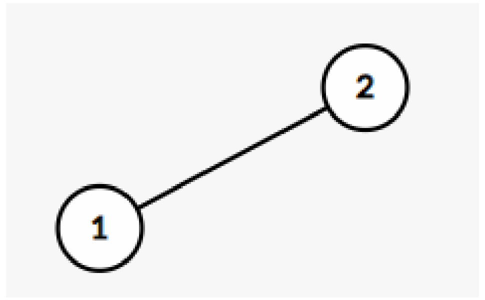
Add 1

Add 2

- Now we have two nodes --> Two CCs

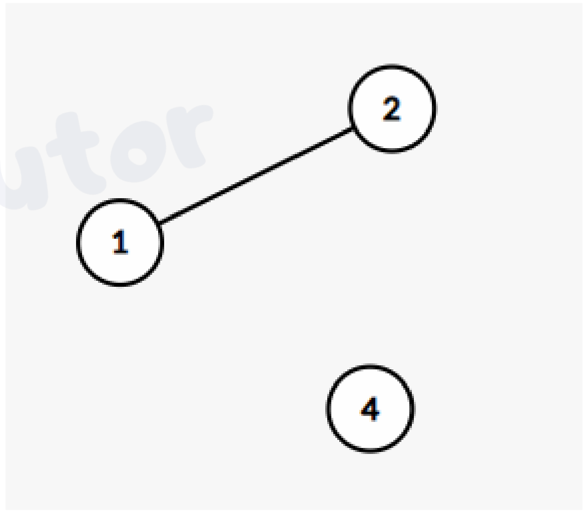


- But we also a new edge (1, 2) --> do union(1, 2)
- Union successful --> One CC



Add 4

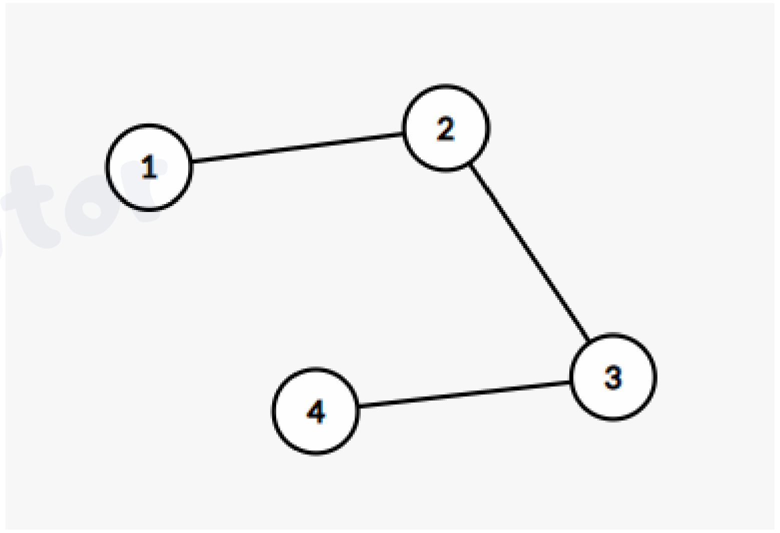
- No new edge --> nothing gets united
- Two CCs



Add 3

- New node --> 3 CCs
- Two new edges (2, 3), (3, 4) --> Union(2, 3) and union(3, 4)

--> Both unions successful, number of CCs reduce by 2 --> One CC



Questions?