Graph Theory.

Disjoint Set Union. (DSU)
OR.
Union Find.

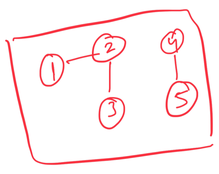
— data structure, to keep track of partition of set into disjoint sets.

0 1 2 3 4 ← Set

{ S1: 0, 1, 4 ; S2: 2, 3 } disjoint sets (non-overlapping).

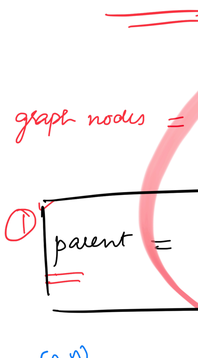
— two main operations —
1. Find (which set a particular element belongs to, done by finding root)
2. Union. (Merges two sets into one)

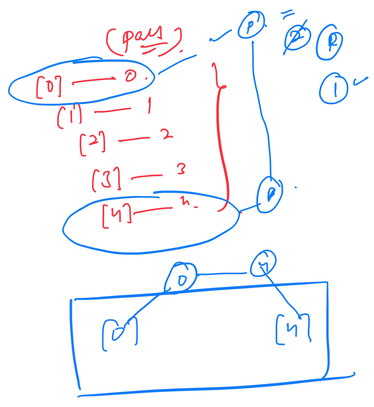
Applications —
1. Detect cycle in graph. undirected graph
2. Find connected components. undirected graph.
3. Kruskal's Algorithm for Minimum Spanning Tree (MST).

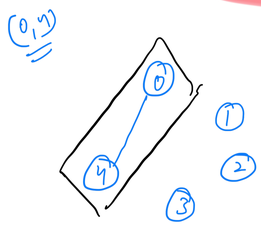


given, graph nodes = [0, 1, 2, 3, 4]

parent = [-1, -1, -1, -1, 0]







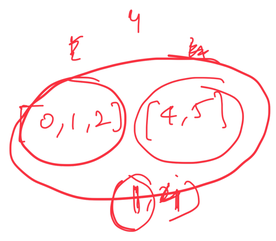
① find(0) → 0 p1
find(4) → 4 p2

② parent[p2] = p1

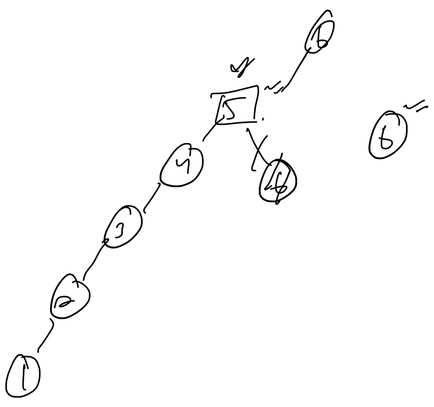
parent = [-1, -1, -1] (0 1 2)

```
int find(x)
{
    if(parent[x] == -1)
        return x;
    return find(parent[x]);
    parent[x] = find(parent[x]);
}
```

f(4) → f(0) → -1 [Subcalls]



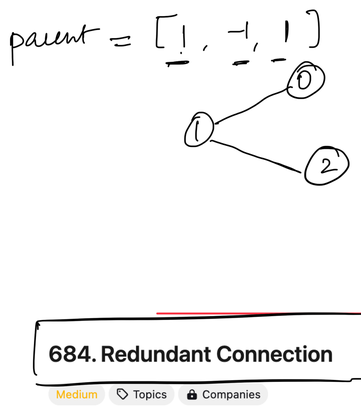
```
void union(int a, int b)
{
    a = find(a);
    b = find(b);
    if(a != b)
        { parent[a] = b; }
}
```



given, n = 3

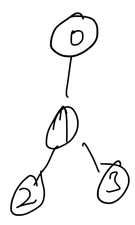
edges = [[0,1], [1,2], [2,0]]

parent = [1, -1, 1] (0 1 2)



loop (cycle)

f(2) → 1
f(0) → 1 (Yes)



0 — n-1.

[n, 1]

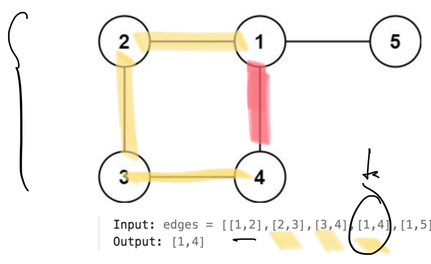
684. Redundant Connection

Medium · Topics · Companies

In this problem, a tree is an **undirected graph** that is connected and has no cycles.

You are given a graph that started as a tree with `n` nodes labeled from `1` to `n`, with one additional edge added. The added edge has two **different** vertices chosen from `1` to `n`, and was not an edge that already existed. The graph is represented as an array `edges` of length `n` where `edges[i] = [a_i, b_i]` indicates that there is an edge between nodes `a_i` and `b_i` in the graph.

Return *an edge that can be removed so that the resulting graph is a tree of `n` nodes*. If there are multiple answers, return the answer that occurs last in the input.

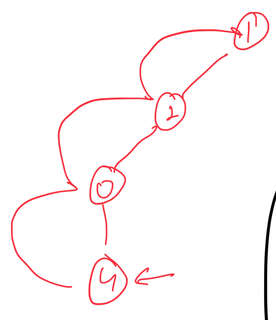


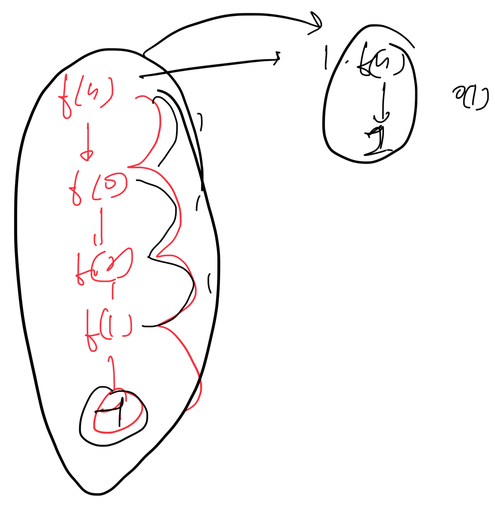
```
Input: edges = [[1,2],[2,3],[3,4],[1,4],[1,5]]
Output: [1,4]
```

f(1) → 2
f(4) → 2

UF.

① Path Compression





T — O(N)
S — O(N)

```
2 usages  new *
static class UnionFind {
    6 usages
    int[] parent;

    1 usage  new *
    public UnionFind(int n) {
        parent = new int[n];
        Arrays.fill(parent, val: -1);        // nc
    }

    new *
    public int find(int x) {
        if (parent[x] == -1) {
            return x;
        }
        return parent[x] = find(parent[x]);
    }

    1 usage  new *
    public boolean union(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b) {
            parent[a] = b;
            return true;
        }
        return false;       // already connected
    }
}
```