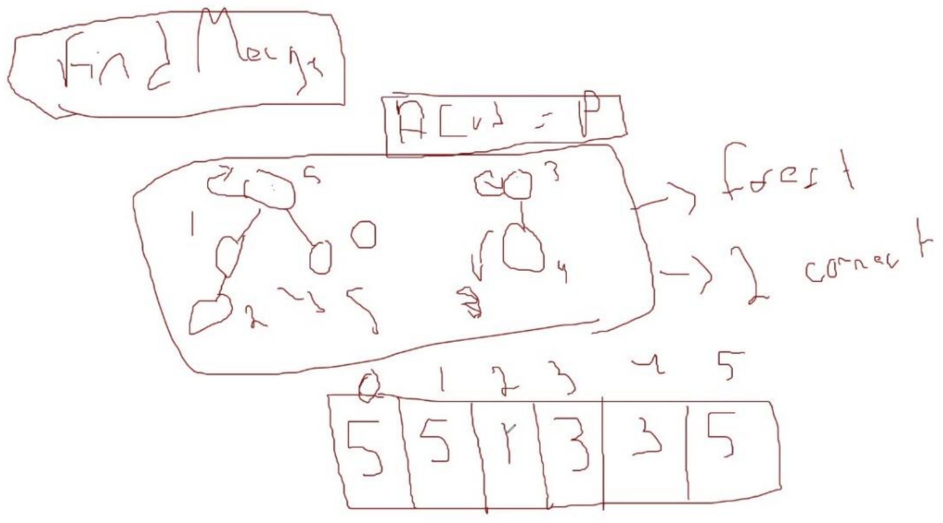
Session 1 DSU

Started session with brief introduction about competitive programming

Set:
0-1-2-3-4….n-1
Take each number alone as a set
{0},{1},{2},….
ID of each set goes 0,1,2…n-1



DSU Operations:
-Find: Find 4 will return ID of set having value searched

-Merge: Merge 0 and 2 => new set {0,2} (O(logn))

Representation:
Sets will be represented as trees. Trees will form a forest of several connected components.

Tree is an array where each mode holds the value of the root so when we find it returns the ID.

If we want to merge, change the parent of one of them to the parent of the other. So one will remain parent and the other will become a node in the tree. The parent of a tree with a larger size will remain the parent of the new tree.

Size tells us the number of nodes in a tree.

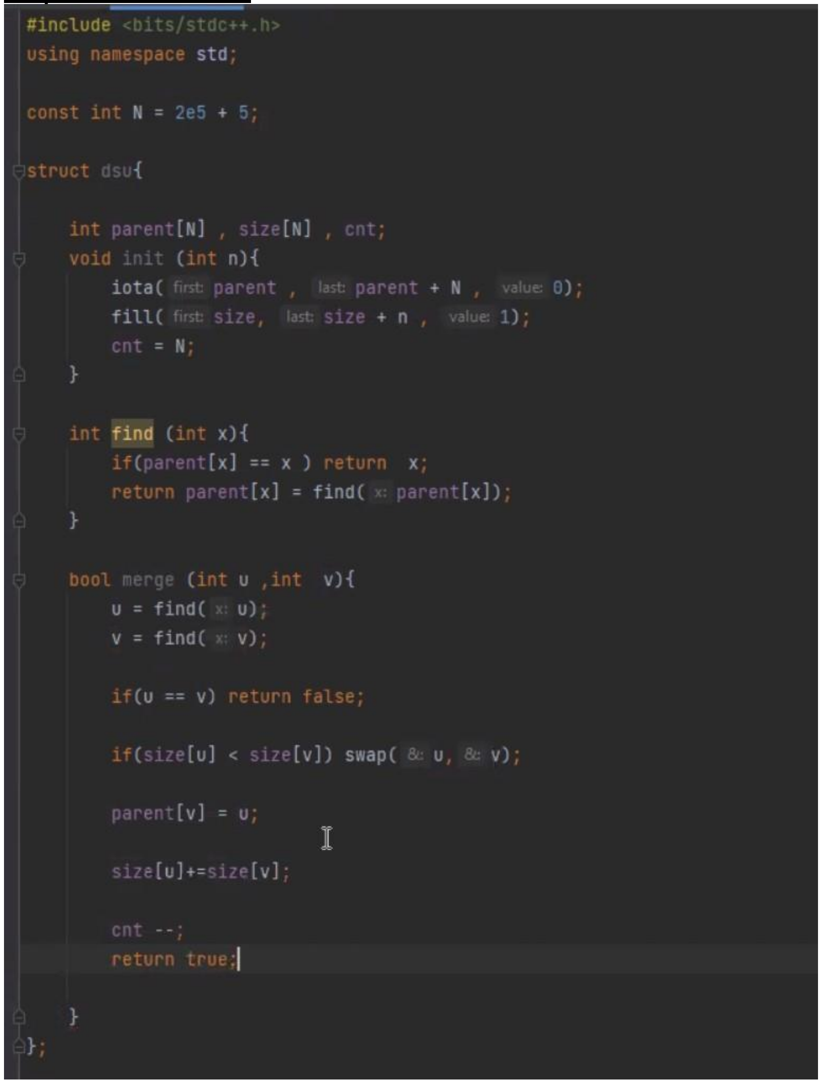
Root of the tree will be the ID of this set.

Find takes linear time but we can do something better called path compression.

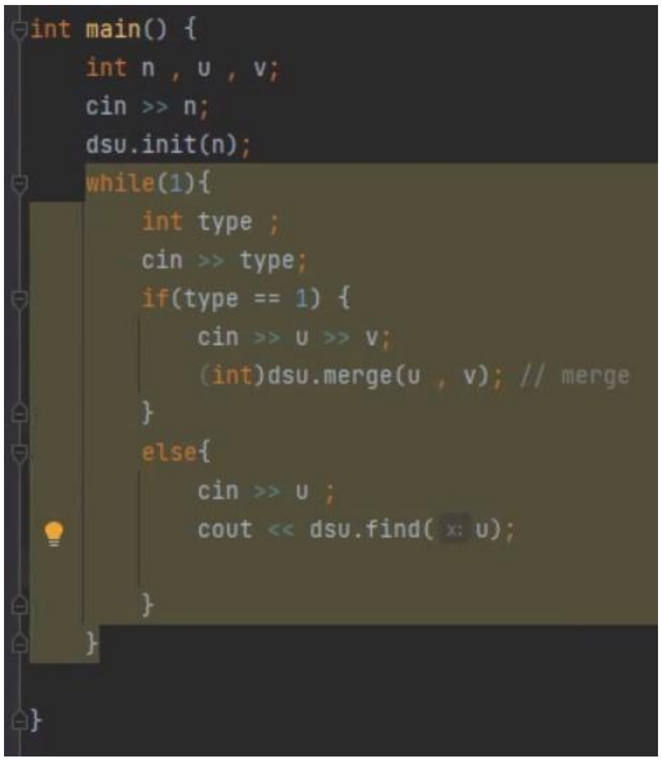
Naive way if parent x=x return x
else return find(parent(x))

Path compression won’t need more than 10 steps for 10^18 since it runs in an inverse log.

<u>Implementation:</u>



```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 5;

struct dsu{

    int parent[N] , size[N] , cnt;
    void init (int n){
        iota( first: parent ,  last: parent + N ,  value: 0);
        fill( first: size,  last: size + n ,  value: 1);
        cnt = N;
    }

    int find (int x){
        if(parent[x] == x ) return  x;
        return parent[x] = find( x: parent[x]);
    }

    bool merge (int u ,int  v){
        u = find( x: u);
        v = find( x: v);

        if(u == v) return false;

        if(size[u] < size[v]) swap( &: u, &: v);

        parent[v] = u;

        size[u]+=size[v];

        cnt --;
        return true;

    }
};
```



```
int main() {
    int n , u , v;
    cin >> n;
    dsu.init(n);
    while(1){
        int type ;
        cin >> type;
        if(type == 1) {
            cin >> u >> v;
            (int)dsu.merge(u , v); // merge
        }
        else{
            cin >> u ;
            cout << dsu.find( x: u);

        }
    }

}
```

Notes:
iota can be used instead of loop

We can find the number of connected components by returning the cnt in the class.

Problem:
-https://vjudge.net/contest/536760#problem/U
-https://vjudge.net/contest/536760#problem/X
-https://www.hackerearth.com/problem/algorithm/teachers-dilemma-3-00955296/
-https://assessment.hackerearth.com/challenges/college/codemania-20/algorithm/owl-fight/?