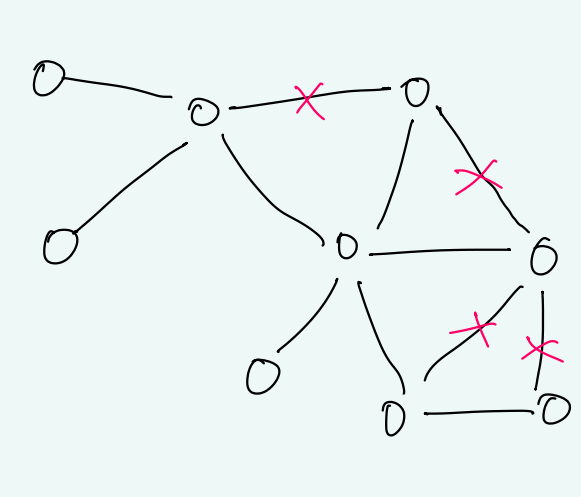
Graphs 5

Saturday, 25 November 2023 3:41 PM

Q You are a delivery person! N houses (all connected) Can you remove some roads such that it's still possible to deliver to all houses. (Remove as many edges as possible) The graph left should still have 1 connected component

⇒



V = 9
E = 12

we only need = V-1
= 9 - 1 = 8

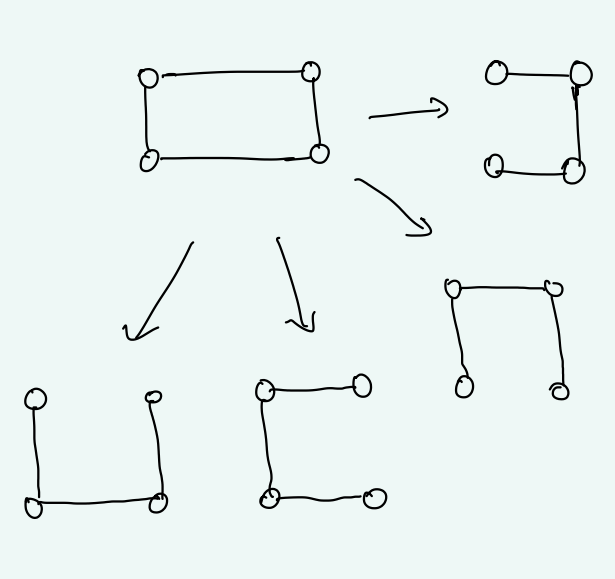
④ are extra

⦿ Observations

→ For 1 CC with V nodes, we need at least V-1 edges

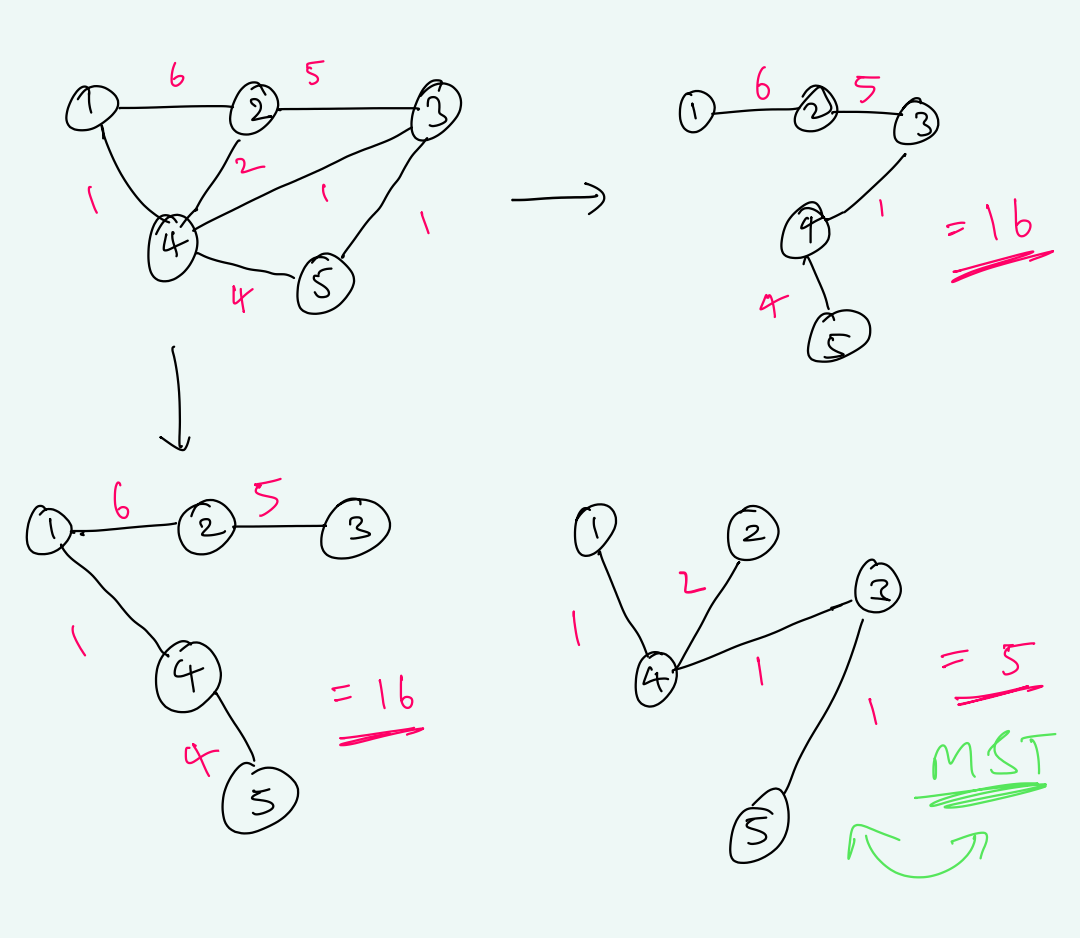
$\boxed{\text{Extra Edges: } E - (V-1)}$

⦿ Spanning Trees



⦿ A tree that spans over all the nodes of the graph, obtained by removing some edges s.t. the graph is still connected.

⦿ Weighted Graphs



⦿ Minimum Spanning Tree

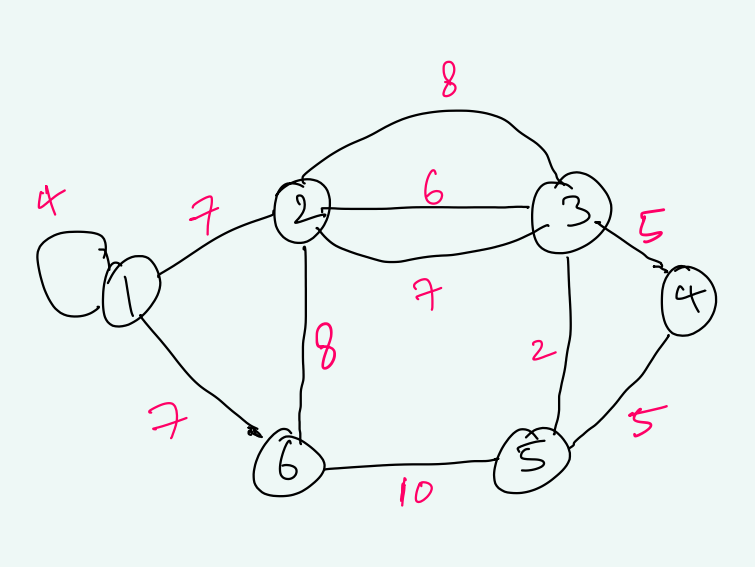
→ The spanning Tree with the least total edge weights

→ There can be multiple MSTs in a Graph

⦿ 2 Algorithms for finding MSTs

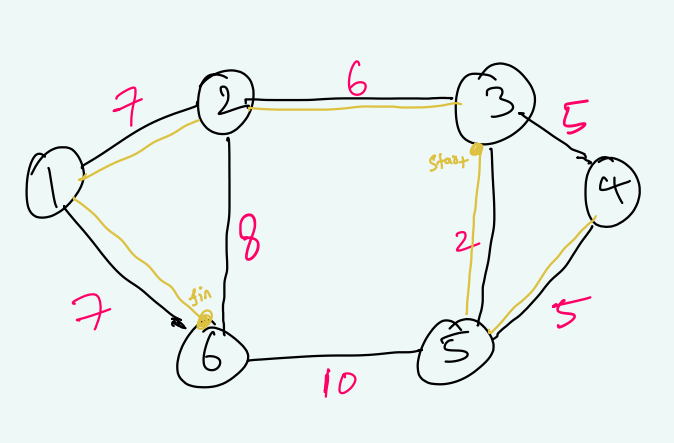
1) Prim's Algo
2) Kruskal's Algo

⦿ Prim's Algorithm



→ Remove the self loops

→ For multiple edges b/w 2 nodes, remove all but smallest wt. edge



⦿ This is a Greedy Algo

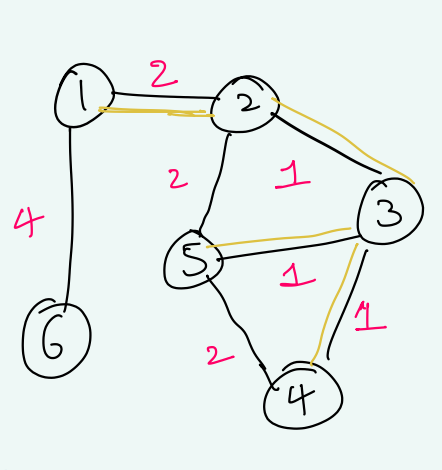
→ Start from any random node. Pick the smallest outgoing edge from it and move to that node

→ Now these 2 nodes form a CC. Now pick min outgoing edge from the whole CC (from all nodes of the CC) and move through that edge.

→ This way you move through min wt. edges and keep increasing the CC.

⦿ Implementation

~~<2, <1,2>>~~ 1
~~<4, <1,6>>~~ 7
~~<2, <2,5>>~~ 6
~~<1, <2,3>>~~ 2
~~<1, <3,5>>~~ 4
~~<1, <3,4>>~~ 3
~~<2, <5,4>>~~ 5



① Insert
② Get min
③ Del min

[<wt, <u,v>>] → min Heap

+ visited array

⦿ Algo

1. Pick a random node
2. Insert all the outgoing edges in a min heap. Mark it as visited.

<w, <u,v>>

```
3. While heap != empty:
       pop min → w, u, v
       check if v is not visited
       if vis[v]: continue
       vis[v] = True
       insert neighbouring edges leading to
       unvisited nodes in the heap.
```

⦿ Initially, start with <0, <r,r>> in the heap.

TC:

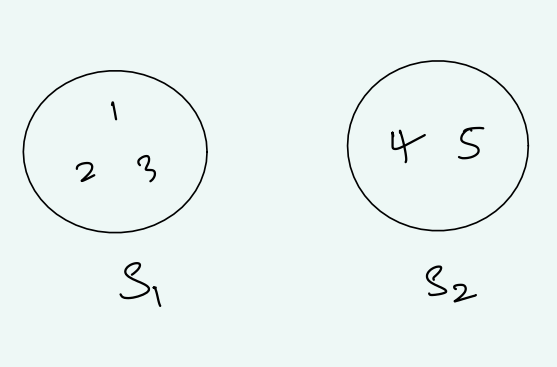
worst case we visit all edges of every node and insert nodes (2E) in the heap. $TC = O(E \log E)$

- And visiting every edge itself is $TC = O(V)$

⇒ $TC = O(V + E \log E)$
$SC = O(V+E)$
↳ min heap = 2E, vis = V

Disjoint Set Union (DSU)

Disjoint sets → don't have anything in common



$\boxed{S_1 \cap S_2 = \phi}$

S_1 & S_2 are disjoint

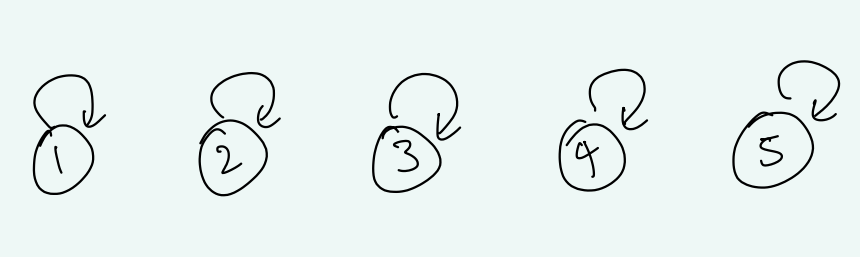
set	①	②	③	④	⑤
label	1	2	3	4	5

Union (1,2) = (1,2)
2 → any label

Union (3,4) = (3,4) 3

Union (1,4) = (1, 2, 3, 4) 2

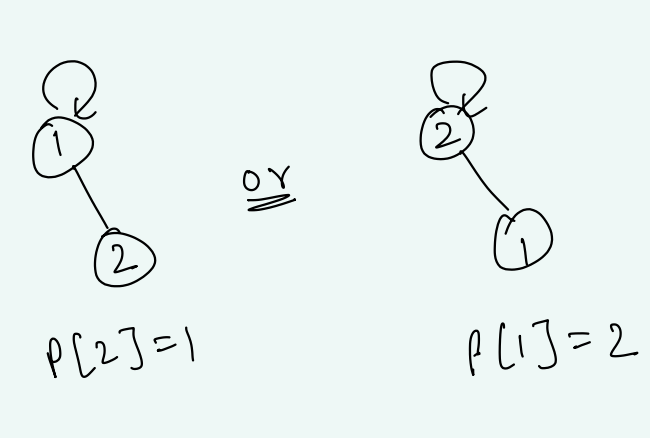
⦿ Consider sets as Trees



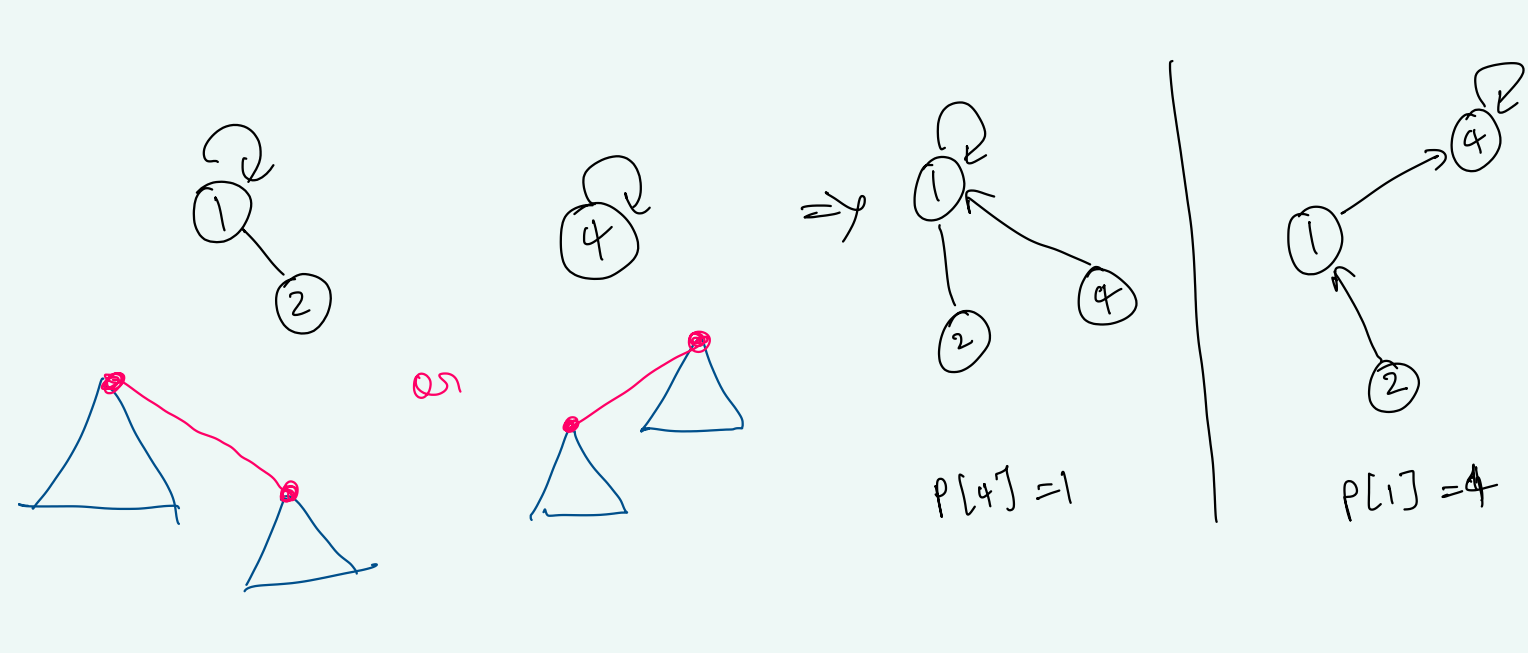
Make these nodes parents of themselves

p[i] = i → parent array

Union(1,2) →



Union(2,4) →

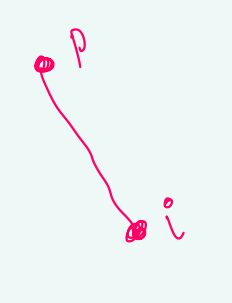


⦿ CODE

```
int p[N] ∀i p[i]=i

def find(x):
    while x != p[x]:
        x = p[x]
    return x
```

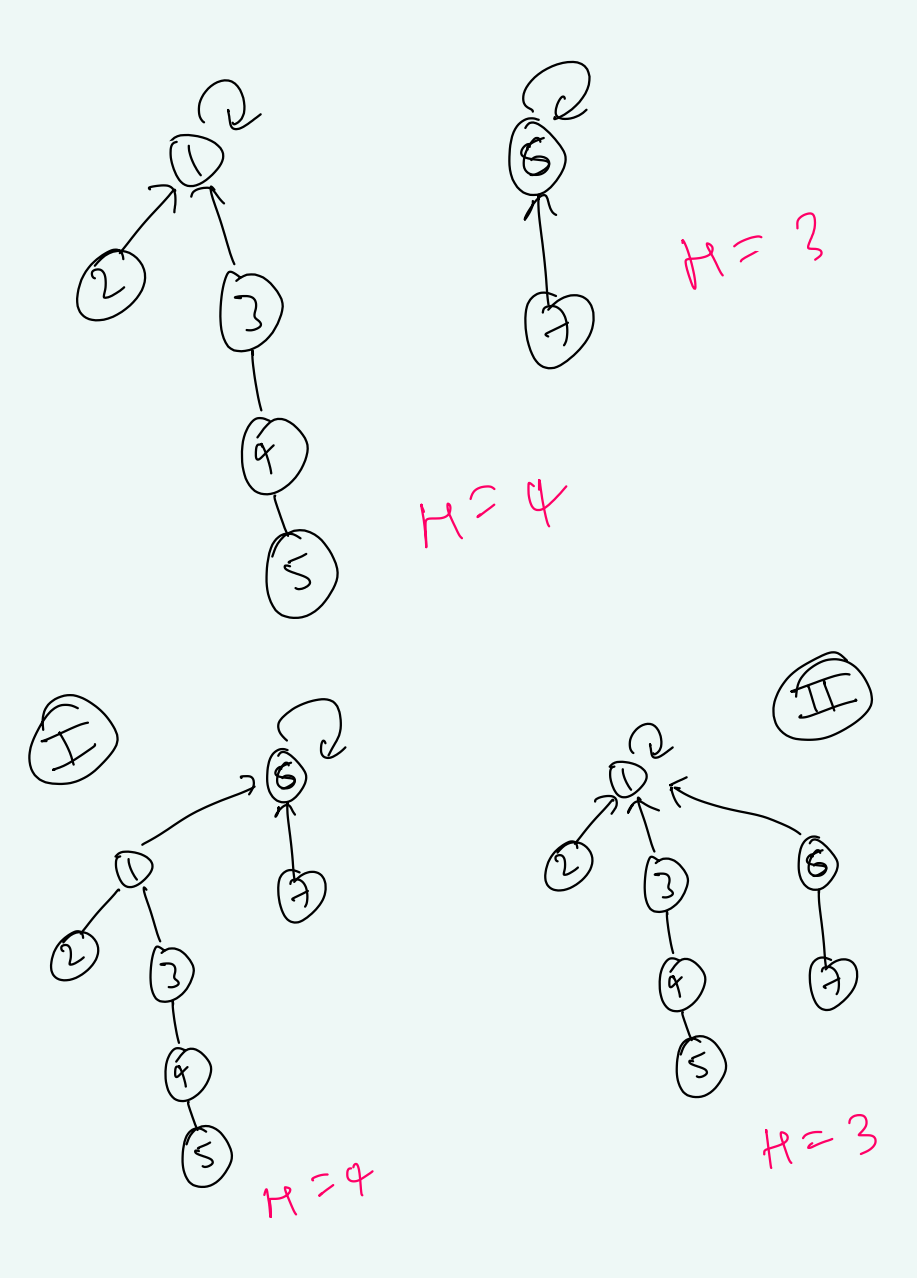
TC = O(N)



```
def union(x,y):
    x = find(x)
    y = find(y)
    if x==y: return
    p[x] = y
      or
    p[y] = x
```

TC = O(N)

⦿ Optimisation

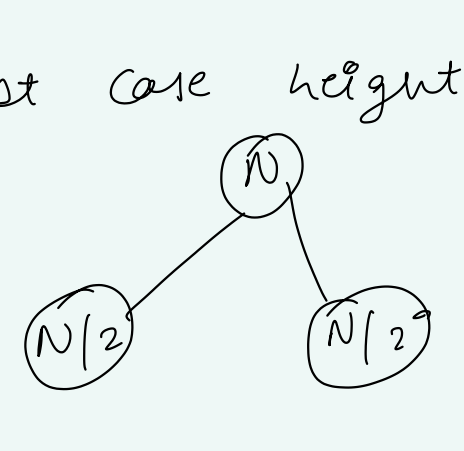


⦿ make tree with larger H as parent

⦿ But when does the H actually change?

→ If 2 trees of equal H are joined, then: $\boxed{H = H+1}$

⦿ worst case height for N nodes



$\boxed{H(N) = H(N/2) + 1}$

$\boxed{H(N) = \log N}$

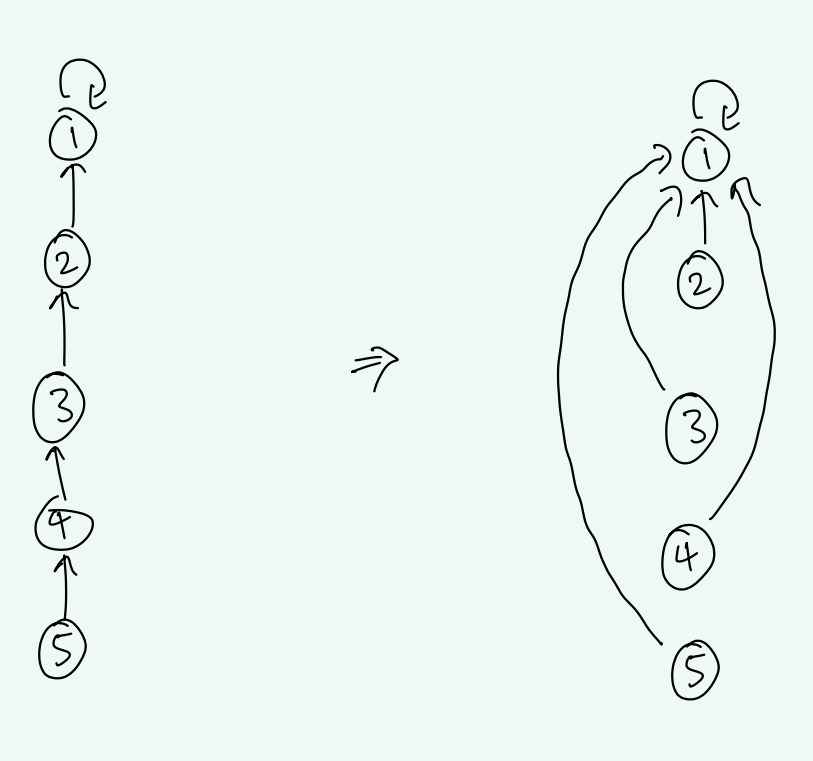
⦿ CODE ⇔ Union By Height

```
P = [i for i in range(1,N+1)]
H = [0] * N

def union(x,y):
    x = find(x)
    y = find(x)
    if x==y: return
    if H[x] > H[y]:
        p[y] = x
    elif H[x] < H[y]:
        p[x] = y
    else:
        p[y] = x
        H[x] += 1
```

$\boxed{TC: O(\log N)}$

⦿ Optimisation - Path Compression



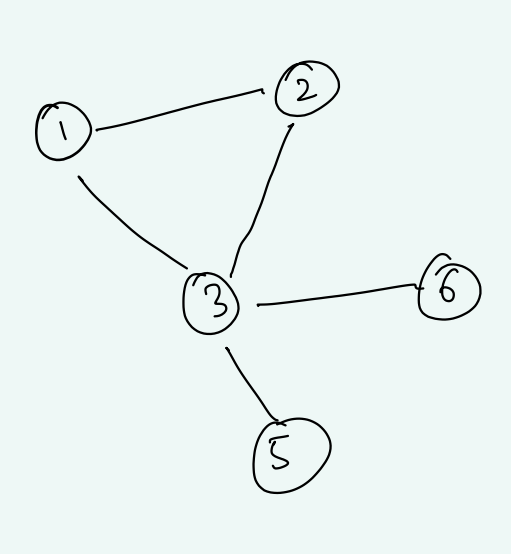
$\boxed{\text{Amortised } TC = O(1)}$

```
def find(x):
    if x == p[x]:
        return x

    p[x] = find(p[x])
    return p[x]
```

Q Given a graph, find if it's connected. i.e. 1 C.C

⇒



→ Consider all edges and do union of the nodes

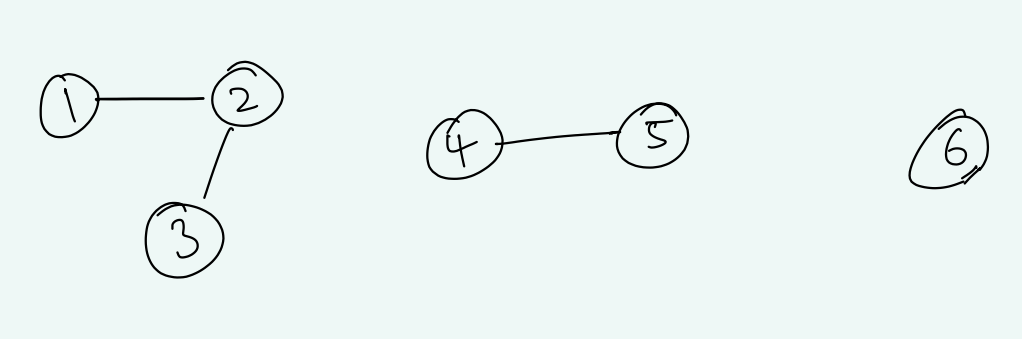
→ ∀i if find(i) is same, then they're part of 1 CC

Steps
1) Initialize DSU → V (#vertices)
2) ∀ [u,v] → union(u,v) → O(1)

$\boxed{TC = V+E}$ $\boxed{SC: O(V)}$

Q Given undirected graph, find # of CC

⇒

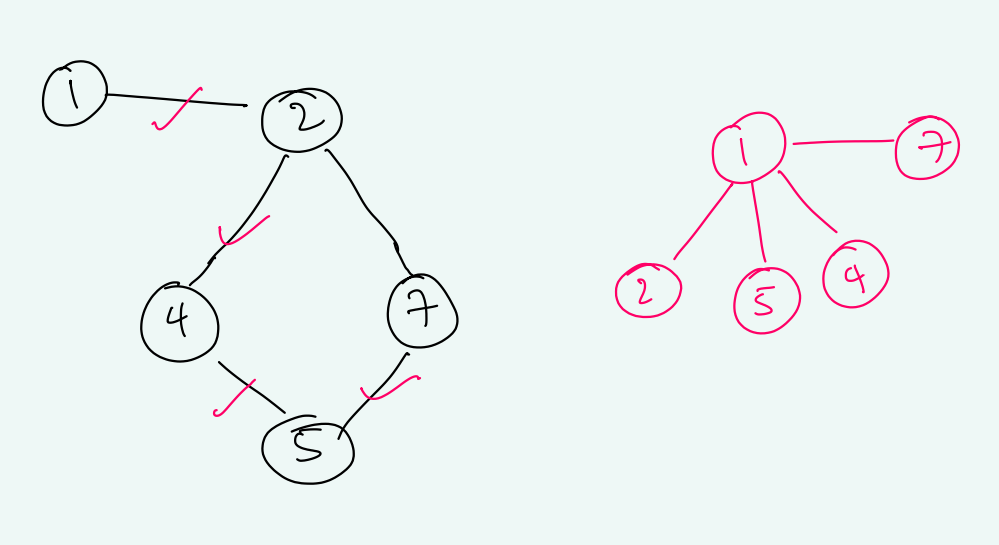


- Go over edge list (u-v) → E
- union (u,v) → O(1)
- Count # i s.t. p[i] = i → O(V)

$TC = O(V+E)$
$SC = O(V)$

Q Given an undirected graph, check if there is a cycle.

⇒

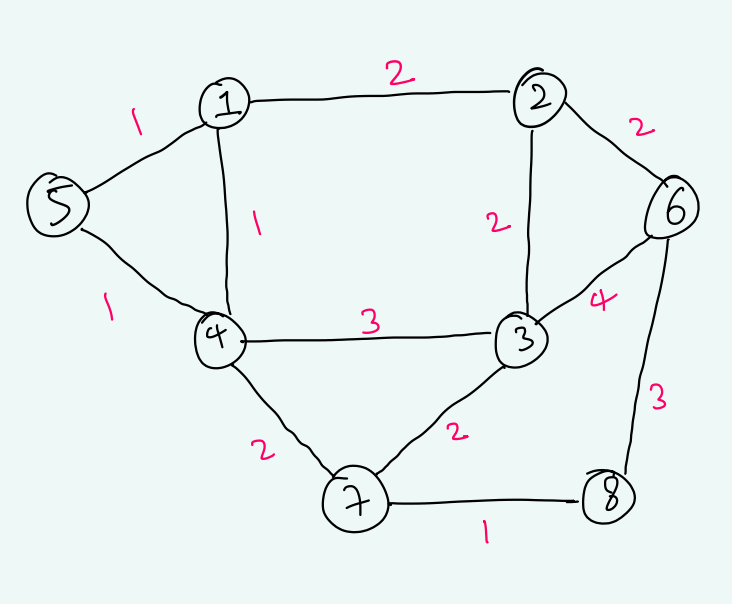


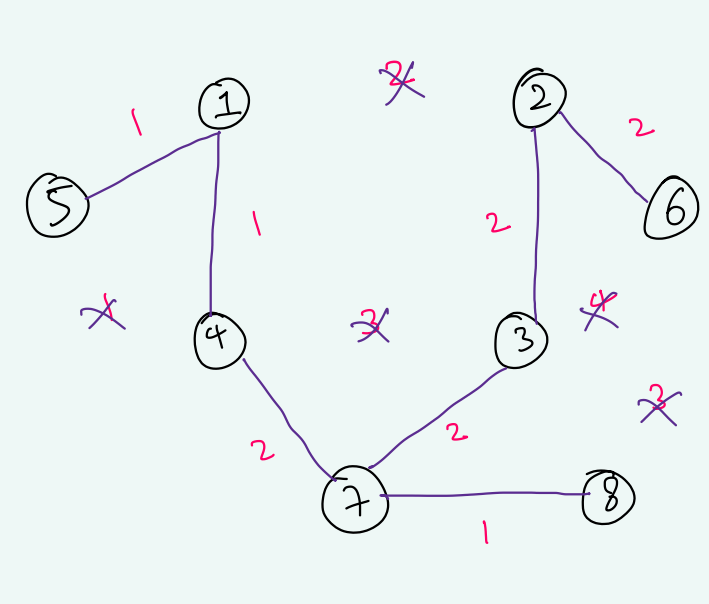
- Go through edge list (u,v)

```
if find(u) == find(v):
    return True
union(u,v)
return False
```

$\boxed{TC = O(V+E) \quad SC = O(V)}$

Kruskal's Algorithm





Sort edges in the increasing order of weights & try to add them (using DSU)

While adding edge, check if the two nodes are part of the same root. If they have same root, then adding the edge will introduce a cycle so we don't add the edge.

⦿ When to stop?
When we have added V-1 edges

Algo

```
1) Initialize DSU                    O(V)
2) Sort edges (inc wts)              O(E log E)
3) Iterate on sorted edges list
   (w, [u,v])   O(E)
      if find(u) != find(v):         O(1)
          ⇒ this edge is part of MST
          ⇒ process it
          union(u,v)
```

$\boxed{\begin{aligned} TC &= O(V + E\log E) \\ SC &= O(V) \end{aligned}}$