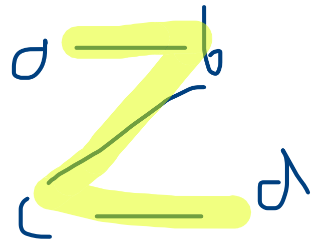
Disjoint Set Union (DSU)

DS + Algo

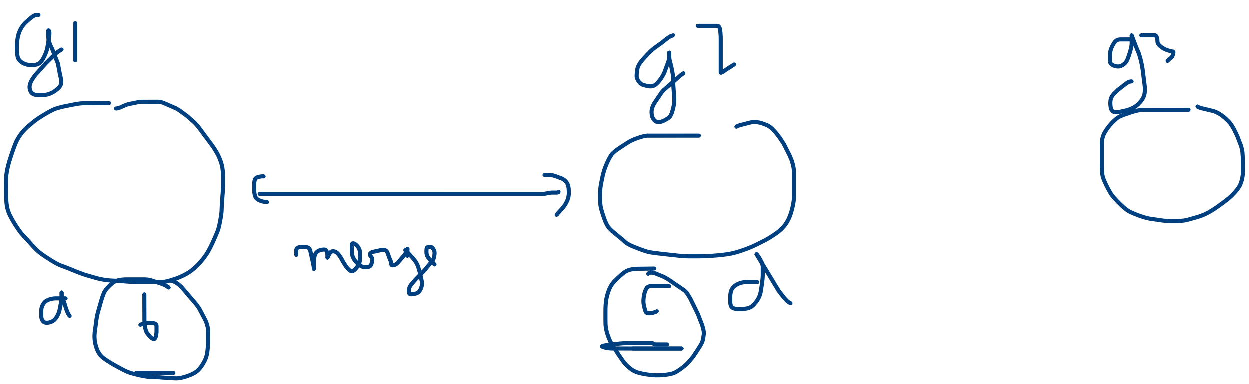


algo →

a — b

c — d

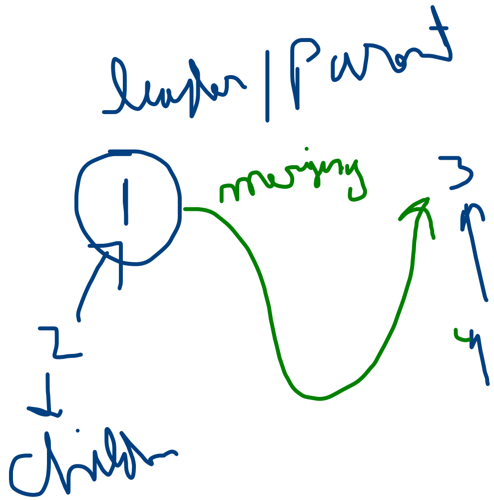
b — c



Transistive --> must for this algo

DSU

1. Union merge the grps
2. Find → find the leader



Edges

1 2

3 4

2 4

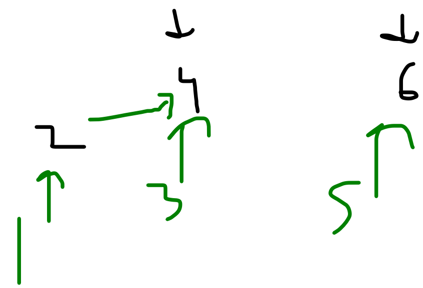
5 6

union

find

Parent
leader arr

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



Edges

1 2

3 4

2 4

5 6

PS int find(int x)

Parent
leader arr

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

Edges

1	2
3	4
1	3
5	6

f(1) → 1 → ret

f(2) → 2 → ret

U(1,2) → merge

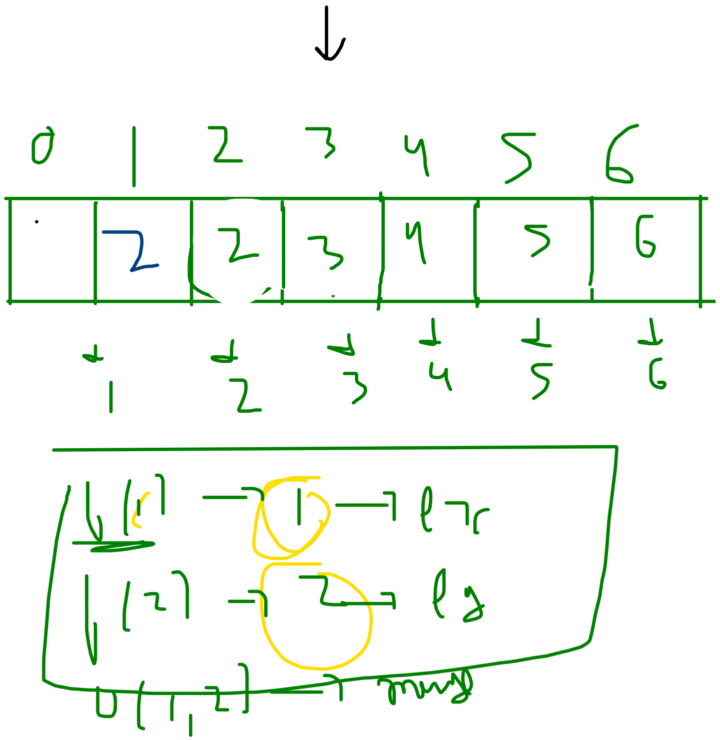
find --> to find the leader / parent

Union --> merge the groups

```java
public static int find(int x) {
    if (par[x] == x) {
        return x;
    }

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

```java
public static void union(int x, int y) {
    int lx = find(x);
    int ly = find(y);

    if (lx != ly) { // if they are not pr
        par[lx] = ly;
    }

}
```

Parent
leader arr



Edges

1 2
3 4
1 3
5 6

find --> to find the leader / parent x

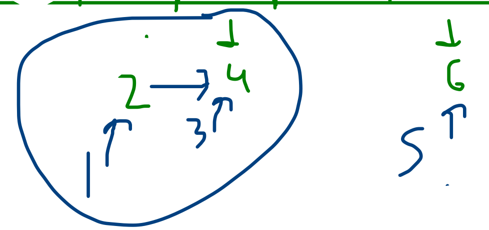
Union --> merge the groups

```java
public static int find(int x) {
    if (par[x] == x) {
        return x;
    }

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

```java
public static void union(int x, int y) {
    int lx = find(x);
    int ly = find(y);

    if (lx != ly) { // if they are not pr
        par[lx] = ly;
    }

}
```

Parent
leader arr

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



Edges

1 2
3 4
1 3
5 6

u(1,3)

lx = l(1) = 2
ly = l(3) = 4

find --> to find the leader / parent

Union --> merge the groups

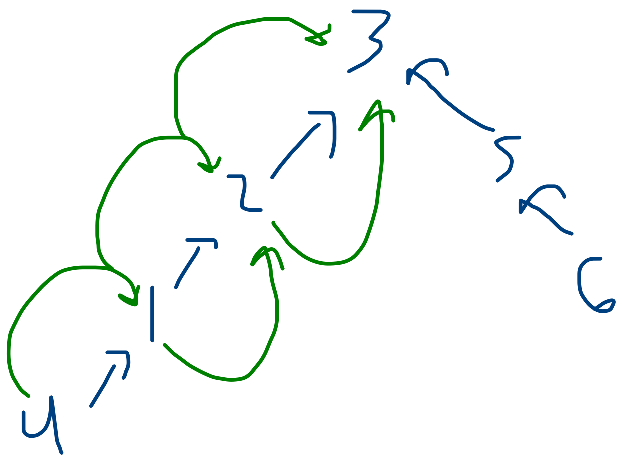
optimization of DSU

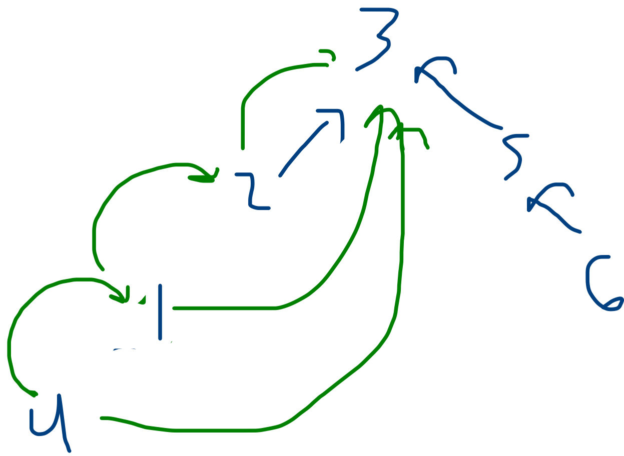
1. Path Compression
2. Union by Rank / Size

1. Path Compression

$\boxed{f[4] = 3}$

$f[1] = 3$





avg = $\log(n)$

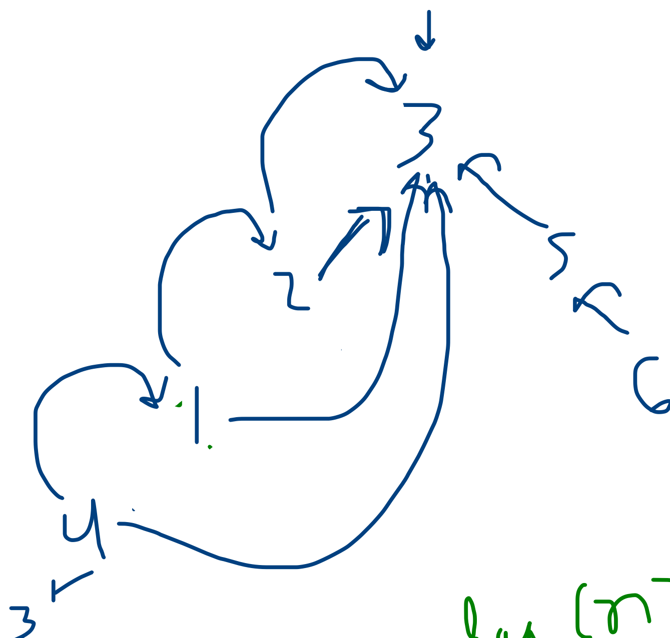
```java
public static int find(int x) {
    if (par[x] == x) {
        return x;
    }

    int temp = find(par[x]);
    par[x] = temp; // path compression
    return temp;
}
```

optimization of DSU

1. Path Compression
2. Union by Rank / Size

1. Path Compression

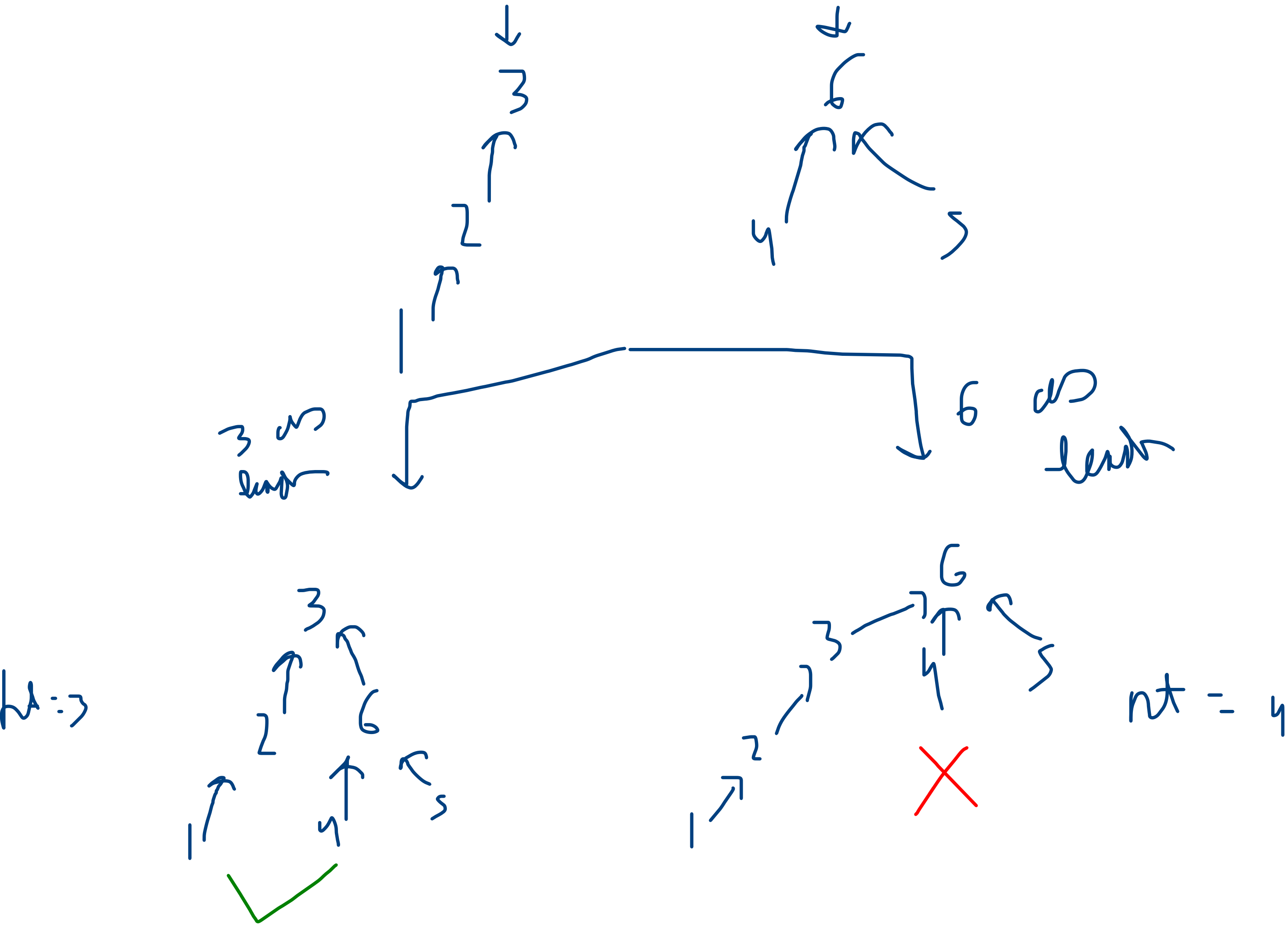
find(4) = 3



avg = log(n)

2. Union by rank

union



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

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

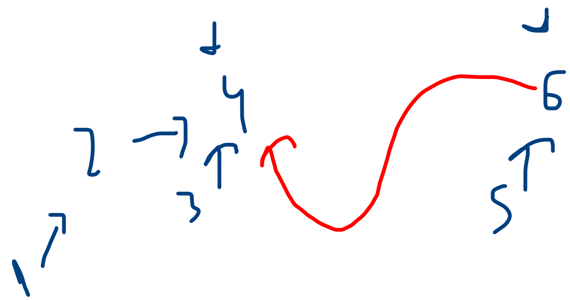
→ 1 2

→ 3 4

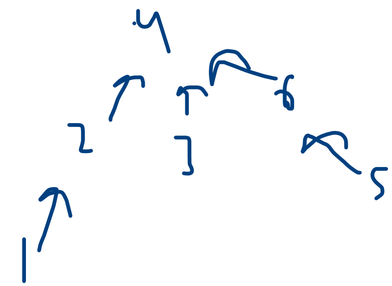
→ 1 3

5 6

1 5



=



p[4] = 3

p[6] = 2

union

union by rank --> whoever have highest rank is new leader

avg = $\log n$

Union by Size

U(1 2)

U(2 3)

U(4 5)

U(6 7)

U(5 6)

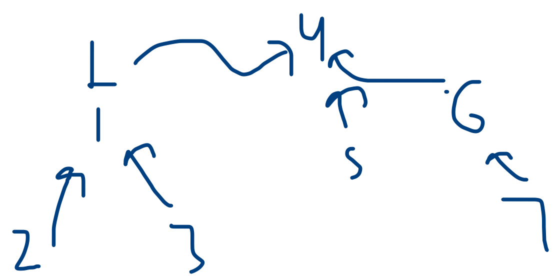
U(3, 7)

Par

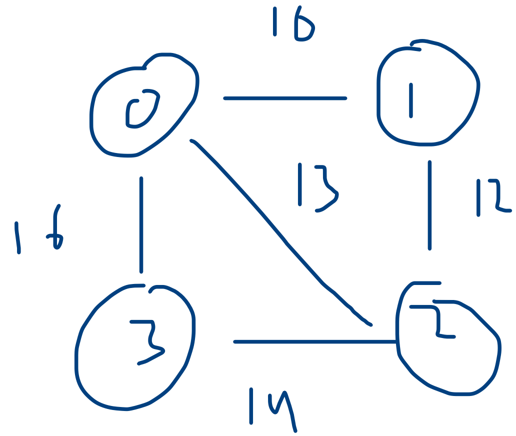
0	1	2	3	4	5	6	7
	4	1	1	4	4	4	6

Size

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



Kruskal's Algorithm (Minimum Spanning Tree)



U — V		wt
0	1	10
1	2	12
2	3	14
0	3	16

V → vertices
↓
V − 1 → edges

goal

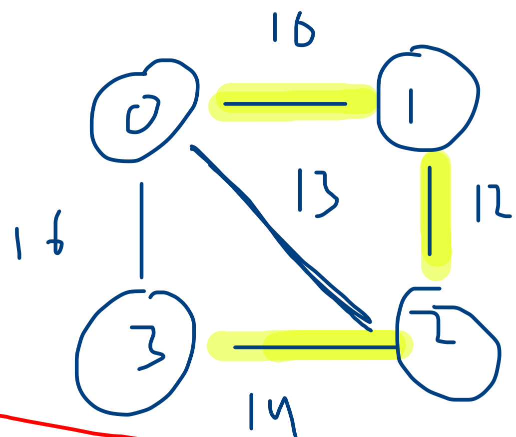
1 edge

2 vertices

V − 2
edge

V − 2
vertices

Kruskal's Algorithm (Minimum Spanning Tree)



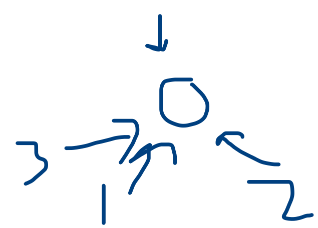
MST

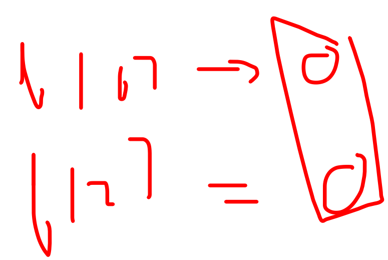
0 — 1

3 — 2

	U — V		Wt
→	0	1	10
→	1	2	12
→	~~0~~	~~2~~	~~13~~
→	2	3	14
→	~~0~~	~~3~~	~~16~~

ans $= 0 + 10 + 12 + 14 = 36$





Step1: Sort on the basis of Wts
Step2: Apply DSU

and ignore same leader edge