Today's agenda

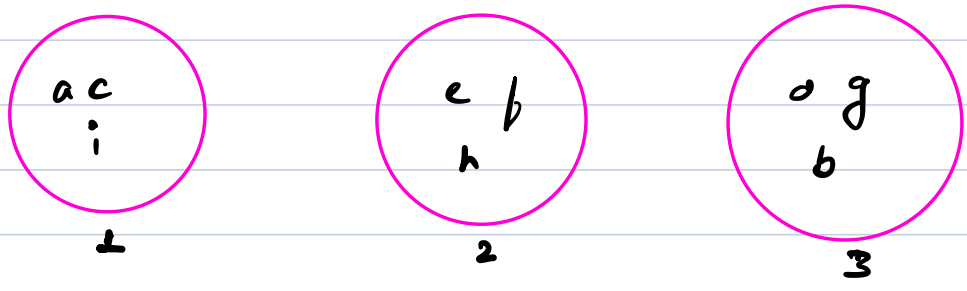
- DSU
- Problems on DSU

AlgoPrep

DSU

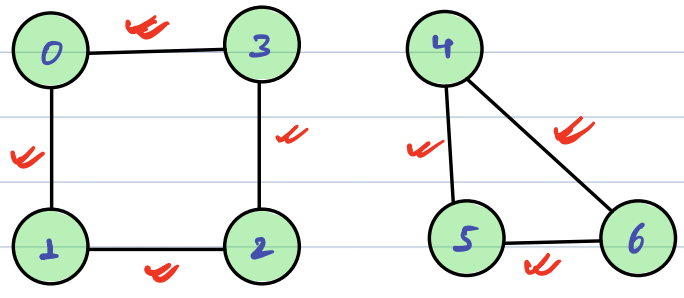
↳ Disjoint set union → (creating sets by following property).
Given in the Problem statement

a b c d e f g h i



AlgoPrep

Q) No. of Connected Component in a graph.

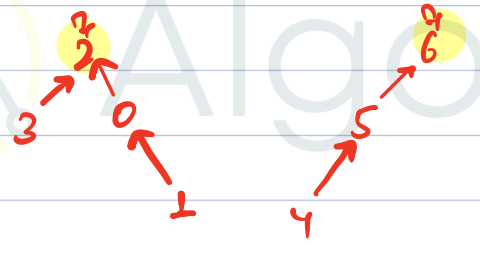


edges:

0	1
1	2
2	3
0	3
4	5
5	6
4	6

Par:

0	1	2	3	4	5	6
~~0~~ 2	~~1~~ 0	2	~~3~~ 2	~~4~~ 5	~~5~~ 6	6



x	y	P(x)	P(y)
1	2	0	2
2	3	2	3
0	3	2	2
4	5	4	5
5	6	5	6
4	6	6	6

AlgoPrep

//Pseudo code → v1

```
int Par[n];
int ConnectedComponents (int [m][2] edges, int N){
    for (int i=0; i<N; i++){
        Par[i] = i;
    }

    for (int i=0; i<m; i++){
        int x = edges[i][0];
        int y = edges[i][1];

        union (x,y);
    }
    int ans=0;
    for(int i=0; i<N; i++){ if (Par[i]==i){ ans++; } }
    return ans;
}
```

O(E*V) ←

```
Public void union (int x, int y){
    int Px = find (x);
    int Py = find (y);

    if (Px==Py) { return; }

    Par[Px] = Py;
}
```

T.C: O(V) ←

AlgoPrep

T.C: O(V)

```
Public int find (int x){
    if (x == Par[x]){ return x; }

    int temp = find (Par[x]);
    return temp;
}
```

Tracing

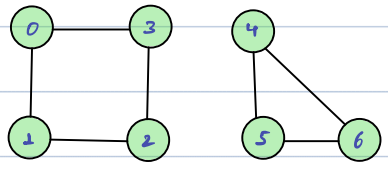
```
int Par [n];
int ConnectedComponents ( int [n][2] edges , int N){
    for (int i=0; i<N; i++){
        Par[i] = i;
    }

    for (int i=0; i<m; i++){
        int x = edges[i][0];
        int y = edges[i][1];

        union (x,y);
    }
}
Public void union (int x, int y){
    int Px = find (x);
    int Py = find (y);

    if (Px == Py) {return;}
    Par[Px] = Py;
}
Public int find (int x){
    if (x == Par[x]){ return x; }

    int temp = find (Par[x]);
    return temp;
}
```



edges:

0	1
1	2
2	3
0	3
4	5
5	6
4	6

i →

Par:

0	1	2	3	4	5	6
~~0~~ 1	~~1~~ 2	~~2~~ 3	3	4	5	6

x	y	Px	Py
0	1	0	1
1	2	1	2
2	3	2	3
0	3	3	3

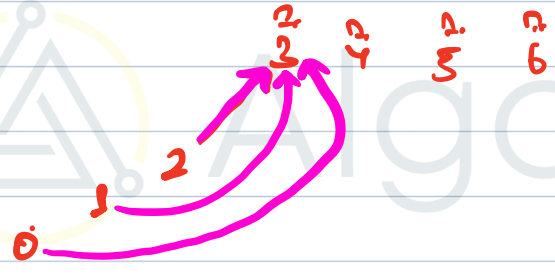
AlgoPrep

* Optimization of DSU
  - (i) Path compression.
  - (ii) union by rank.

(1) Path Compression.

Par:

0	1	2	3	4	5	6
~~3~~ 2	~~2~~ 3	~~3~~ 2	3	4	5	6



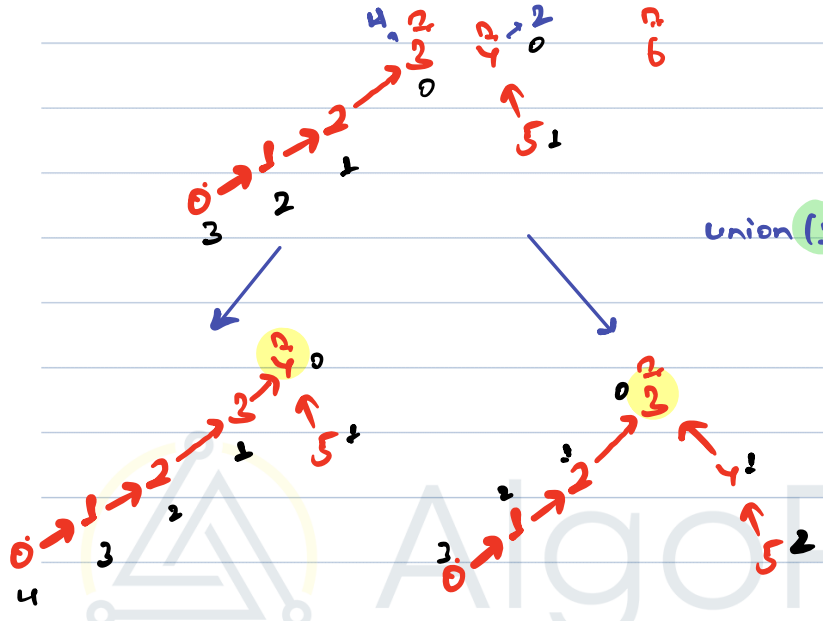
find(3) 1
3
find(2) 123
3
find(1) 1234
3
find(0) 1234
3

```
Public int find (int n){
    if (n == Par[n]){ return n; }
    int temp = find (Par[n]);
        Par[n] = temp;
        return temp;
}
```

(ii) union by rank

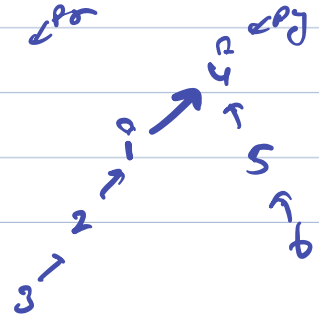
0	1	2	3	4	5	6
~~1~~	~~2~~	~~3~~	3	4	~~5~~ 4	6

Par:



union (3,4) → Pr : 3, Py = 4

↳ while doing union, you make leader of longer chain as your new leader.



//Pseudo code ⇝ v2

```
int Par[N];
int rank[N];
int ConnectedComponents (int [M][2] edges, int N){

    for (int i=0; i<N; i++){
        Par[i] = i;
        rank[i] = 1;
    }                                   O(V)

    for (int i=0; i<m; i++){
        int x = edges[i][0];
        int y = edges[i][1];
        union(x,y);
    }                                   O(E)

    int ans=0;
    for(int i=0; i<N; i++){ if (Par[i]==i){ ans++; } }
    return ans;
}                                       T.C: O(V+E)

Public void union(int x, int y){
    int Px = find(x);
    int Py = find(y);
    if (Px==Py){ return; }
    if (rank[Px] > rank[Py]){
        Par[Py] = Px;
    }
    else if (rank[Px] < rank[Py]){
        Par[Px] = Py;
    }
    else{ Par[Px] = Py;
          rank[Py]++; }
}
```

O(1) ←

union by rank

AlgoPrep

```
Public int find (int x){
        if (x == Par[x]){ return x; }
    int temp = find (Par[x]);
        Par[x] = temp;   ] Path compression
        return temp;
}
```

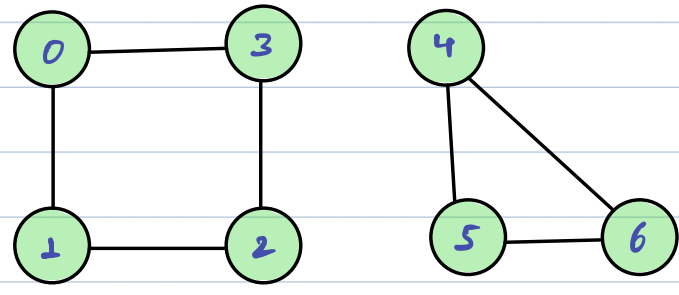
O(1) ←

→ Amortized analysis : A big data set of cases and then we come up with time complexity.

AlgoPrep

DFS vs DSU

Static



DFS

$O(V+E)$

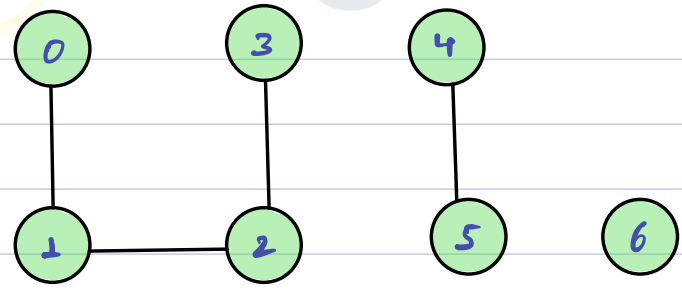
(better)

DSU

$O(V+E)$

AlgoPrep

Dynamic graph

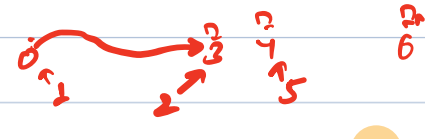


DFS

$O[E * (V+E)]$

DSU

$O(E+V)$



Ans = 7 ~~8~~ ~~4~~ 3

leetcode 1579

Q) Remove max number of edges to keep graph fully traversable.

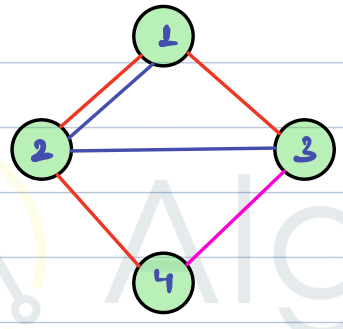
→ A graph with 3 types of edges are given:

Type 1: Can be traversed by Alice only.

Type 2: Can be traversed by Bob only.

Type 3: Can be traversed by both Alice and Bob.

Ex:

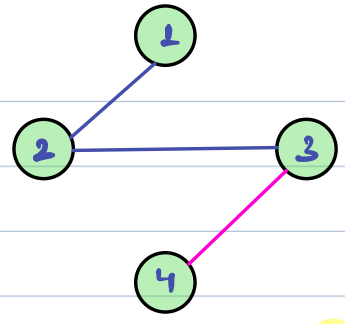


	0	1	2
0	3	1	2
1	3	2	3
2	1	1	3
3	1	2	4
4	1	1	2
5	2	3	4

//idea

↳ remove max edge :: keep min no. of edge.

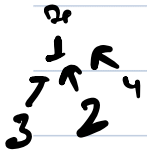
AlgoPrep



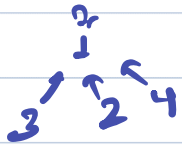
	0	1	2
0	3	1	2 ✓
1	3	2	3 ✓
2	1	1	3 ✗ ✓
3	1	2	4 ✓
4	1	1	2 ✗ ✓
5	2	3	4 ✓

↳ ans = 2

Alice



Bob



//Pseudo Code

```
Public boolean union (int x, int y, int[] Parent, int[] rank) {
    int Px = find(x);
    int Py = find(y);
    if (Px == Py) { return false; }
    if (rank[Px] > rank[Py]) {
        Par[Py] = Px;
    }
    else if (rank[Px] < rank[Py]) {
        Par[Px] = Py;
    }
    else { Par[Px] = Py;
           rank[Py]++; }
    return true;
}
```

```
int maxEdgeRemoval (int edges[m][3], int N){
    Arrays.sort( edges, (a,b) -> Integer.compare(b[0],a[0]));

        int Parenta[N+1];
        int ranka[N+1];
        int Parentb[N+1];
        int rankb[N+1];
        for(int i=1; i<=N; i++) {
            Parenta[i]= Parentb[i]= i;
            ranka[i] = rankb[i] = 1;
        }
                                    int countA=1;
                                    int countb=1;
        int ans=0;
        for (int i=0; i<edges.length; i++) {
            int cat = edges[i][0];
            int x = edges[i][1];
            int y = edges[i][2];

            if (cat == 3) {
                bool mergeA= union(x,y, Parenta, ranka);
                bool mergeb= union(x,y, Parentb, rankb);
                if (mergeA == false && mergeb == false){
                    ans++;
                }
                if (mergeA == true) { countA++;}
                if (mergeb == true) { countb++;
            }
```

```
else if (cat == 2) {
    bool mergeb = union(x, y, Parentb, rankb);
    if (mergeb == false) {
        ans++;
    }
    else { countb++; }
}
else {
    bool mergea = union(x, y, Parenta, ranka);
    if (mergea == false) {
        ans++;
    }
    else { counta++; }
}
}

if (counta != n || countb != n) { return -1; }

return ans;
}
```

AlgoPrep