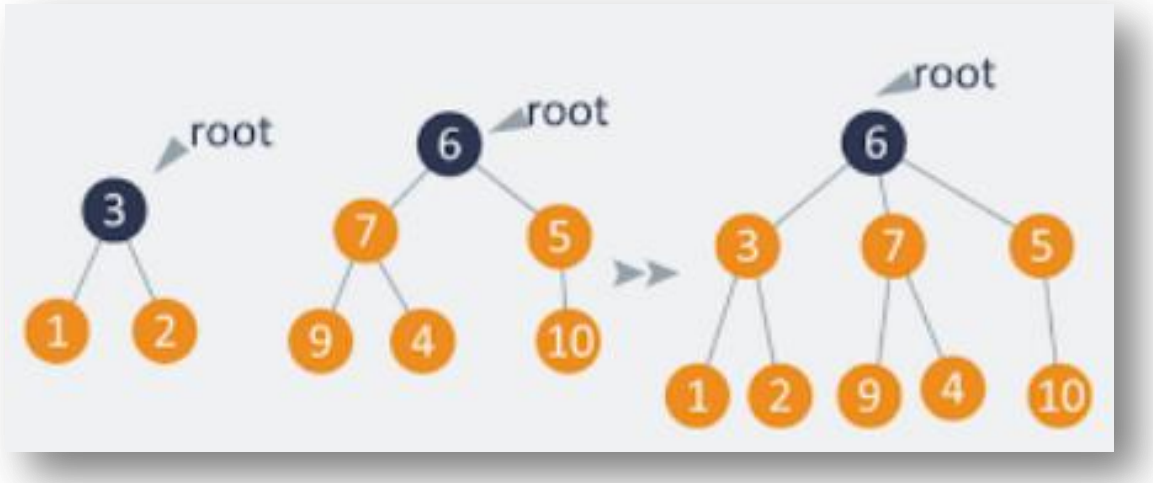
STAR LEAGUE

05 Disjoint Set Union (DSU)



DSU Definition

- A **DSU** (or **disjoint-set data structure)** is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. No item can be in more than one set.
- Operations
  - **Init:** Each element is a separate disjoint set.
  - **Find**: Determine which subset a particular element is in.
  - **Union**: Join two subsets into a single subset

Basic Array Implementation

```
struct DSU
{
    vector<int> parent;
    void init(int n)
    {
        parent.resize(n);
        for (int i = 0; i < n; i++)
            parent[i] = i;
    }

    int find(int u)
    {
        while (u != parent[u])
            u = parent[u];
        return u;
    }

    void unite(int u, int v)
    {
        u = find(u);
        v = find(v);
        if (u != v)
            parent[v] = u;
    }
};
```

```
int main()
{
    DSU dsu;
    dsu.init(10);

    cout << dsu.find(5) <<endl;      //5
    cout << dsu.find(9) << endl;     //9

    dsu.unite(1, 2);
    dsu.unite(2, 3);
    dsu.unite(3, 5);

    cout << dsu.find(5) << endl;     //1
    cout << dsu.find(9) << endl;     //9

    for (int x : dsu.parent)
        cout << x << " ";
    //0 1 1 1 4 1 6 7 8 9
}
```

Implementation with Size and Path Compression

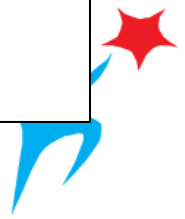
```
struct DSU
{
    vector<int> parent, size;
    int n;
    void init(int s)
    {
        n = s;
        parent.resize(n);
        size.resize(n, 1);
        for (int i = 0; i < n; i++)
            parent[i] = i;
    }
    //Conect each node on the path the the root
    int find(int u)
    {
        if(u == parent[u])
            return u;
        return parent[u] = find(parent[u]);
    }

    /*
    //Iteratively shorten the path in half
    int find(int u)
    {
        while (u != parent[u])
        {
            //connect u to its grand parent
            parent[u] = parent[parent[u]];
            u = parent[u];
        }
        return u;
    }
    */
```

```
    //Connect the smaller set to the larger one
    void unite(int u, int v)
    {
        u = find(u);
        v = find(v);
        if (u == v) //u and v are in the same set
            return;

        if (size[v] > size[u])
            swap(u, v);
        parent[v] = u;
        size[u] += size[v];
    }
};

int main()
{
    DSU dsu;
    dsu.init(10);
    cout << dsu.find(8) << endl;    //8
    cout << dsu.find(9) << endl;    //9

    dsu.unite(2, 8);
    dsu.unite(1, 2);
    dsu.unite(3, 4);
    dsu.unite(5, 7);
    dsu.unite(4, 5);
    dsu.unite(7, 2);

    for (int x : dsu.parent)
        cout << x << " ";           //0 2 3 3 3 3 6 3 2 9
    cout << endl;
    cout << dsu.find(8) << endl;    //3
    cout << dsu.find(9) << endl;    //9

    for (int x : dsu.parent)
        cout << x << " ";           //0 2 3 3 3 3 6 3 3 9
    cout << endl;
    for (int x : dsu.size)
        cout << x << " ";           //1 1 3 7 1 2 1 1 1 1
}
```

Exercise 1: DSU with Rank

Instead of size use the rank of subtrees for union. Attach smaller depth tree under the root of the deeper tree.

```
struct DSU
{
    vector<int> parent, rank;
    int n;
    void init(int s)
    {
        n = s;
        parent.resize(n);
        size.resize(n, 0);
        for (int i = 0; i < n; i++)
            parent[i] = i;
    }
    int find(int u)
    {
        while (u != parent[u])
        {
            //connect u to its grand parent
            parent[u] = parent[parent[u]];
            u = parent[u];
        }
        return u;
    }
    //Connect the smaller rank tree to the larger one
    void unite(int u, int v)
    {
        //Complete this function
    }
};
```



Problem 1: Cycle Detection

Check if a graph contains a cycle or not.

Sample Input

7 5 -> Number of nodes and edges
1 2
3 5
4 5
6 4
3 4

Sample Output

YES

Implementation with Map

```
struct DSU
{
    map<int, int> parent, size;
    void init(vector<int> v)
    {
        for (int x : v)
        {
            parent[x] = x;
            size[x] = 1;
        }
    }

    int find(int u)
    {
        while (u != parent[u])
        {
            //connect u to its grand parent
            parent[u] = parent[parent[u]];
            u = parent[u];
        }
        return u;
    }

    //Connect the smaller set to the larger one
    void unite(int u, int v)
    {
        u = find(u);
        v = find(v);
        if (u == v) //u and v are in the same set
            return;

        if (size[v] > size[u])
            swap(u, v);

        parent[v] = u;
        size[u] += size[v];
    }
};
```

Problem 2: Largest Component

Calculate the size of the largest connected component of a growing graph with N ($1 <= N <= 100,000$) nodes. Node IDs are between -10^9 and 10^9.

At the beginning graph has no edges. The M ($0 <= M <= 100,000$) edges will be added one by one.

Print the size of the largest connected component after adding each edge.

Problem 2: Largest Component

Sample Input

5 6
1 -5 6 20 -10
1 -5
6 -10
20 6
-10 20
1 6
-5 6

Sample Output

1
2
2
3
3
5
5

Problem 3: Chess Teams

Some players from two different chess teams (A and B) playing N (1 <= N <= 100,000) chess games. In each game one of the players is selected from team A and the other player is selected from team B. Each player has a unique ID between 1 and 1000,000,000.

Calculate at most how many players can be in the larger team.

Problem 3: Chess Teams

SAMPLE INPUT

```
6
2 6
10 50
3 4
7 9
50 8
4 10
```

SAMPLE OUTPUT

```
5
```

USACO Problems

- Closing the Farm (2016 US Open Gold)