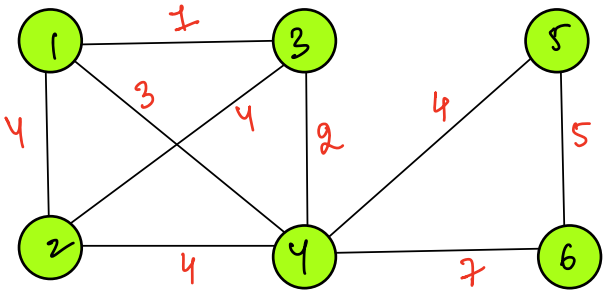
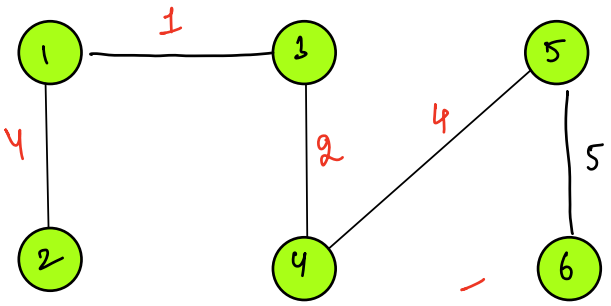


every city/node should be connected

min cost

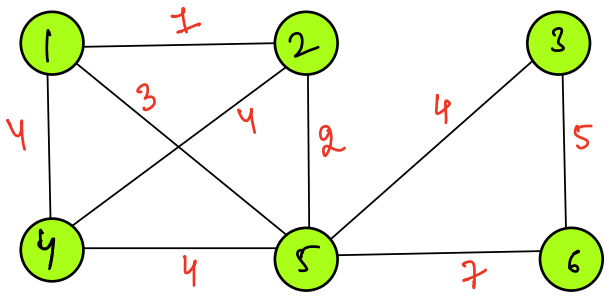


tree → n nodes → n-1

cost = 16

Minimum Spanning Tree

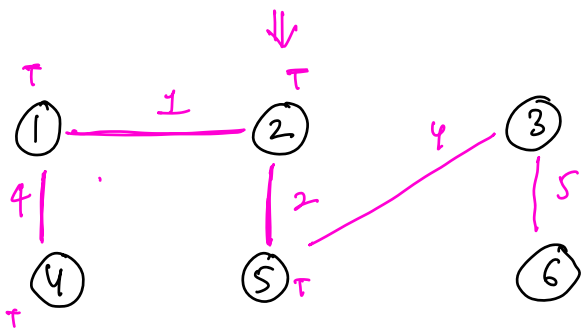
1) Kruskal's algorithm

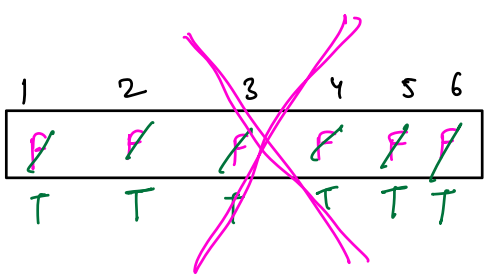


sort in asc order of weight

✓ 1 – 2	1
✓ 2 – 5	2
✗ 1 – 5	3
✓ 1 – 4	4
2 – 4	4
4 – 5	4
✓ 5 – 3	4
✓ 3 – 6	5
✗ 5 – 6	7

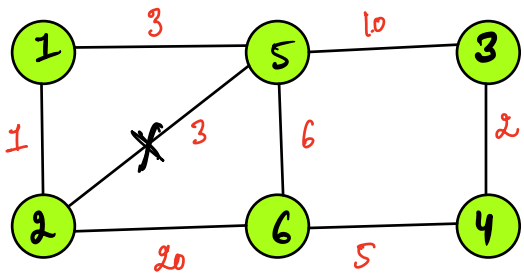
6 vert
5 edges

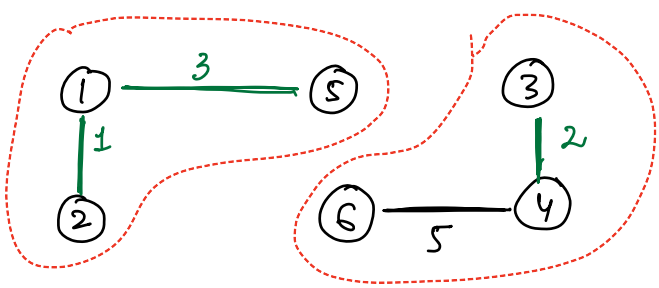


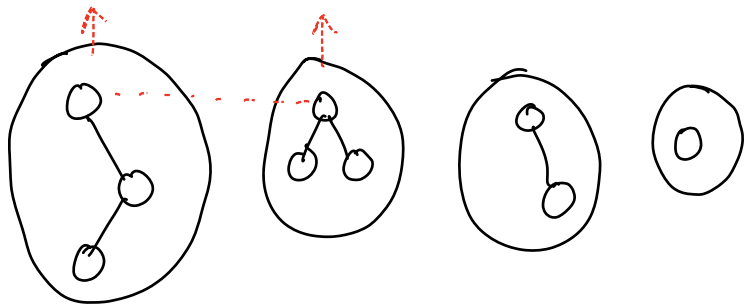


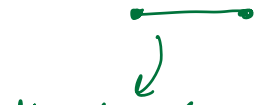
MST graph

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

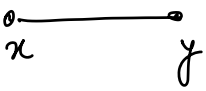








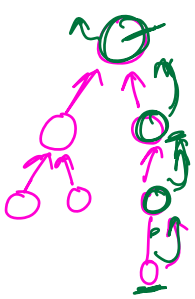
If edge connects nodes of same component, you should not pick it.

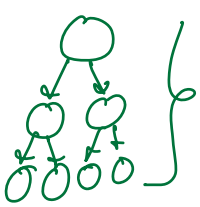


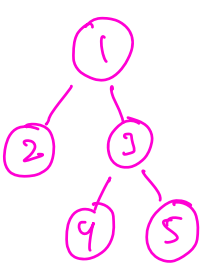
① given edge → does it connect two diff components or the same component.

② merge those 2 components

uniquely define component → tree itself → root



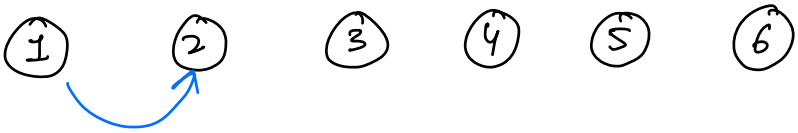




node == parent[node] ⇒ root

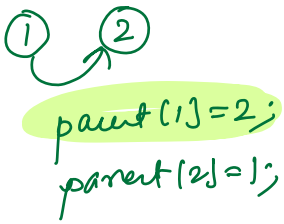
1	2	3	4	5
1	1	1	3	3

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



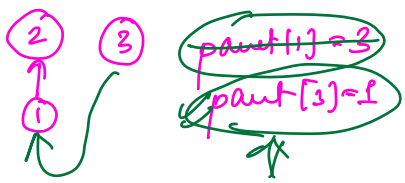
1-2

root(1) ≠ root(2)
1 2



1-3

root(1) ≠ root(3)
2 3



5-6

root(5) ≠ root(6)
5 6

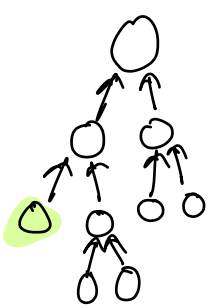
parent[5]=6

3-5

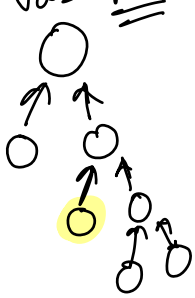
root(3) ≠ root(5)
2 6

parent[3]=5
parent[5]=3

Akbar



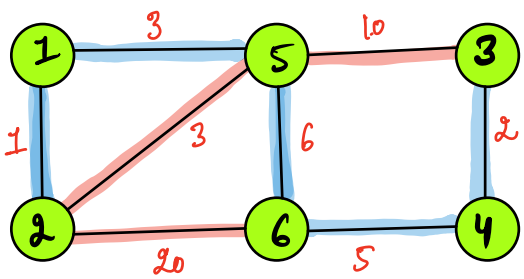
vascoda

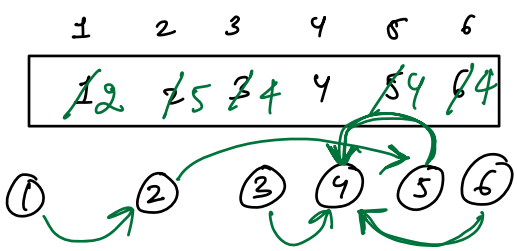


$x \longrightarrow y$

$root(x) \neq root(y)$

$parent[root(x)] = root(y)$ or $parent[root(y)] = root(x)$ — merge





1-2 : $root(1) \neq root(2)$; $parent[root(1)] = root(2)$

2-4 : $root(3) \neq root(4)$; $parent[root(3)] = root(4)$

1-5 : $root(1) \neq root(5)$; $parent[root(1)] = root(5)$

2-5 : $root(2) = root(5)$; ✗

6-4 : $root(6) \neq root(4)$; $parent[root(6)] = root(4)$

5-6 : $root(5) \neq root(6)$; $parent[root(5)] = root(6)$

find(x) → root of x

union(x, y) → find if x & y are in the diff component & yes, if diff, it will merge

Disjoint set union

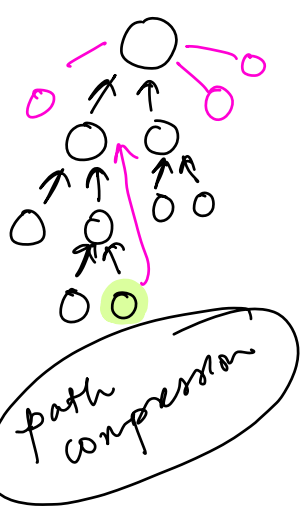
parent[n+1]

```
int find(int x)
{
    while(parent[x] != x)
    {   x = parent[x];
    }
    return x;
}
```

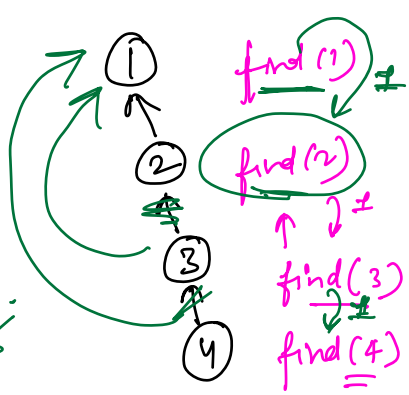
T.C: O(H) ≈ O(N) → worst case

```
bool union(int x, int y)
{
    int root_x = find(x);
    int root_y = find(y);
    if(root_x != root_y)
    {
        parent[root_x] = root_y;
        return true;
    }
    return false;
}
```

T.C: O(n)




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



T.C: O(H)

O(1) amortized

sort your edges on the basis of cost

list < pair < int, pair < int, int >> > edges;

cost, node, node2

```
for( i=0; i< edges.size(); i++)
{
    { cost, { x, y } } = edges[i];
    if( union(x, y))
    {
        total_cost += cost;
    }
}
```

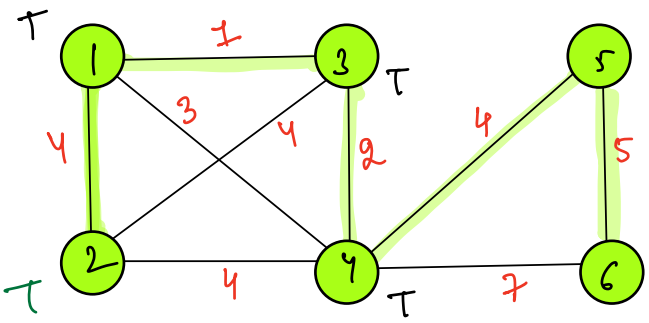
T.C: $E \log E + E * o(1)$

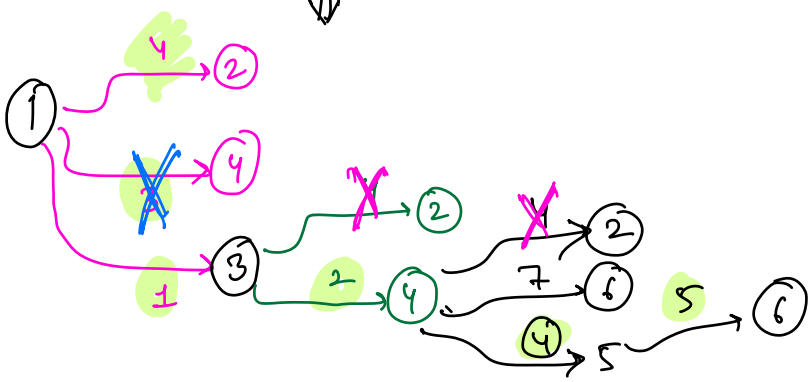
$= E \log E$

DSU = no of connect components

↳ cycle detection

Prim's





insert options in heap

⇓

get the best out of heap

⇓

If already not visited

- update your cost
- include more options
- mark node visited

```cpp
// PRIM'S ALGORITHM

bool visited [N + 1]; // Initialise with false

ordered_set<int , int > s; // TreeSet in java , can also be min heap

int total_cost = 0;
s.insert({ 0 , 1 });
visited[1] = true;

while(!s.empty()){
    {cost , u} = s.begin(); // get fist value from the TreeSet
    s.erase(s.begin());
    total_cost += cost;

    for(int i = 0 ; i < graph[u].size(); i++){
        {v , edge_cost} = graph[u][i];

        if(!visited[v]) {
            visited[v] = true;
            s.insert({edge_cost, v});
        }

    }
}
```