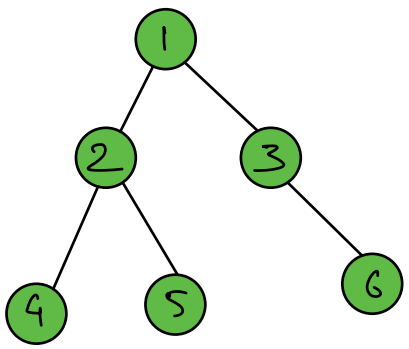
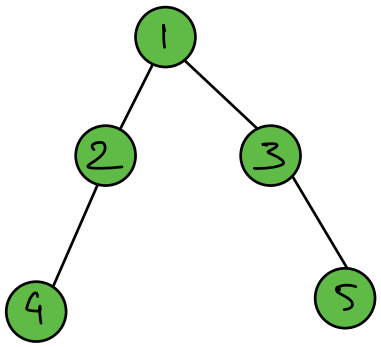
Cycle detection in undirected graph :-





In a Tree with N nodes, how many edges?

⇒ N-1 edges & 1 component

Given a graph with N nodes & components

# Nodes	# Components	# Edges it should have so that there's NO cycle.
N	1	N-1
N	2 → x ⇒ x-1 edges; → y ⇒ y-1 edges	x+y-2 = N-2 (N)
N	3 → x ⇒ x-1; → y ⇒ y-1; → z ⇒ z-1	x+y+z-3 ⇒ N-3

N	C	N-C

* Given a graph with N nodes & E edges, check if there's a cycle.

⇒ Calculate the no. of components = C

⇒ No cycle, if E == N − C

Algo

i) if E >= N ⇒ return true.

ii) No. of components = c
if(E != N-C) return true
return false

E ≈ N

TC ⇒ O(N+E) ≃ O(N)

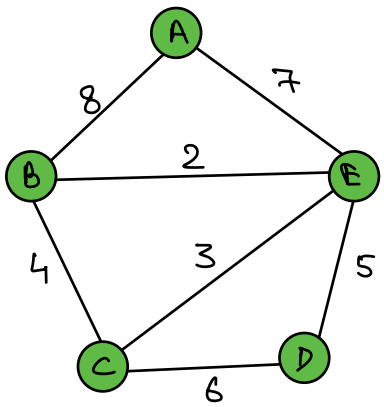
Minimum Spanning Tree (MST)

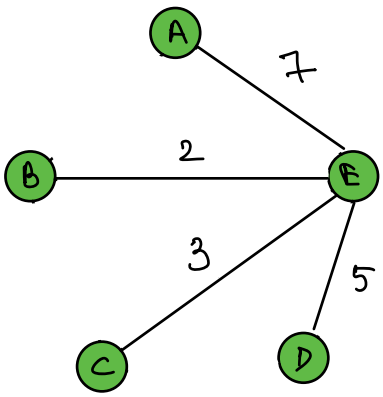
Given an undirected weighted connected graph, convert this graph into a tree with Minimum total weight.

→ {Sum of overall weights should be Min}

→ Minimum Spanning Tree

Ex

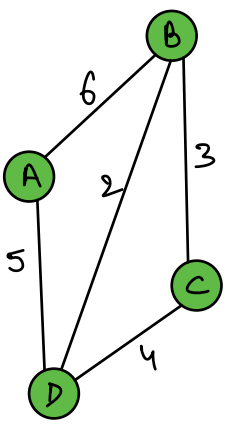


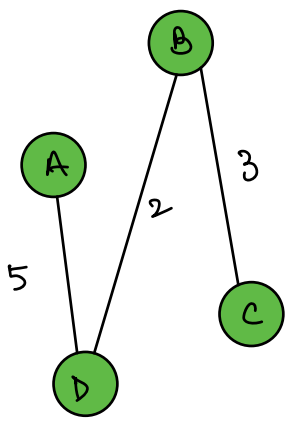


⇒ MST

wt = 17

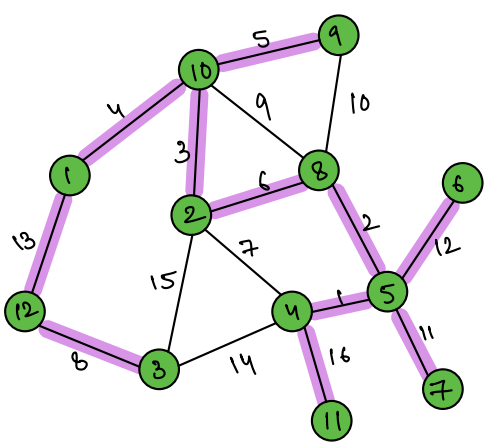
Ex





wt = 10

Ex



⇒ MST

wt = 81

Idea (Kruskal's Algorithm)

1) Sort the edges based on weights. $\Rightarrow O(E \log E)$

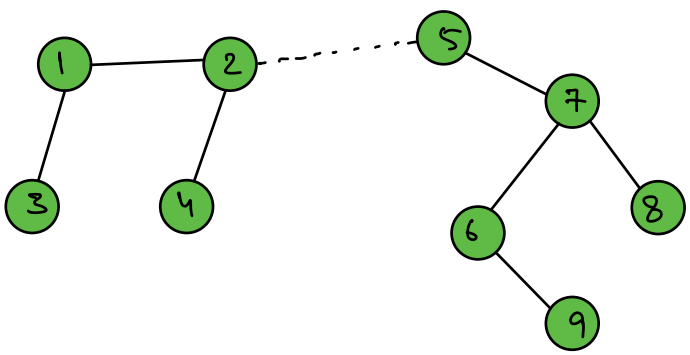
2) Add edges 1 by 1 to graph $\Rightarrow O(E \cdot N)$

   $\Rightarrow$ If a particular edge is forming a cycle, skip that edge, don't add.

   ↳ $O(N)$ → we need to optimize this step.

TC: $O(E \log E + E \cdot N)$

#



Obs:

1) When 2 nodes of 2 different components are getting connected, it forms a single component (without any cycle)

2) When 2 nodes of same component are getting connected, it forms a cycle.

Eg N = 10

int comp

0	1	2	3	4	5	6	7	8	9	10
x	1	~~2~~ 1	~~3~~ 2	~~4~~ 3	~~5~~ 1	~~6~~ 4	~~7~~ 2	~~8~~ 7	~~9~~ 2	~~10~~ 1

W u v

1 4 6 ⇒ Comp[6] = Comp[4]

3 3 4 ⇒ Comp[4] = Comp[3]

5 7 8 ⇒ Comp[8] = Comp[7]

5 2 7 ⇒ Comp[7] = Comp[2]

6 3 2 ⇒ Comp[3] = Comp[2]

8 6 8 ⇒ 2 : 2 ⇒ Skip

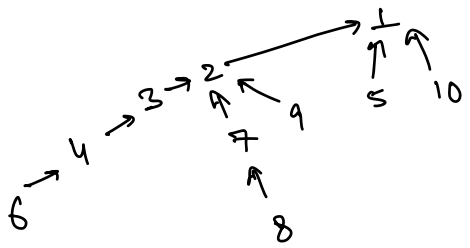
10 9 6 ⇒ 9 : 2 ⇒ Comp[9] = Comp[2]

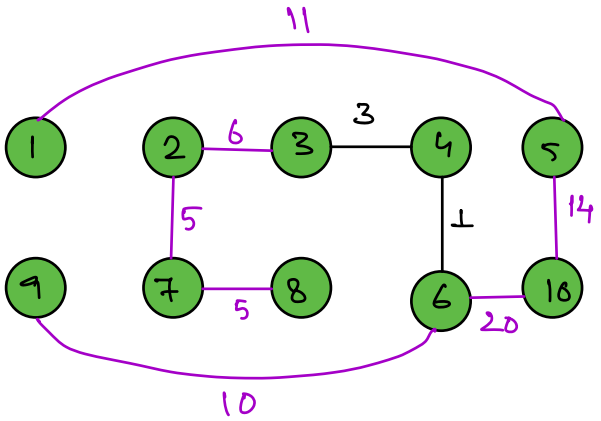
11 1 5 ⇒ 1 : 5 ⇒ Comp[5] = Comp[1]

12 7 9 : 2 : 2 ⇒ x

14 5 10 ⇒ 1 : 10 ⇒ Comp[10] = Comp[1]

20 6 10 ⇒ 2 : 1 ⇒ Comp[2] = Comp[1]





Rule:- Assign smaller component value to a larger component

```
int Kruskals ( list<pair<int, pair<int, int>>> edges, N){
    sort(edges) // Sort based on weights. → ElogE
    int comp[N+1];
    for(i=1; i<=N; i++) comp[i] = i;
    ans = 0;
    for( i= 0; i< edges.size(); i++) { → O(E)
        pair<int, pair<int, int>> data = edges[i];
        w = data.first
        u = data.second.first
        v = data.second.second;
        cu = find(u, comp)
        cv = find(v, comp)        } O(N)
        if( cu != cv) { // u&v belongs to different
                        // components.
            comp[max(cu,cv)] = comp[min(cu,cv)];
            ans += w; // u-v is considered.
        }
    }
    return ans;
}
```

Union find Algorithm

```
int find ( int x, int comp[]) {
    if( x == comp[x] ) return x;
    comp[x] = find ( comp[x], comp );
    return comp[x];
}
```

O(N) ⇓ ≈ O(1)

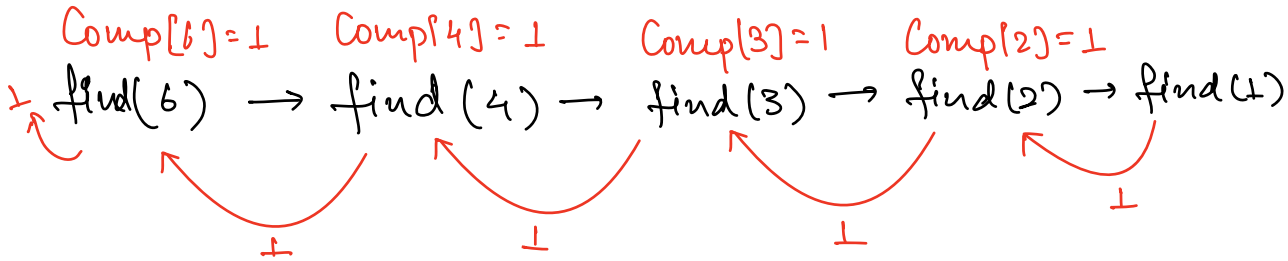
* Union find Algo : Detecting a cycle in an optimised way

TC : $O(E \log E + E \cdot N)$ ← find() fun

↓

TC : $O(E \log E + E \cdot O(1)) \Rightarrow O(E \log E + E)$

*



MST

↳ Kruskal's Algo

i) Sort all the edges

ii) Add edge by edge, after adding an edge check if there's a cycle or not.

↳ To optimize

⇒ Union find Algorithm.