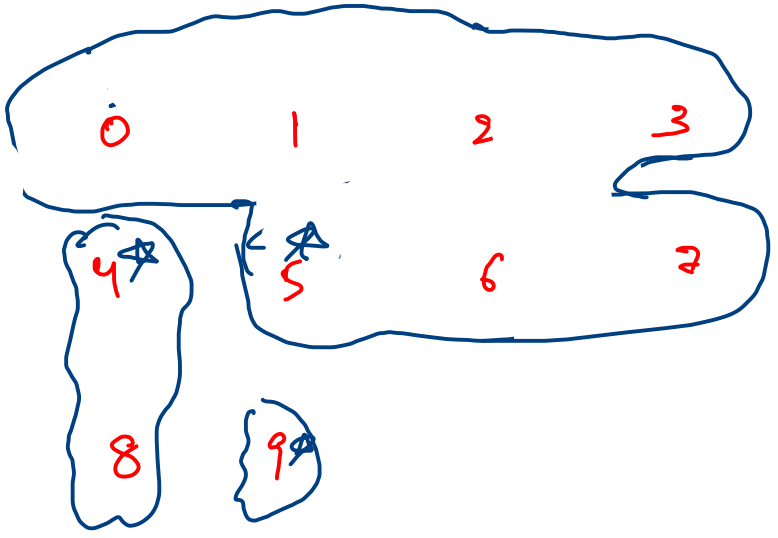
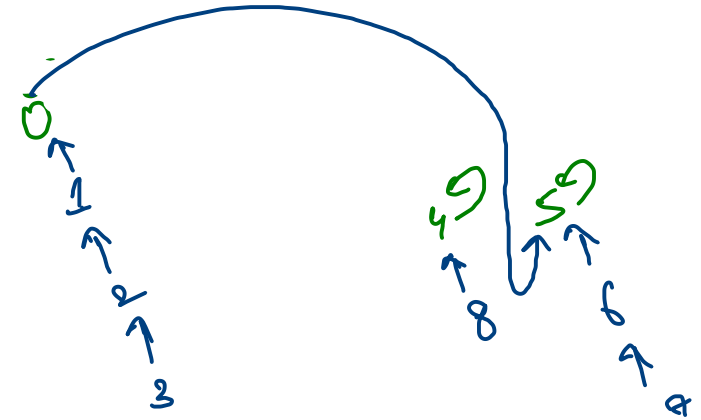




✓ (0,1)

✓ (1,2)

✓ (4,8)

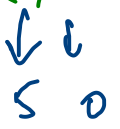
✓ (5,6)

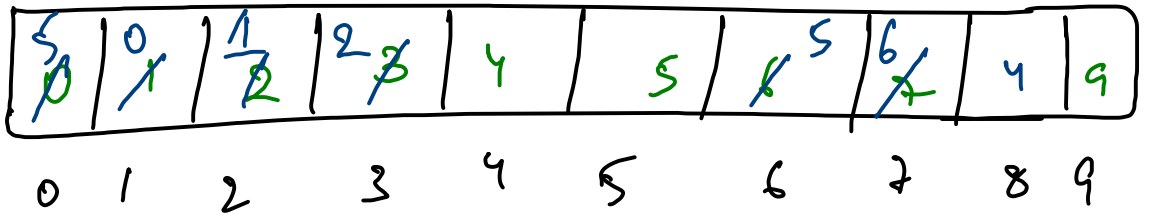
✓ (2,3)

✓ (6,8)

9

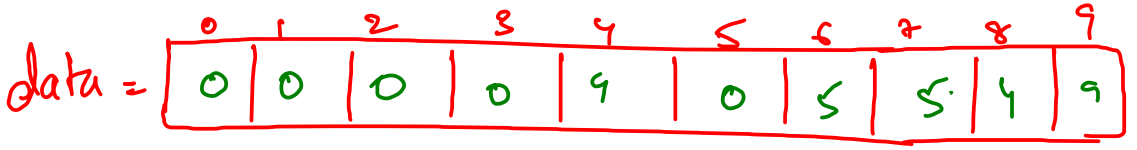
[visualize] (6,1)

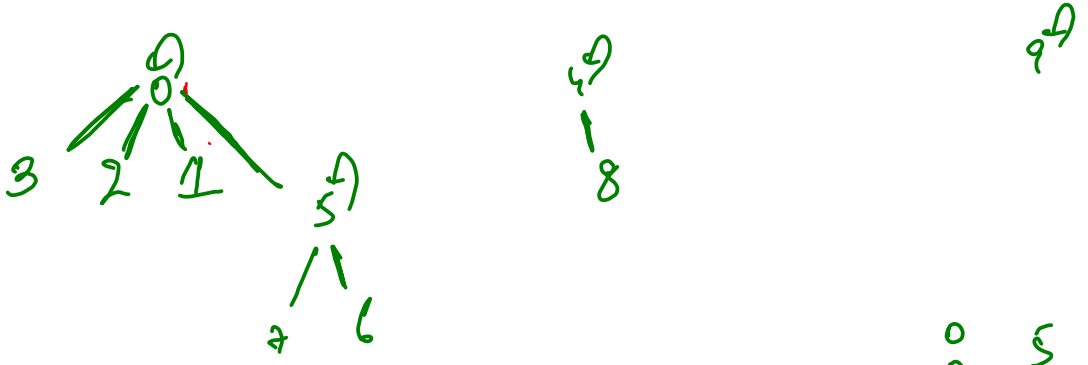




[Parent]

[vertex]

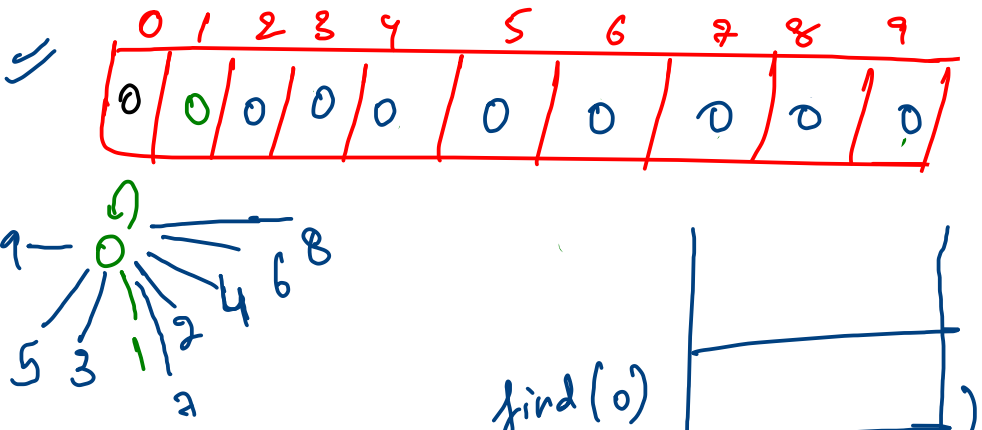


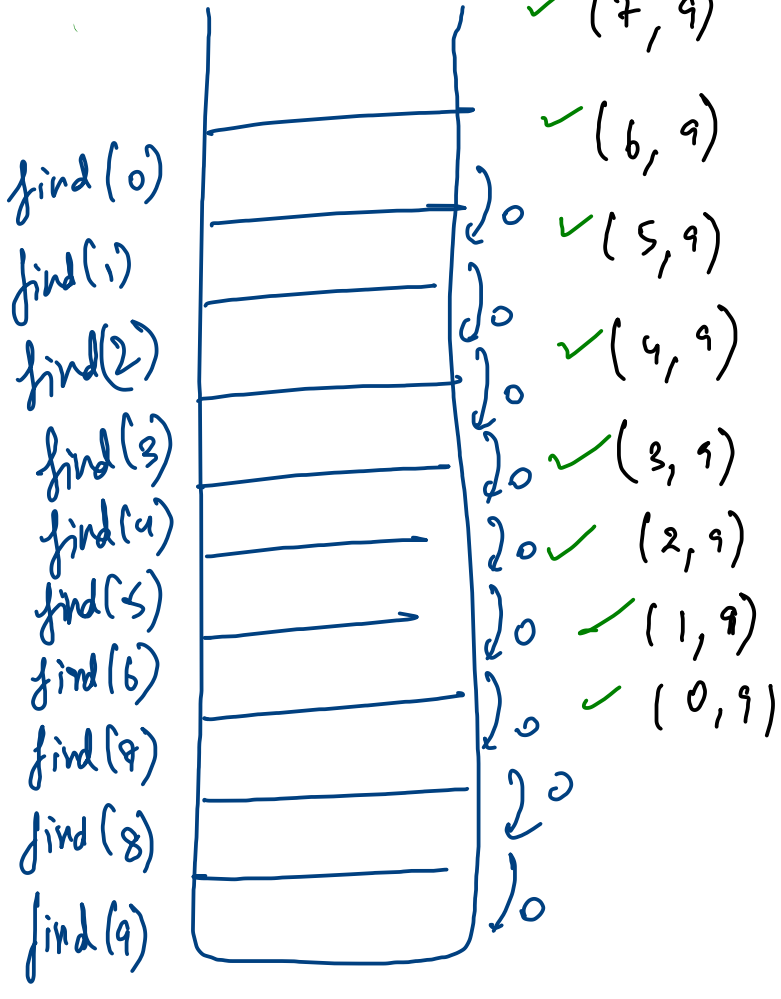



```java
public static void main(String[] args) {
    UnionFind obj = new UnionFind(10);

    obj.union(0, 1);
    obj.union(1, 2);
    obj.union(4, 8);
    obj.union(5, 6);
    obj.union(2, 3);
    obj.union(6, 7);

    System.out.println(obj.isConnected(2, 1));
    System.out.println(obj.isConnected(2, 6));
    System.out.println(obj.isConnected(8, 3));
}
```

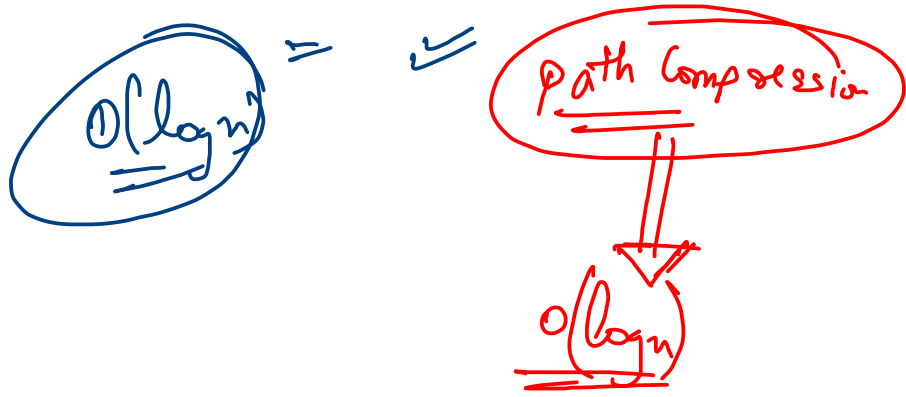
quick Union

```java
public static class UnionFind{
    int []data;

    public UnionFind(int vtces){
        data = new int[vtces];

        for(int i =  0 ; i < vtces ; i++){
            data[i] = i;
        }
    }

    public void union(int vtx1,int vtx2){
        int rootv1 = find(vtx1);
        int rootv2 = find(vtx2);

        if(rootv1 == rootv2){
            // do nothing
        }else{
            data[rootv2] = rootv1;
        }
    }

    public int find(int vtx){ // returns root
        if(data[vtx] == vtx){
            return vtx;
        }

        return find(data[vtx]);
    }

    public boolean isConnected(int v1,int v2){ // part of same comp. or not
        return find(v1) == find(v2);
    }
}
```





(8, 9)
(7, 9)
(6, 9)
(5, 9)
(4, 9)
(3, 9)
(2, 9)
(1, 9)
(0, 9)




```
public int find(int vtx){ // returns root
    if(data[vtx] == vtx){
        return vtx;
    }

    return data[vtx] = find(data[vtx]);
}
```

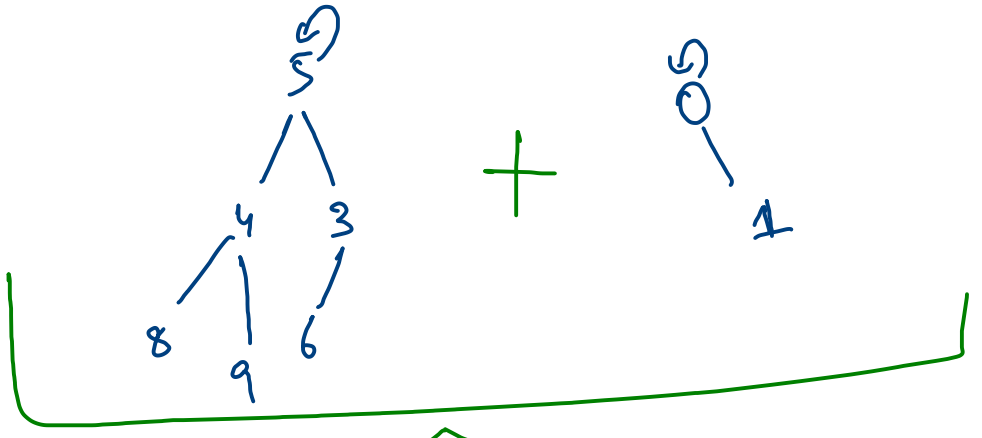
Union by Rank

Edge (3, 1)

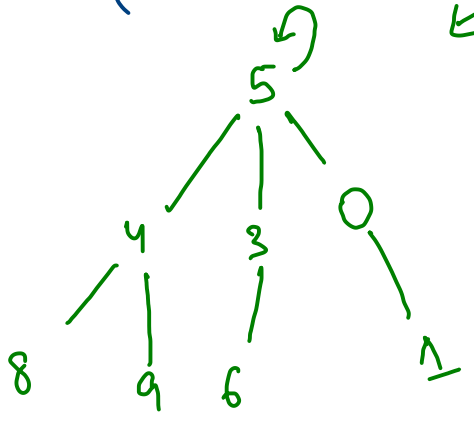
5 0

find

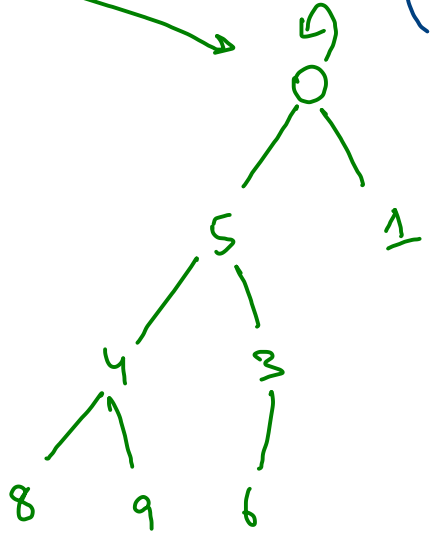
union



(A)



(B)



Par :

0	1	2	3	4	5	6	7	8	9
0	1	3	3	3	4	7	3	8	9

Rank:

0	1	2	3	4	5	6	7	8	9
1	1	1	3	2	1	1	2	1	1

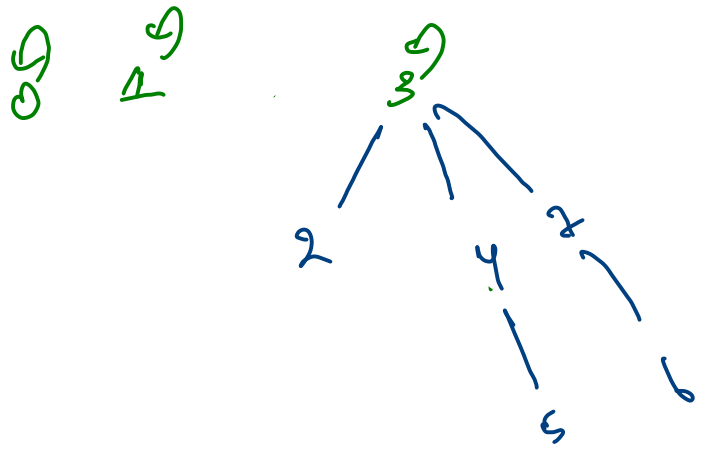
(3, 2) ✓

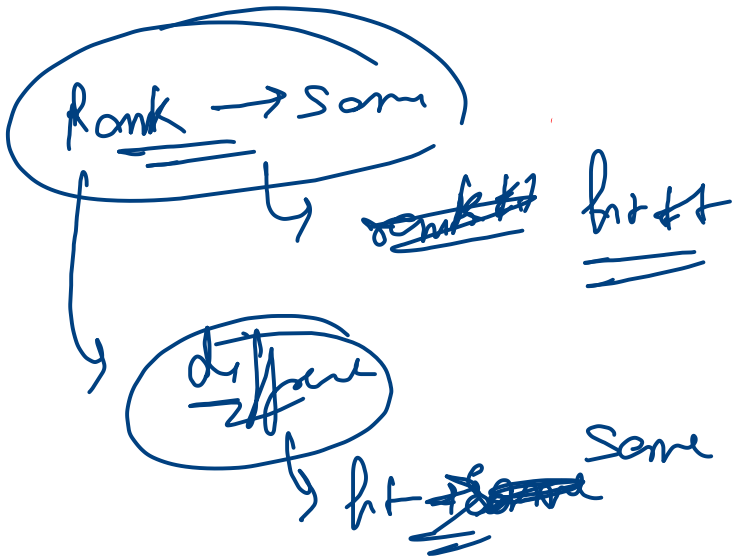
(4, 5) ✓

(2, 5) ✓

(7, 6) ✓

(5, 6) ✓





Union find ⇒ Brute Force

Time Complexity → O(N

Only Path Compression ⟶ Time Complexity → O(log n)

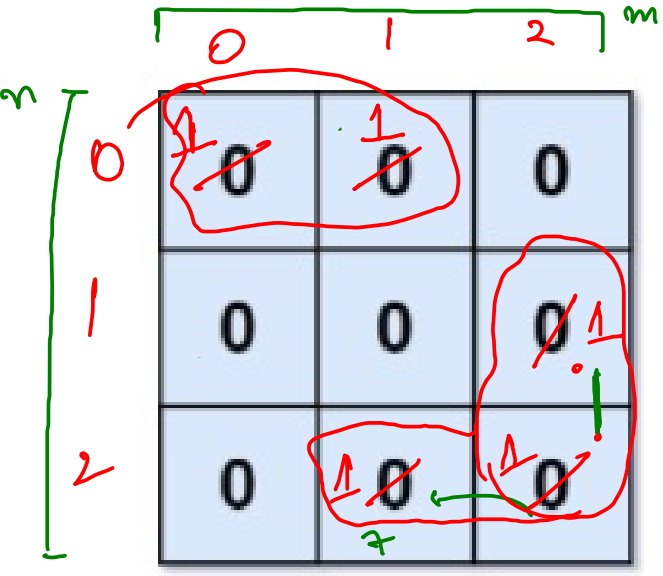
only Union by Rank ⟶ Time Complexity → O(log n)

Path Compression + Union by Rank ⟶ Time Complexity ⇒ O(α(N))

↳ Inverse Ackermann function

Average → Constant

O(1)



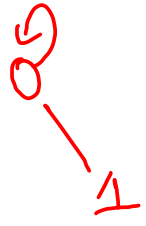
r, c
↳ idx => r*m+c

2, 3+2

```
positions = [[0,0],[0,1],[1,2],[2,1]]
```

[2,2]

0 1 5 7 8



Count = 0 1 2 1 2 3 4 5 2

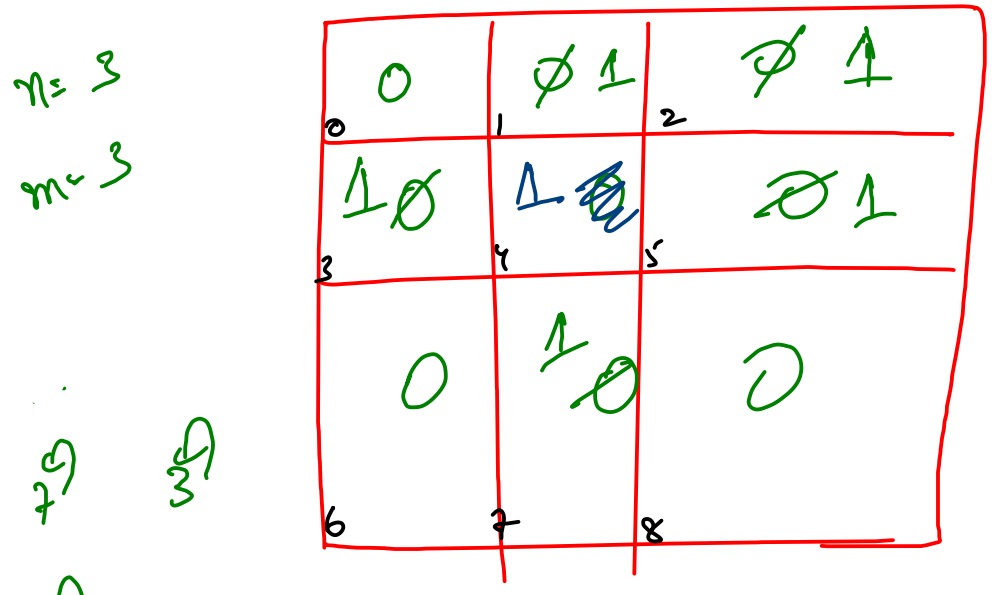
4

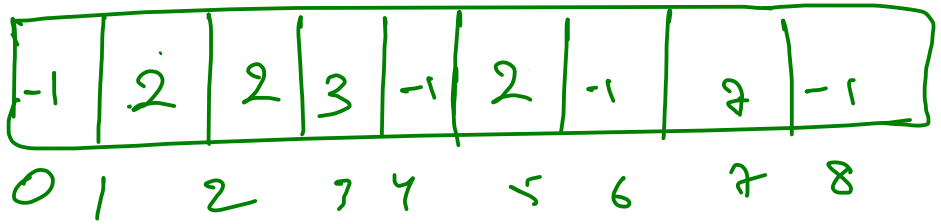
par =

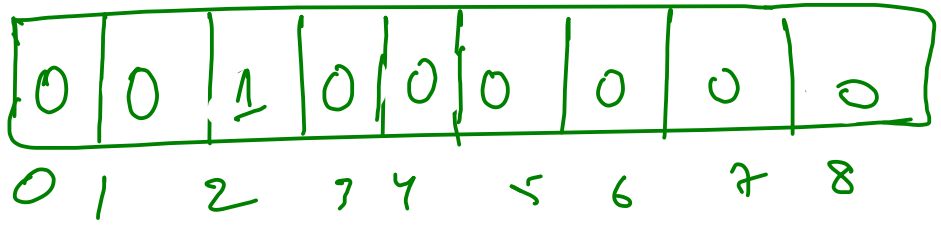
0	1	2	3	4	5	6	7	8
0	0	-1	-1	-1	5	-1	5	5

rank =

0	1	2	3	4	5	6	7	8
1	0	0	0	0	1 0	0	0	0

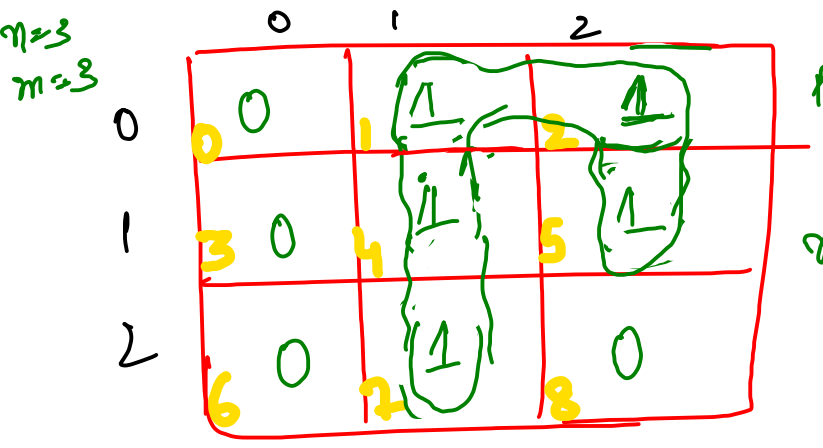






```
ArrayList<Integer> res = new ArrayList<>();
for(int pos[] : positions){
    int r = pos[0] , c = pos[1];
    int idx = r*m + c;
    if(mat[r][c] != 1){
        mat[r][c] = 1;
        obj.init(idx);

        int ni = (r-1)*m + c , ei = (r)*m + (c+1) , si = (r+1)*m + (c) , wi = (r)*m + (c-1);
        if(r-1 >= 0 && mat[r-1][c] == 1){
            obj.union(idx,ni);
        }
        if(c+1 < m && mat[r][c+1] == 1){
            obj.union(idx,ei);
        }
        if(r+1 < n && mat[r+1][c] == 1){
            obj.union(idx,si);
        }
        if(c-1 >= 0 && mat[r][c-1] == 1){
            obj.union(idx,wi);
        }
    }

    res.add(obj.count);
}
```

Input	3 3 [[0,1],[1,2],[2,1],[1,0],[0,2],[0,0],[1,1]]
Output	[1,2,3,4,3,2,-1]
Expected	[1,2,3,4,3,2,1]



par

0	1	2	3	4	5	6	7	8
-1	2	2	-1	2	2	-1	7	-1

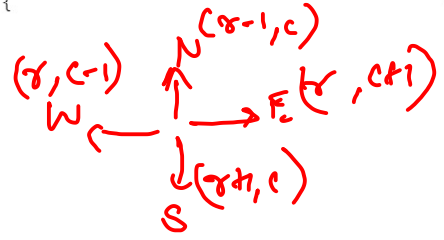
rank

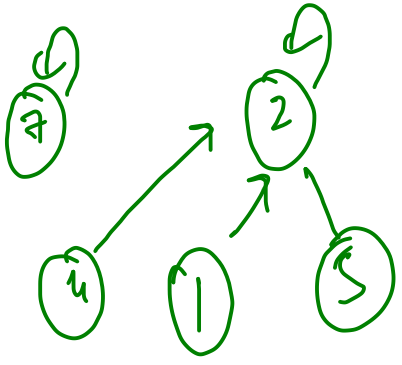
0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	0	0	0

1 2 3

Count = 1 2 3 4 5 6

idx = 4 ⇒ r : 1 , c : 1






```java
public List<Integer> numIslands2(int n, int m, int[][] positions) {
    UnionFind obj = new UnionFind(m*n);
    int mat[][] = new int[n][m];

    ArrayList<Integer> res = new ArrayList<>();
    for(int pos[] : positions){
        int r = pos[0] , c = pos[1];
        int idx = r*m + c;
        if(mat[r][c] != 1){
            mat[r][c] = 1;
            obj.init(idx);

            int ni = (r-1)*m + c , ei = (r)*m + (c+1) , si = (r+1)*m + (c) , wi = (r)*m + (c-1);
            if(r-1 >= 0 && mat[r-1][c] == 1){
                obj.union(idx,ni);
            }
            if(c+1 < m && mat[r][c+1] == 1){
                obj.union(idx,ei);
            }
            if(r+1 < n && mat[r+1][c] == 1){
                obj.union(idx,si);
            }
            if(c-1 >= 0 && mat[r][c-1] == 1){
                obj.union(idx,wi);
            }
        }

        res.add(obj.count);
    }

    return res;
```

```java
class UnionFind{
    int par[];
    int rank[];
    int count;


    UnionFind(int vtces){
        par = new int[vtces];
        rank = new int[vtces];
        count = 0;
        Arrays.fill(par,-1);
    }
    void init(int v){
        count++;
        par[v] = v;
    }
    void union(int v1,int v2){
        int rt1 = find(v1);
        int rt2 = find(v2);
        if(rt1 != rt2){
            count--;
            int rank1 = rank[v1];
            int rank2 = rank[v2];

            if(rank1 < rank2){
                par[rt1] = rt2;
            }else if(rank1 > rank2){
                par[rt2] = rt1;
            }else{
                par[rt2] = rt1;
                rank[rt1]++;
            }
        }
    }
    int find(int v){
        if(par[v] == v){
            return v;
        }
        return par[v] = find(par[v]);
    }
}
```