Todays Content

→ Cycle detection in Undirected Graph

→ Minimum Spanning Tree

→ Kruskals

→ Union find Algorithm

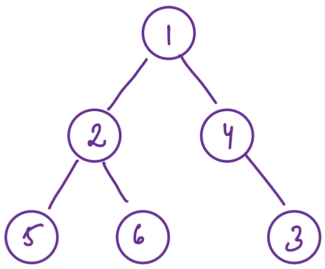
CPY

- → graphs
- → dp
- → Hashing
- → Trees
- → BS

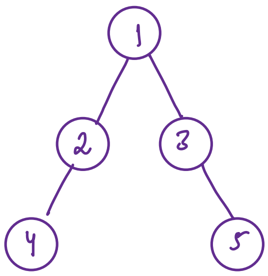
Cycle detectim in Undirected Graph

→ In Tree with N Nodes, how many edges = **N-1 Edges**
& **1 Component**

Ex:



5 Edges



4 Edges

// Given graph with N Nodes, & components?

# Nodes	# Components	# Edges it should have so that there is no cycle
N	1	N-1
N	2: $C_1 : x : x-1$ Edges $C_2 : y : y-1$ Edges	$= \underbrace{x+y}-2 =$ $= N-2$
N	3: $C_1 : x : x-1$ Edges $C_2 : y : y-1$ Edges $C_3 : z : z-1$ Edges	$= x-1+y-1+z-1$ $= \underbrace{x+y+z}-3$ $= N-3$

obs

$\underline{N}$ $\quad$ $\underline{C}$ → # Edges we need to have so that there is no cycle = N-C

Co1: Given a undireted graph with N Nodes & E Edges, check cycle, Calculate no: of components in graph = c

No cycle if Edges $E = (N-c)$ ⇒ TC: $O(N+E)$

↳ Obs1: If Total Edges $E >= N$, {100% there is cycle}

Pseudocode:

```
Step1: If $E >= N$ {return True}
Step2: Calulate no: of components, $(E == N-c)$
    ↳ TC: $O(N+E)$, $E < N$        TC: $O(N)$
```

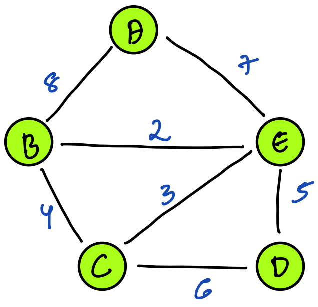
Minimum Spanning Tree

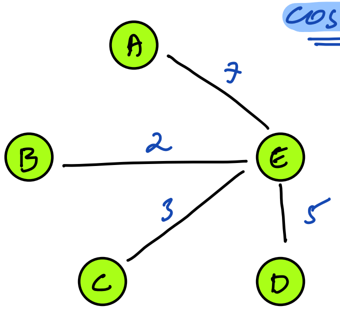
Given a undirected weighted connected graph, Convert into a tree with Minimum Weight,

{Sum of overall weights should be min},

Above tree is called Minimum Spanning Tree

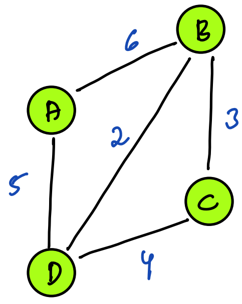
Ex1:

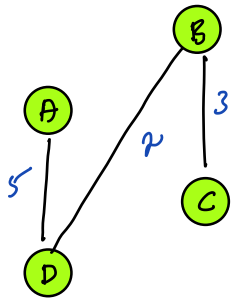




cost = 17

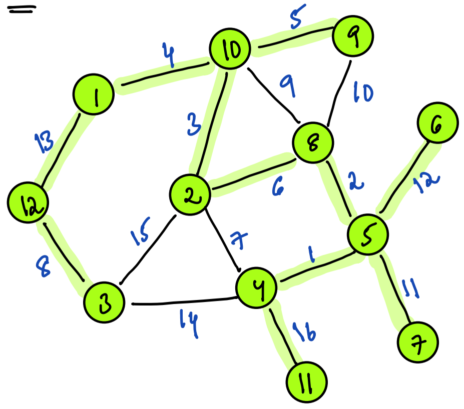
Ex2:

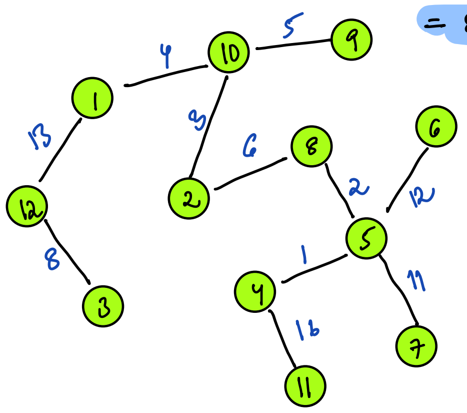




cost = 10

Ex3:





= 81

Idea: {Kruskals}

Step 1: Sort all Edges based on weight → TC: E log E

Step 2: [Add Edges 1 by 1 to graph ⟶ TC: E N]

Note: If a particular edge forms a cycle, skip edge, don't add

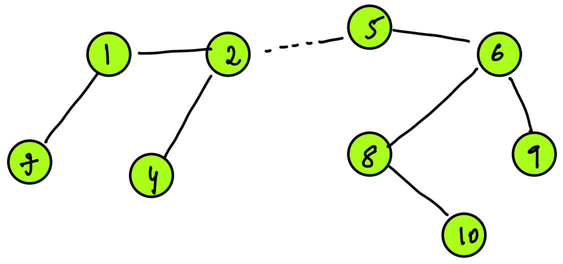
[After adding we need to check for cycle] → [We need to optimize]

TC: E * (N)

Final TC: (E log E + E * N)

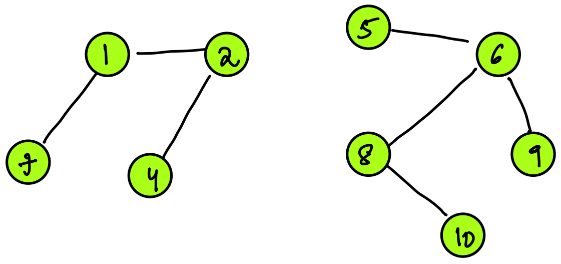
After adding 1 Edge Optimization Cycle detection

Ex1.



obs1: When 2 nodes of 2 different components are conneted, it forms a single component

Ex2:



obs2: When 2 nodes of same components are conneted, it forms cycle

//Given N= 10

W Edges

cmp[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

1 4-6 : cmp[6] = cmp[4]

3 3-4 : cmp[4] = cmp[3]

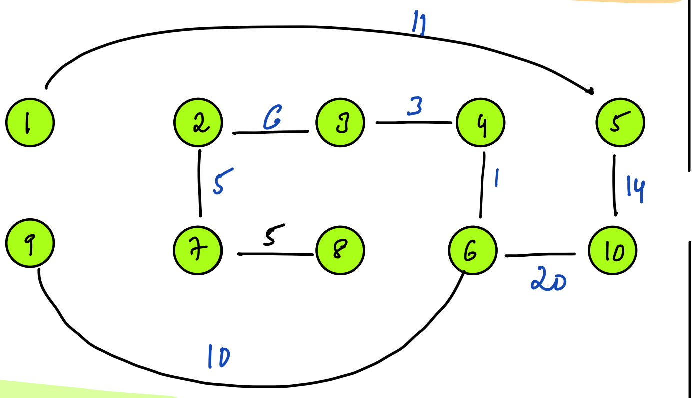
5 7-8 : cmp[8] = cmp[7]

5 2-7 : cmp[7] = cmp[2]

6 3-2 : cmp[3] = cmp[2]

8 6-8 : 2 : 2

rule: assign smaller component → larger



10 9-6 : 9 : 2 : cmp[9] = cmp[2]

11 1-5 : 1 : 5 : cmp[5] = cmp[1]

12 7-9 : 2 : 2

14 5-10 : 1 : 10 : cmp[10] = cmp[1]

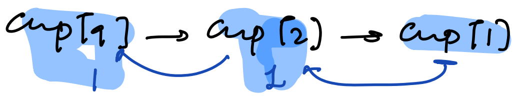
20 6-10 : 2 : 1 : cmp[2] = cmp[1]

25 6-8 : 1 : 1

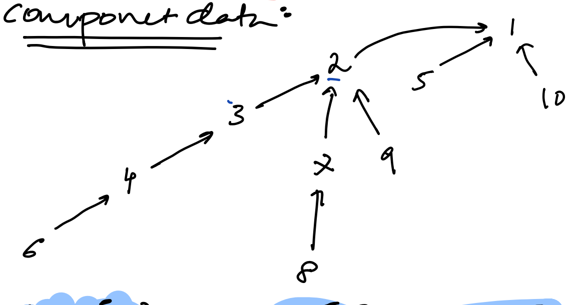
30 6-5 : 1 : 1

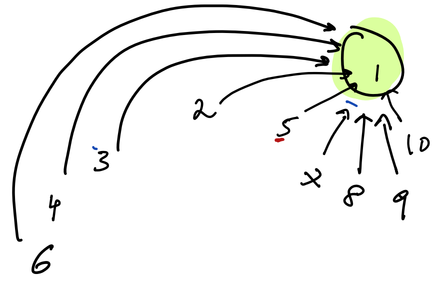
35 9-6 : 1 : 1

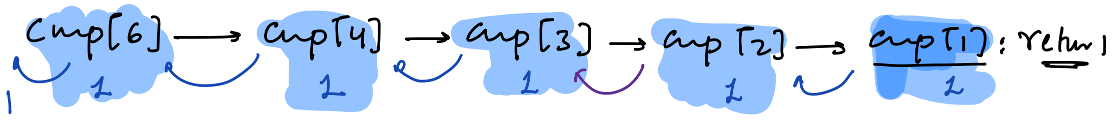
//we can skip



component data :







```
                        w              u   v
int kruskals (list< pair<int, pair<int, int>> Edges, int N){
        sort ( Edges) // sort edges based on weight → ElogE
        int cmp[N+1];
        int ans = 0;
        for(int i=0; i<=N; i++){ cmp[i]=i }
        for(int i=0; i< Edges.size(); i++){
            pair<int, pair<int, int> data = Edges[i]
            int w = data.first
            int u = data.scond.first , v= data.scond.second
            // union find algo → detecting cycle in optimized manner
              //If u & v belong 2 different comp
              int cu = findc(u, cmp) ] // find super component of
              int cv = findc(v, cmp) }   given node
                                        → TC: O(N)
              if (cu != cv){
                 // assign lower compone to higher component
                 cmp[man(cu,cv)] = cmp[min(cu,cv)]
                 ans = ans+w // Edge from u→v considered
              }
        }
        return ans;
}

int findc(int x, int cmp[]){ TC→O(N) --optimized--> O(1)
    if(x == cmp[x]) return x
    cmp[x] = findc(cmp[x], cmp)  // before return update cmp[x]
                 ↳ find component of parent
    return cmp[x]
}
```

Overall TC → ElogE + E*N

↳ After changing find(

Overall TC → ElogE + E O(1)

Minimum Spanning Tree → prim's 15mins TODO

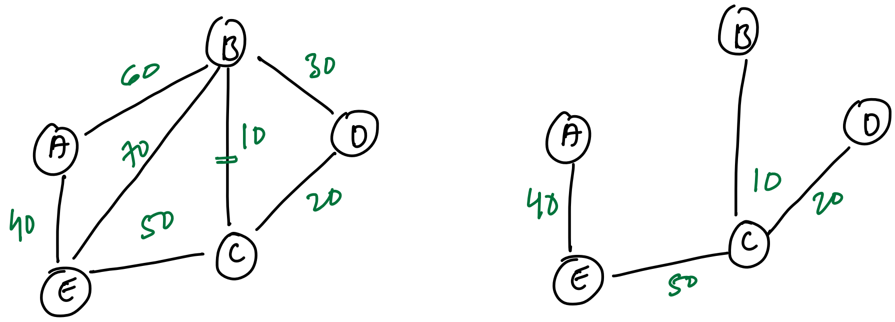
↳ Kruskals : 1) Sort all Edges

2) Add Edge by Edge, after adding an edge check, if cycle there or not

↳ To optimize union / find

A single union : O(1)
A find union : O(1)

Use Case:


B
60
30
A
70
10
D
40
50
20
C
E
B
A
D
10
40
20
C
E
50