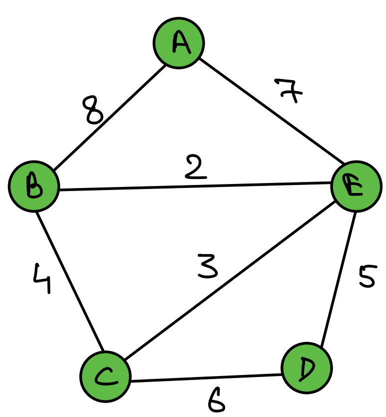
Cycle detection in Undirected graph : $O(N+E)$

Minimum Spanning Tree (MST)

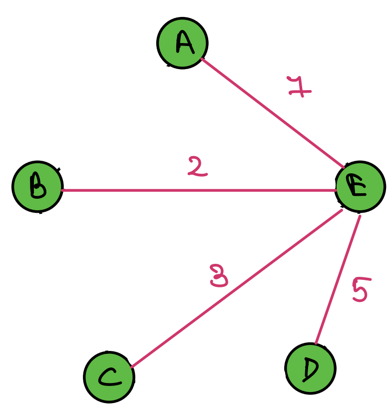
N nodes
N-1 Edges
→ No cycle.

Given an undirected weighted connected graph, Convert it to a Tree with Minimum overall weight.

N-1 Edges

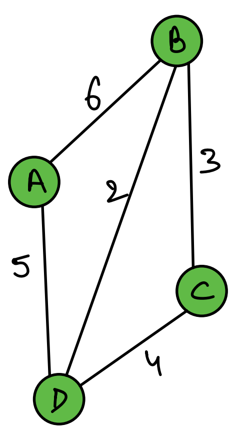


→

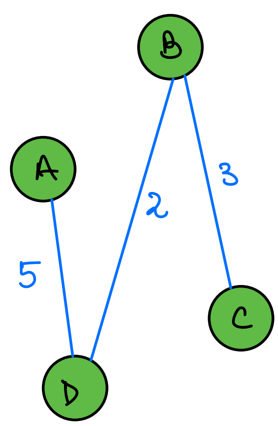


⇒ MST

weight = 17

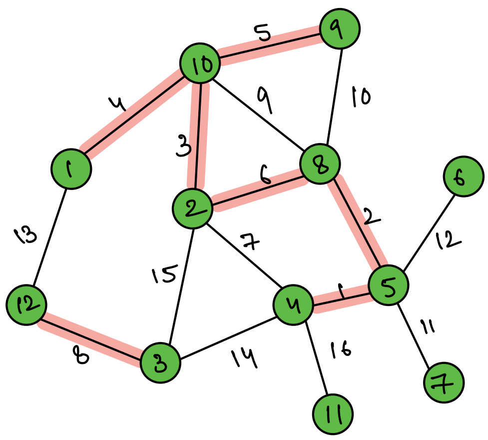


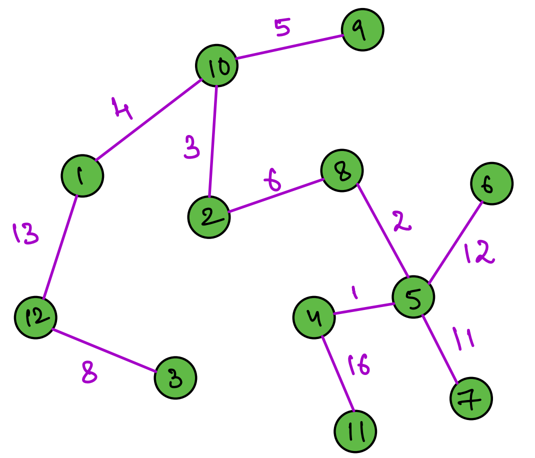
→



wt = 10

MST





Wt = 81.

Idea : N, E {Kruskal's Algorithm}

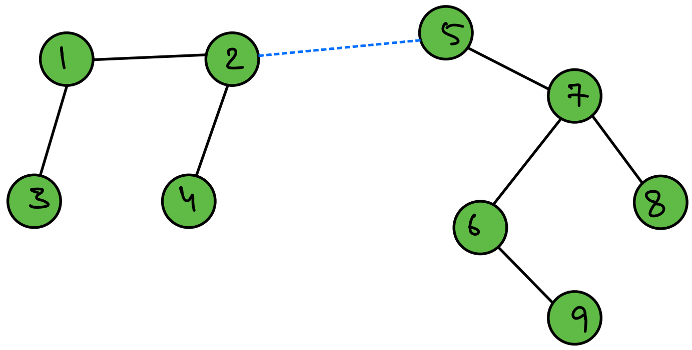
1. Sort the Edges based on the weight. → $E \log E$

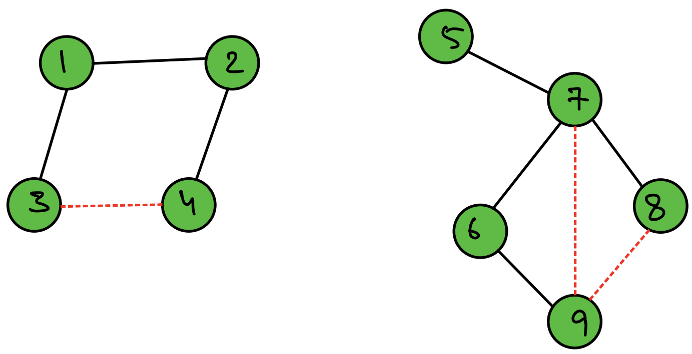
2. Pick the edge one by one, only if it is not forming a cycle in MST.
   ↳ $O(E \cdot N)$

Cycle detection : $O(N + \underset{\downarrow}{E}) \Rightarrow O(N)$
$\qquad\qquad N-1$

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

↑ We need to Optimise this.





Observations :

1. When 2 nodes of 2 different components are getting connected then NO cycle will be formed.

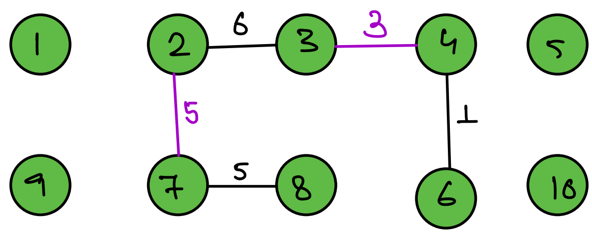
2. When 2 nodes of the same component are getting connected then cycle will be formed.

N= 10

int comp[11] :

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

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	
10	9	6	
11	1	5	
12	7	9	
14	5	10	
20	6	10	



Note : Component of smaller to larger

* Just for understanding.

Note : Insted of finding the direct component, we should find the Super Component.

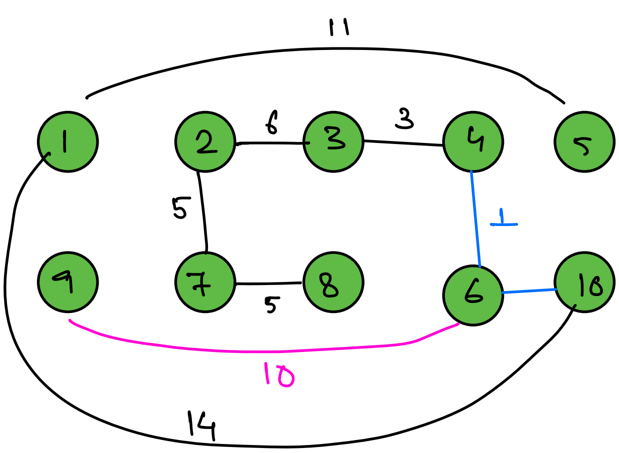
N = 10

int comp[11] :

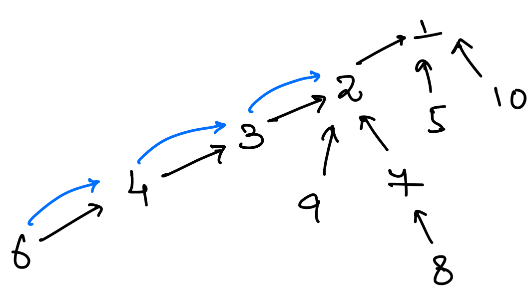
0	1	2	3	4	5	6	7	8	9	10
	1	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~
		1	2	3	1	4	2	7	2	1

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
10	9	6	9 : 2 : Comp[9] = Comp[2]
11	1	5	: Comp[5] = Comp[1]
12	7	9	⇒ 2 : 2
14	5	10	1 : 10 ⇒ Comp[10] = Comp[1]
20	6	10	⇒ 1 : 2 ⇒ Comp[2] = Comp[1]

final components of 6 & 10.



*



```
int kruskals( list<pair<int, pair<int, int>>> Edges, N){
                        w         u    v
    sort(Edges);  // sort it based on weight → E log E.
    int comp[N+1];
    for(i=1; i<=N; i++){ comp[i] = i; } → O(N)
     for(i=0; i< Edges.size(); i++){ → E
         pair<int, pair<int, int>> d = Edges[i];
         w = d.first;
         u = d.second.first;
         v = d.second.second;
O(N)      [ cu = find (u, comp); // Super Component of u
Worst Case [ cv = find (v, comp);
           if ( cu != cv ) {
Union find |
Disjoint Set      ans += w;
Union Algo.       comp[max(cu,cv)] = comp[min(cu,cv)];
           }
     }
     return ans;
}


                  → # of iterations in WC: O(N)
int find (int x, int comp[]){
    if (x == comp[x]) return x;
    return find (comp[x], comp);
}
```

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

Comp[6]=1   Comp[4]=1   Comp[2]=1

find(6) → find(4) → find(2) → find(1)

1   1   1

All these nodes have component = 1

Comp[6] = 1

find(6) ⟶ find(1)

1

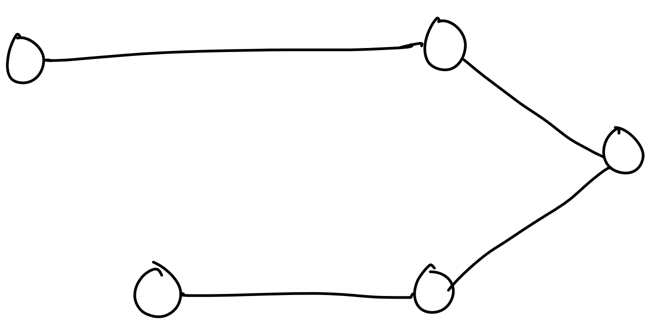
* Optimised find() funⁿ :-

O(1) Avg case.

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

TC: $O(E \log E + E * 1)$

*



MST

— * —