Trees & Graphs Lecture 5

Wednesday, 21 August 2024   6:01 AM

Disjoint Sets

{1, 2, 3}, {4, 6, 8}, {5}, {7, 9}

Find   find (6) → {4,6,8}

union   union(3, 5) → {1, 2, 3, 5}, {4, 6, 8}, {7, 9}

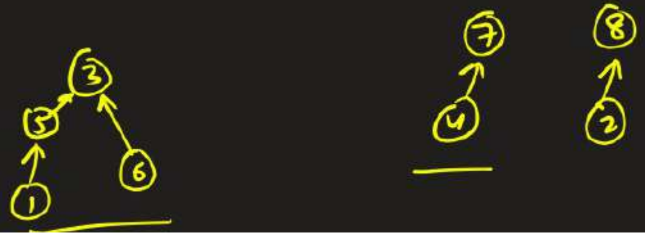
union (1, 5) → {1, 2, 3, 5}, {4, 6, 8}, {7, 9}

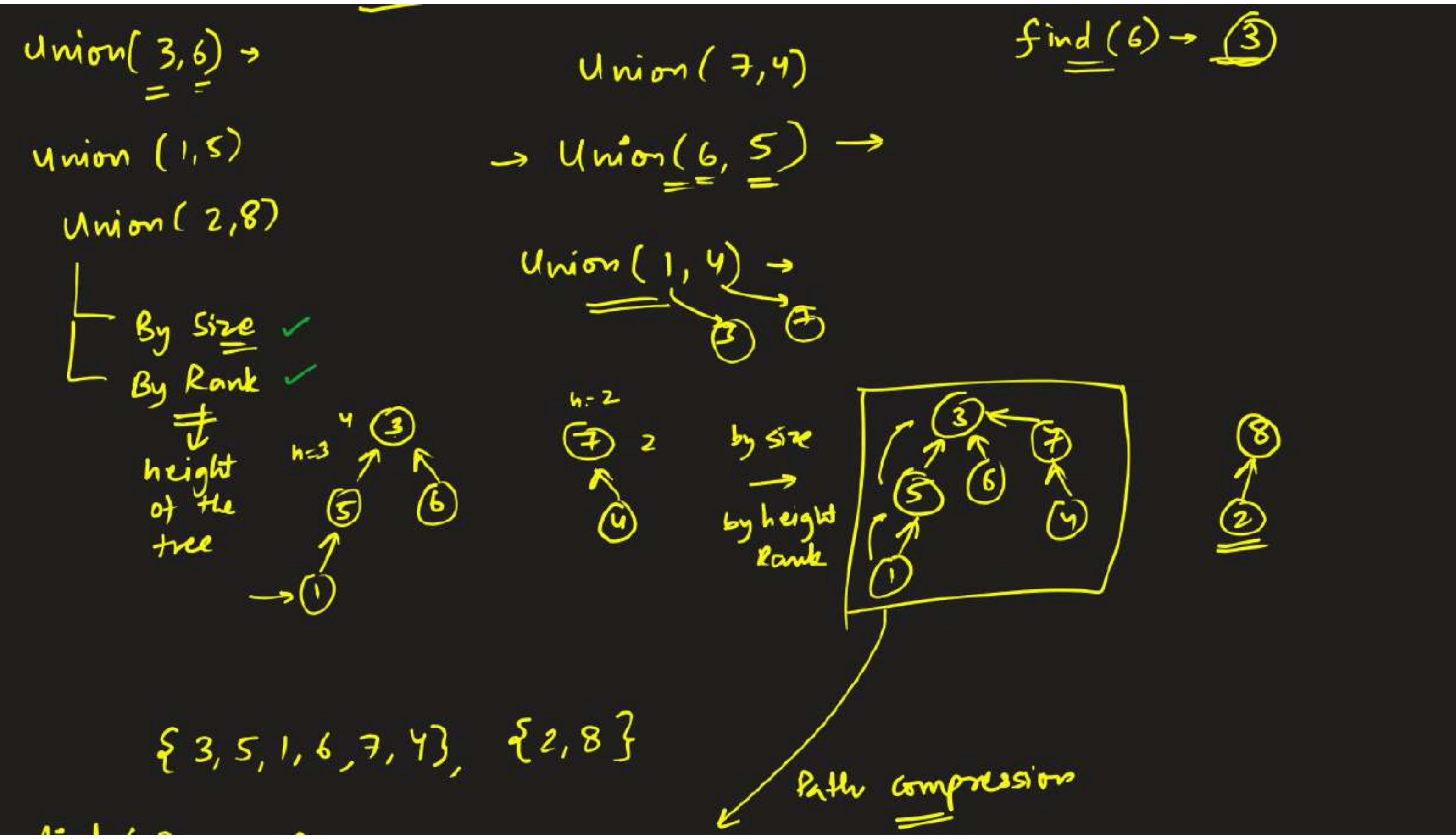
Initial:-   n = 8

{3, 5, 1, 6}, {2, 8}, {7, 4}

Set Representative:-

Root ↑





union(3,6) →

union (1,5)

union(2,8)

- By Size ✓
- By Rank ✓

≠ height of the tree

Union(7,4)

→ Union(6, 5) →

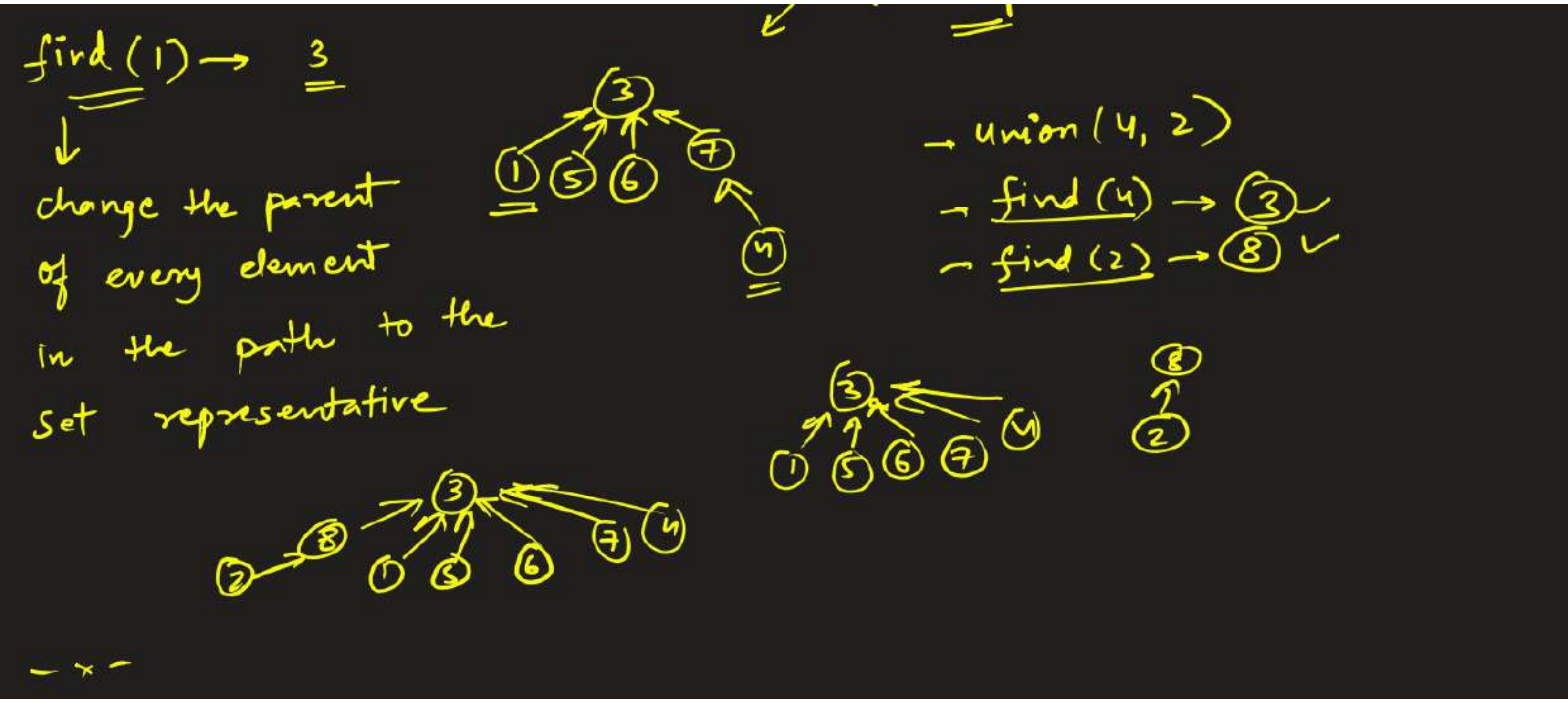
Union(1, 4) →

find (6) → 3

by size → by height Rank

{ 3, 5, 1, 6, 7, 4}, {2, 8}

Path compression



find (1) → 3

↓

change the parent of every element in the path to the set representative

→ union (4, 2)

→ find (4) → 3 ✓

→ find (2) → 8 ✓

— x —

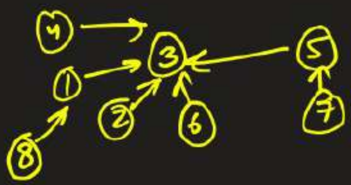
Init :- 1-8

par

1	2	3	4	5	6	7	8
3	3	-3	1	3	3	5	1
		↑			↑		

if the node is representative,
store -rank

otherwise store + parent



		Rank
union(4, 7) →	find(4) → 3	3
	find(7) → 5	2

			- rank
union(3, 6) →	find(3) →	3 →	-1
	find(6) →	6 →	-1

par[6] = 3

par[3] --;

↑

Rank(3)↑

		rank
union(6, 2) →	find(6) → 3	2
	find(2) → 2	1

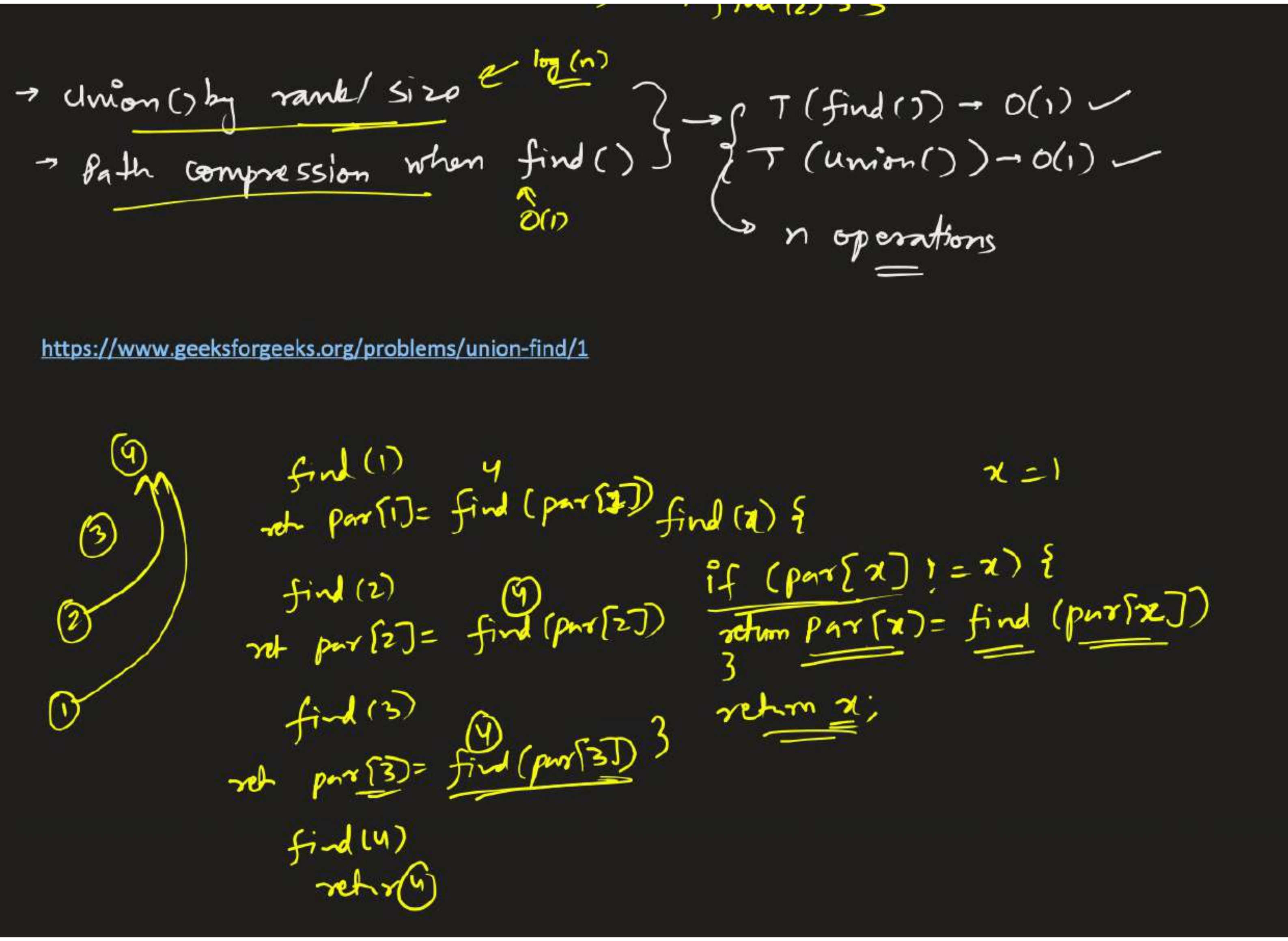
par[2] = 3

union(1, 4)

union(5, 7)

union(4, 8)

		Rank
union(4, 2) →	find(4) → 1	2
	find(2) → 3	2



→ union() by rank/ size ← log(n)

→ Path compression when find()
↑
O(1)

→ T(find()) → O(1) ✓

T(union()) → O(1) ✓

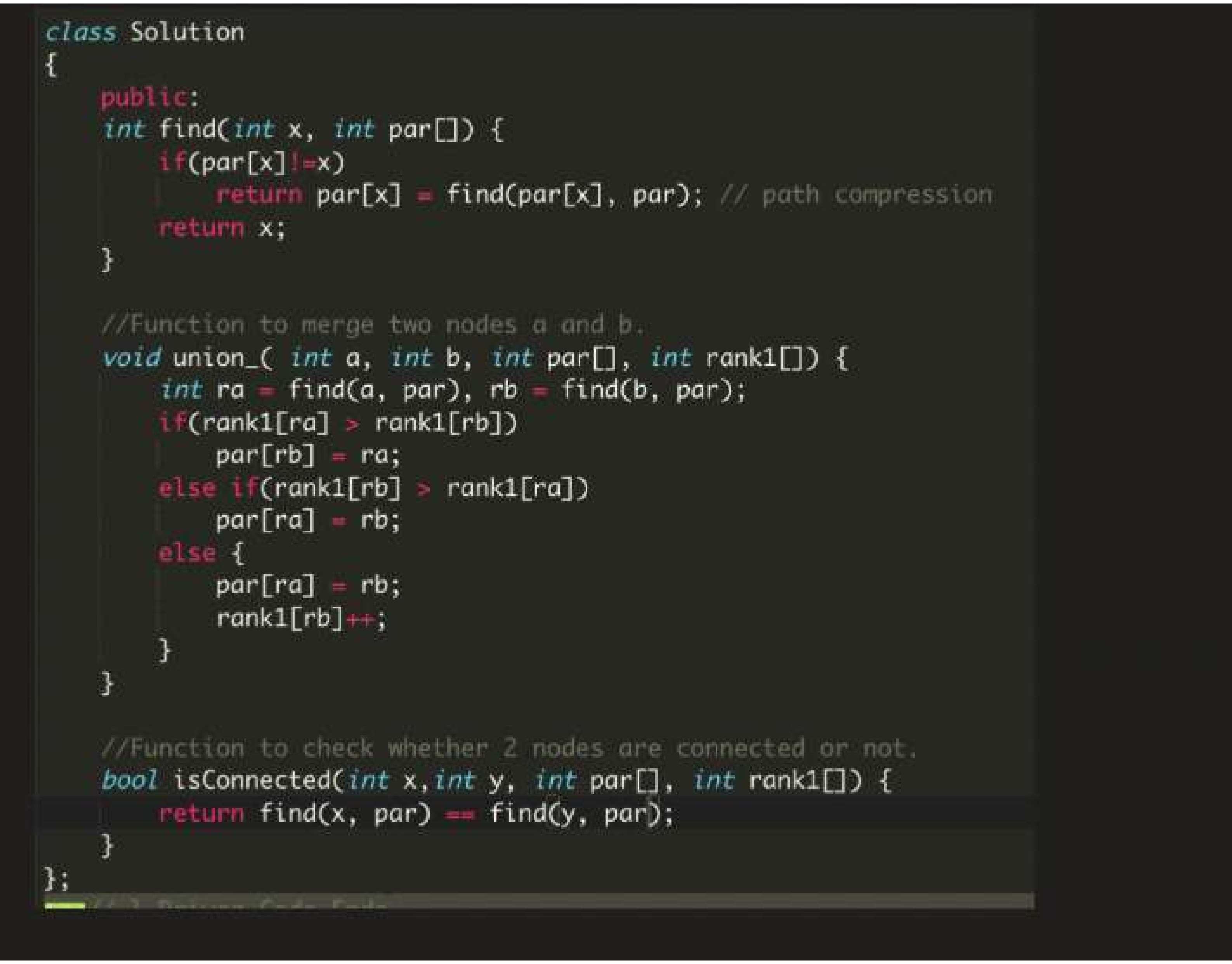
↳ n operations

https://www.geeksforgeeks.org/problems/union-find/1

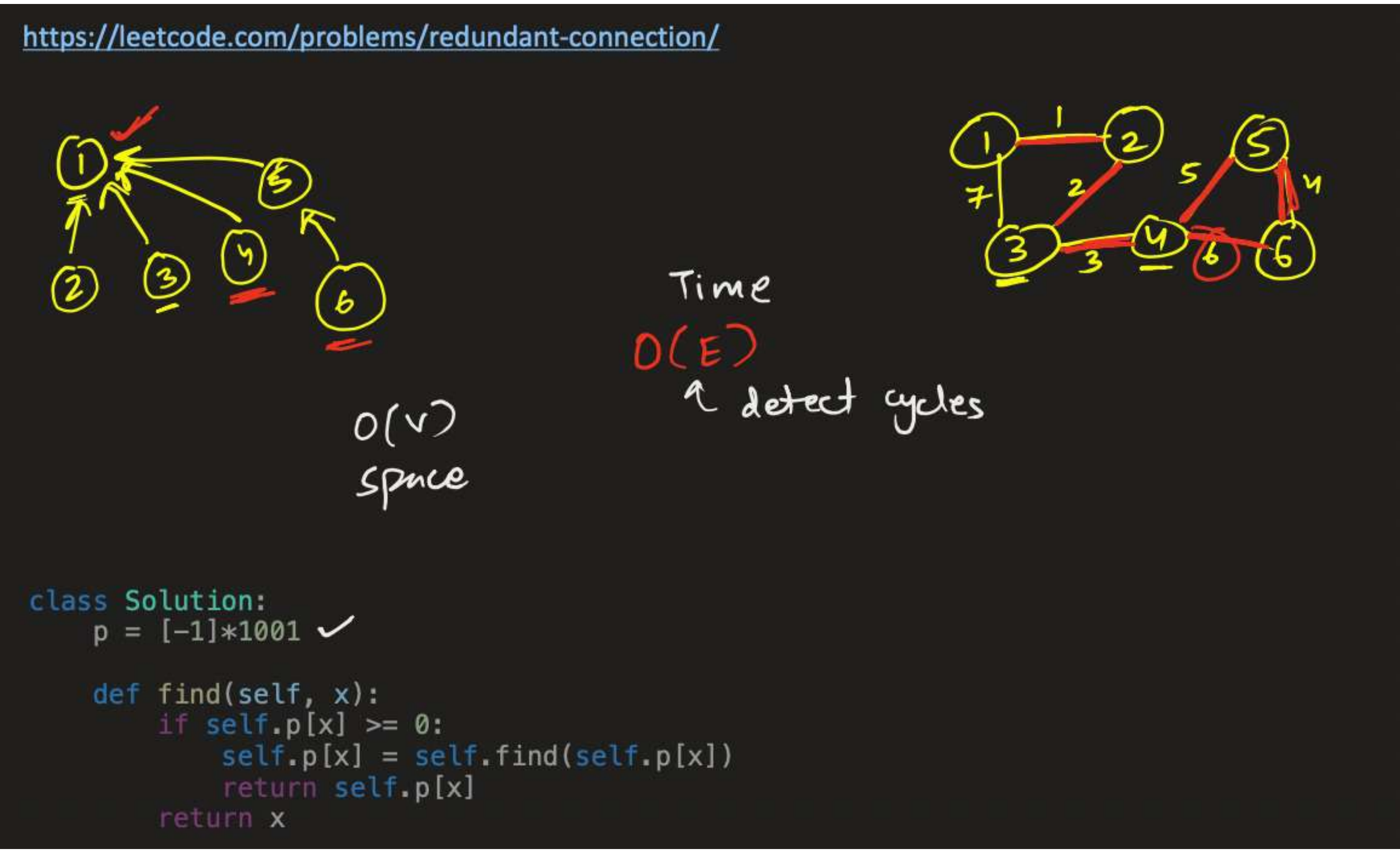
```
find(1)          4
ret par[1]= find(par[1])

find(2)
ret par[2]= (4) find(par[2])

find(3)
ret par[3]= (4) find(par[3]) }

find(4)
return (4)
```

x = 1

```
find(x) {
  if (par[x] != x) {
    return par[x]= find(par[x])
  }
  return x;
```



```
class Solution
{
    public:
    int find(int x, int par[]) {
        if(par[x]!=x)
            return par[x] = find(par[x], par); // path compression
        return x;
    }

    //Function to merge two nodes a and b.
    void union_( int a, int b, int par[], int rank1[]) {
        int ra = find(a, par), rb = find(b, par);
        if(rank1[ra] > rank1[rb])
            par[rb] = ra;
        else if(rank1[rb] > rank1[ra])
            par[ra] = rb;
        else {
            par[ra] = rb;
            rank1[rb]++;
        }
    }

    //Function to check whether 2 nodes are connected or not.
    bool isConnected(int x,int y, int par[], int rank1[]) {
        return find(x, par) == find(y, par);
    }
};
// } Driver Code Ends
```



https://leetcode.com/problems/redundant-connection/

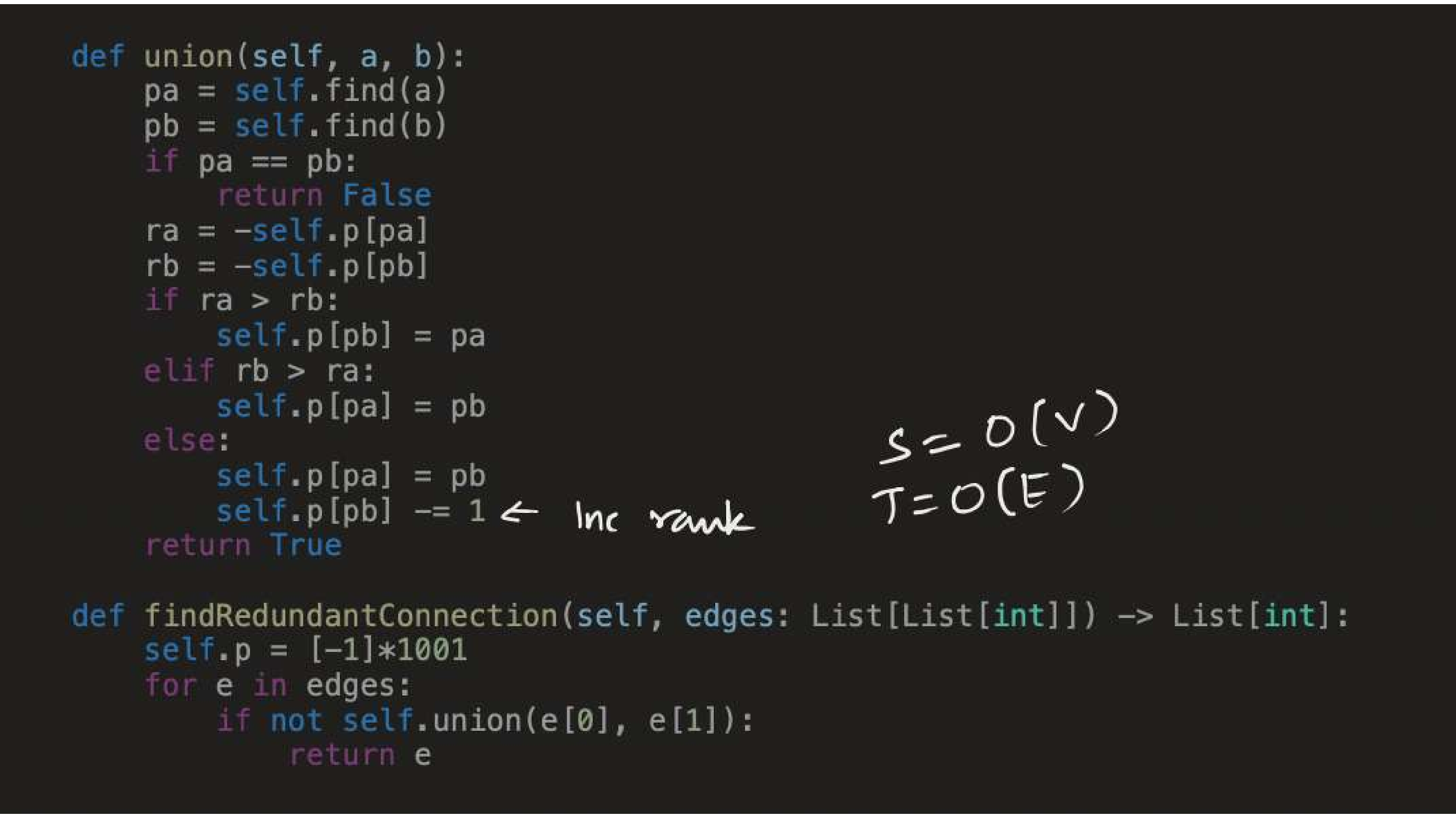
Time

O(E)

↖ detect cycles

O(V)
space

```
class Solution:
    p = [-1]*1001 ✓

    def find(self, x):
        if self.p[x] >= 0:
            self.p[x] = self.find(self.p[x])
            return self.p[x]
        return x
```



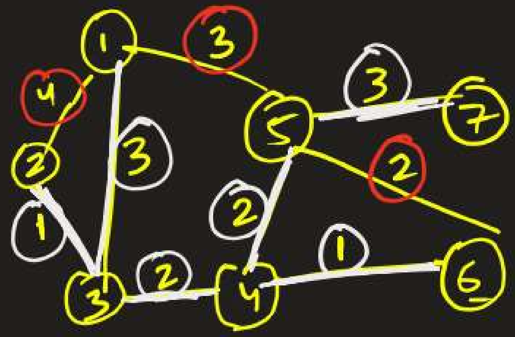
```
def union(self, a, b):
    pa = self.find(a)
    pb = self.find(b)
    if pa == pb:
        return False
    ra = -self.p[pa]
    rb = -self.p[pb]
    if ra > rb:
        self.p[pb] = pa
    elif rb > ra:
        self.p[pa] = pb
    else:
        self.p[pa] = pb
        self.p[pb] -= 1
    return True

def findRedundantConnection(self, edges: List[List[int]]) -> List[int]:
    self.p = [-1]*1001
    for e in edges:
        if not self.union(e[0], e[1]):
            return e
```

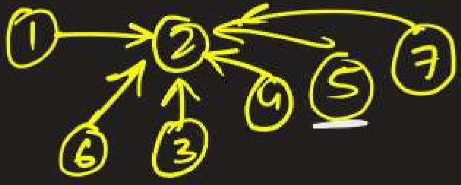
← Inc rank

S = O(V)

T = O(E)

https://www.geeksforgeeks.org/problems/minimum-spanning-tree/1

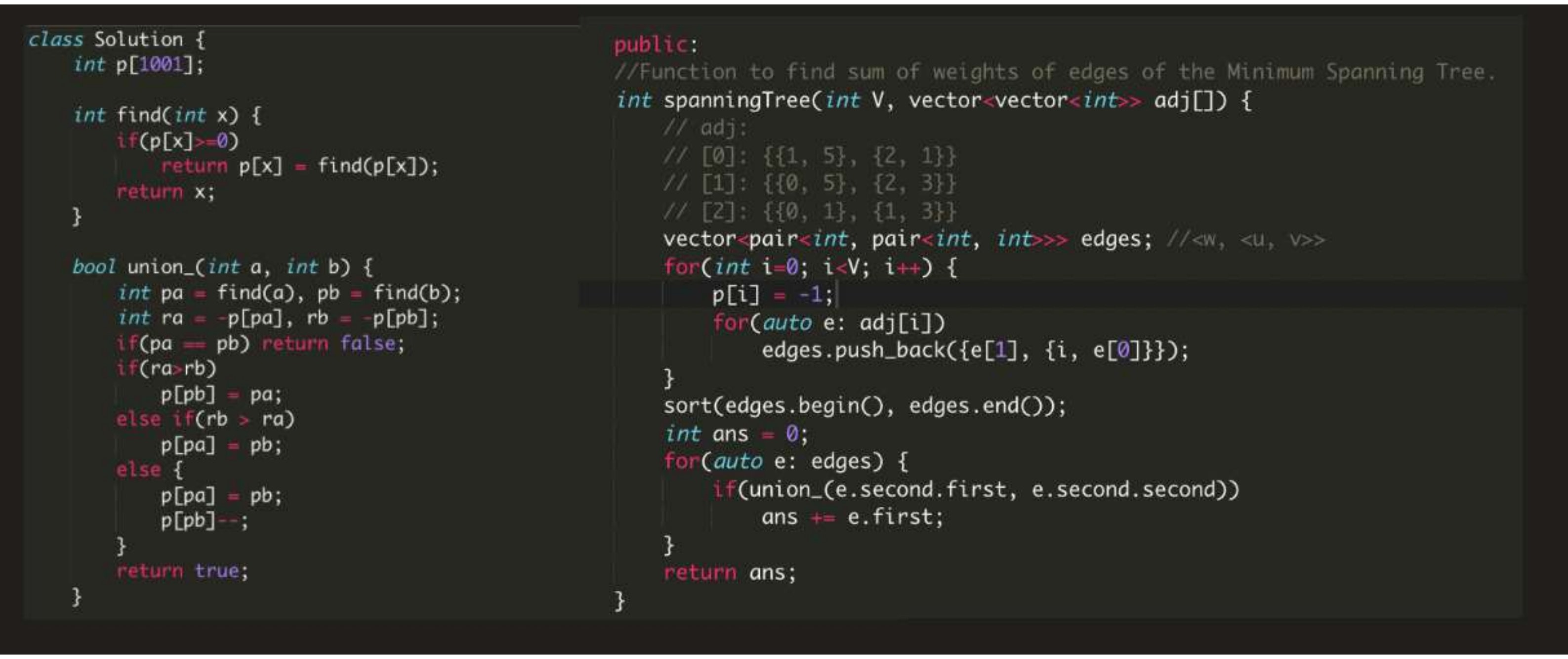




$1+1+2+2+3+3 = \boxed{12}$

$\boxed{E \log E}$

$E \log V$



```cpp
class Solution {
    int p[1001];

    int find(int x) {
        if(p[x]>=0)
            return p[x] = find(p[x]);
        return x;
    }

    bool union_(int a, int b) {
        int pa = find(a), pb = find(b);
        int ra = -p[pa], rb = -p[pb];
        if(pa == pb) return false;
        if(ra>rb)
            p[pb] = pa;
        else if(rb > ra)
            p[pa] = pb;
        else {
            p[pa] = pb;
            p[pb]--;
        }
        return true;
    }
```

```cpp
public:
//Function to find sum of weights of edges of the Minimum Spanning Tree.
int spanningTree(int V, vector<vector<int>> adj[]) {
    // adj:
    // [0]: {{1, 5}, {2, 1}}
    // [1]: {{0, 5}, {2, 3}}
    // [2]: {{0, 1}, {1, 3}}
    vector<pair<int, pair<int, int>>> edges; //<w, <u, v>>
    for(int i=0; i<V; i++) {
        p[i] = -1;
        for(auto e: adj[i])
            edges.push_back({e[1], {i, e[0]}});
    }
    sort(edges.begin(), edges.end());
    int ans = 0;
    for(auto e: edges) {
        if(union_(e.second.first, e.second.second))
            ans += e.first;
    }
    return ans;
}
```