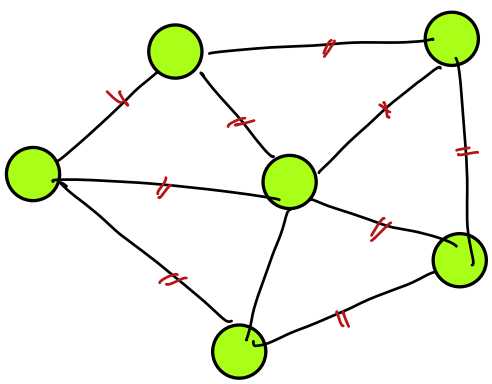
Today's Content :

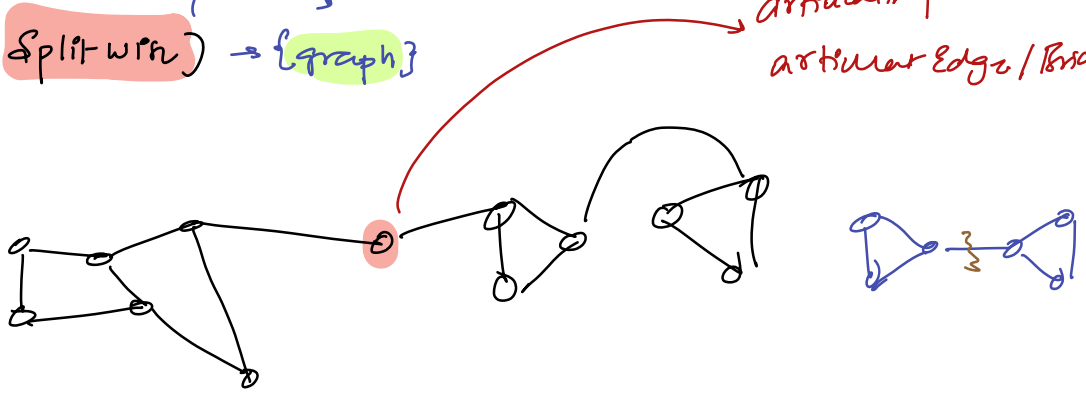
- Minimum Spanning Tree
  - Prims
  - kruskals
- Tue - graphs
- Thu - [double]
- Sat - [double]
  - last session

//



- Construct : City → City Board : Connect
- Min Cost
- MST :- very used in real world

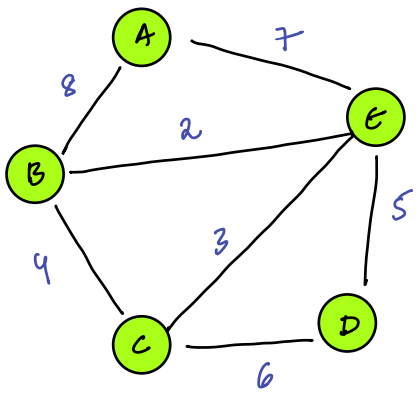
(Split-wise) → {graph}



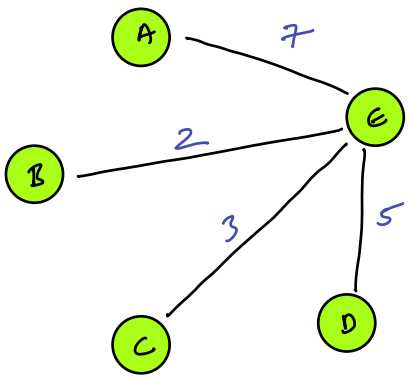
Minimum Spanning Tree : → of In Tree of N Nodes → No Cycle
↳ N-1 Edges

→ Given a undirected weighted Connected graph, Convert it into a Tree with Minimum Weight
↳ Sum of all edge weights should be min

Eg1:

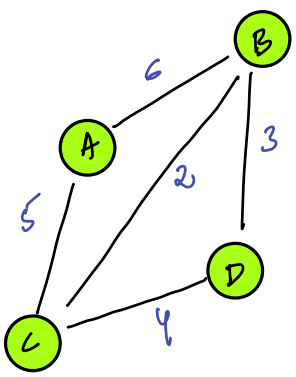


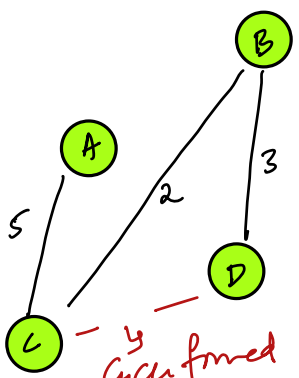
Cost = 17



Cost :
8, 10, 14

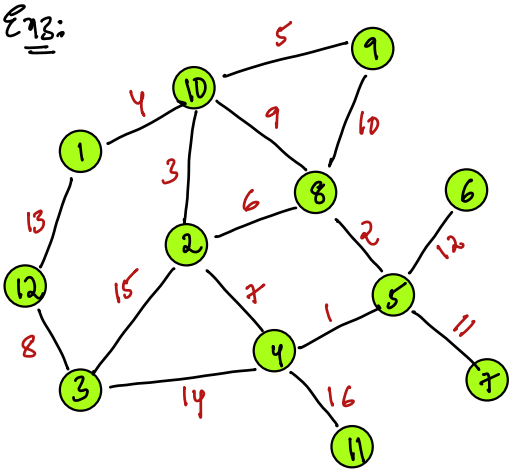
Eg2:

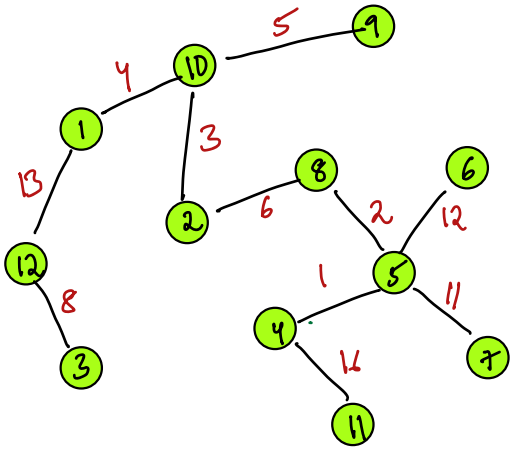




→ ans = 10

Eg:





// All Edges : → {Edges sorted by weight}

u	v	w
1	10	4
1	12	13
2	10	3
2	3	15
3	12	8
3	4	14
2	4	7
2	8	6
4	5	1
4	11	16
5	7	11
5	8	2
5	6	12
8	10	9
8	9	10
9	10	5

u	v	w	
4	5	1	
5	8	2	
2	10	3	
1	10	4	
9	10	5	
2	8	6	
2	4	7	{skipt}
3	12	8	{cycle}
8	10	9	
8	9	10	
5	7	11	
5	6	12	
1	12	13	
3	4	14	
2	3	15	
4	11	16	

→ Idea:
↳ (Kruskals Algorithm)

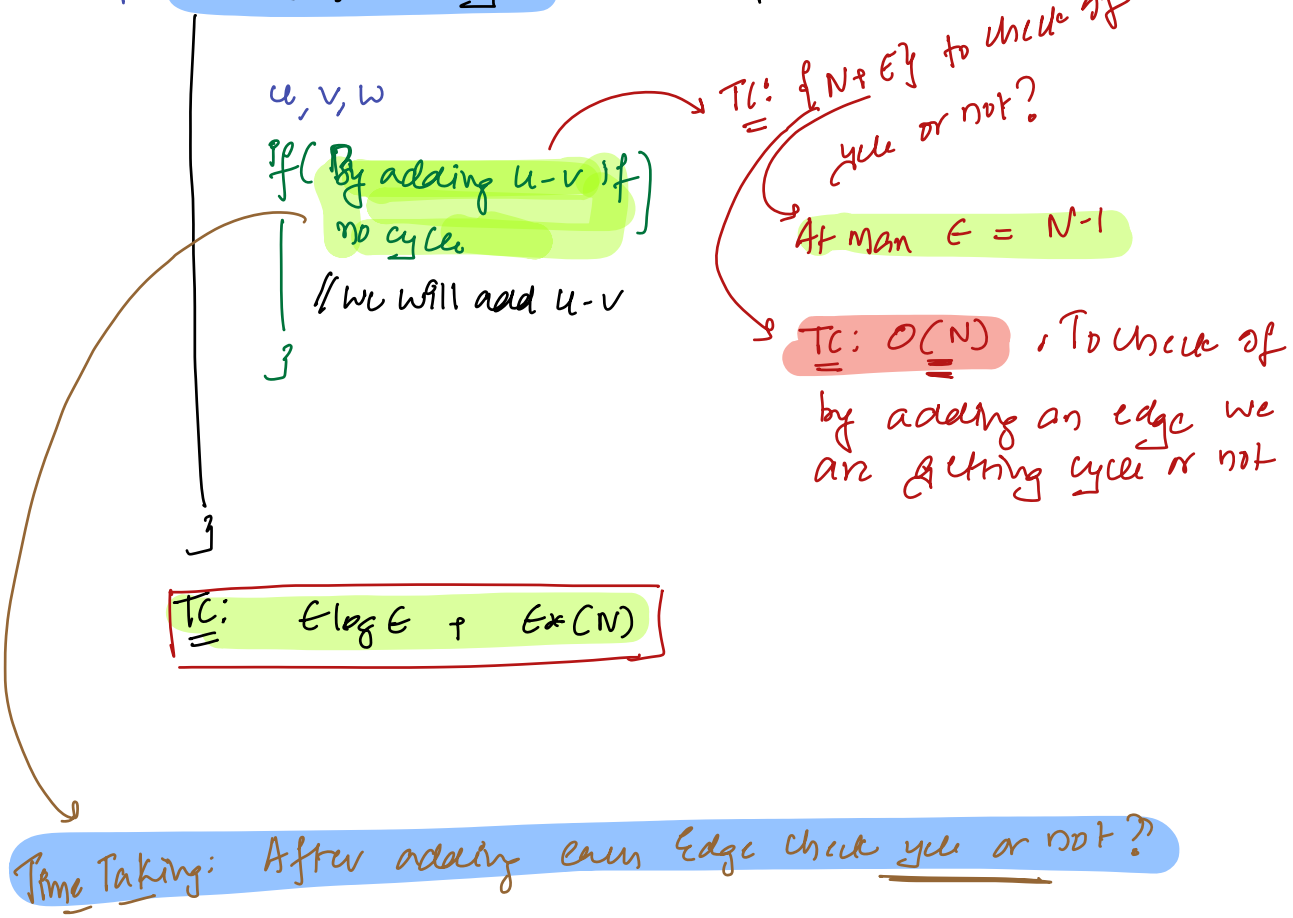
a) Sort all Edges based on increasing order of weights

b) Add every Edge by Edge in increasing order of weight
→ If edge doesn't form a cycle only that it can be added

→ All Edges: list< int, pair<int, int>>

→ Step: Sort based on weight: $E \log E$ →

* Steps: Iterate on all Edges: → Do it for Every Edge

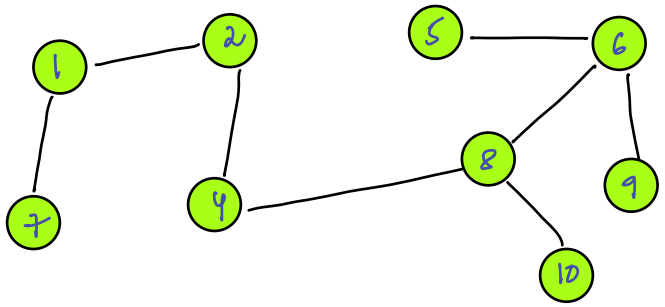


Time Taking: After adding each Edge check cycle or not?

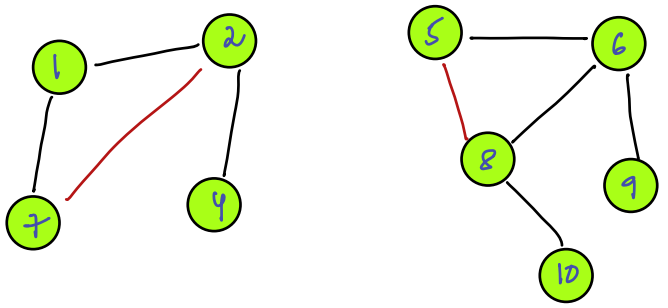
Optimization Cycle detection

Add edge: 4 - 8

Ex1:



Ex2:



Add edge: 5 - 8

Add Edge: 2 - 7

Obs1: When we add edge between 2 components cycle not formed & both of them will become a single component

Obs2: When we add edge between 2 nodes of same Component's Cycle is formed

Ex: N = 10

	1	2	3	4	5	6	7	8	9	10
cmp[11] =	1	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	10
	1	2	3	1	4	2	7	2	.	

Edges:

4 - 6

3 - 4

7 - 8

2 - 7

3 - 2

6 - 8 → color = 2

color = 2

9 - 2

1 - 5

7 - 9 → {don't add}

5 - 10 → {c[10] → 1}

6 - 10 : c[2] = 1

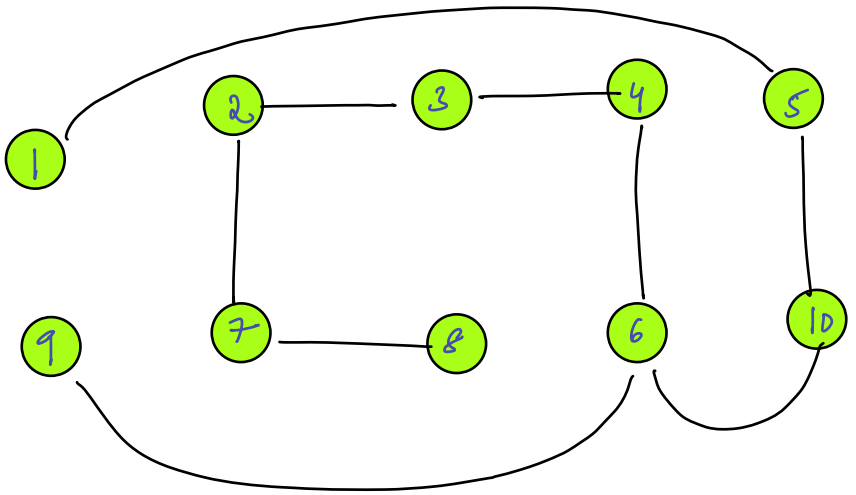
higher compo

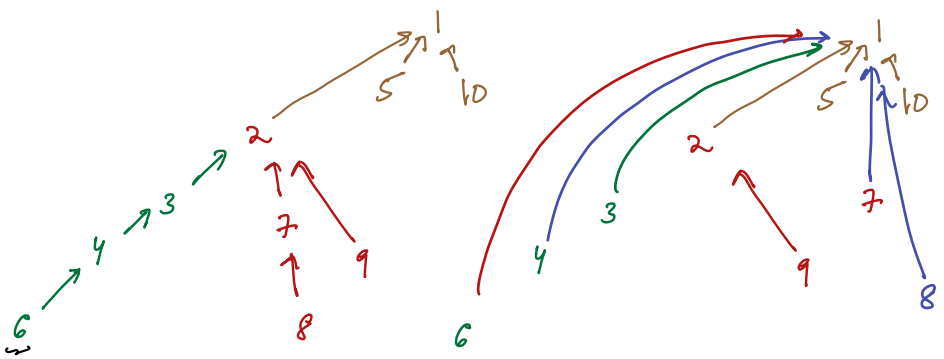
lower compo

6 - 8

6 - 5

Consistent: Assign low component → higher Component





cmp(6) → cmp(4) → cmp(3) → cmp(2) → cmp(1) : return

cmp[6] = 1

cmp[4] = 1

cmp[3] = 1

cmp[2] = 1

// PseudoCode

Sort all Edges

c[N+1];

i=1; i<=N; i++){ c[i]=i }

Iterating on all Edges:

→ u v w

P_u = comp(u, c)

P_v = comp(v, c)

if (P_u != P_v){

// both u & v belong to different component

// Means we can take Edge from u-v

// update u-v in your ans list

c[max(P_u, P_v)] = min(P_u, P_v)

} }

{Path compression}

```
int comp(int n, int c[]){
    if(c[n]==n) {return n}
    c[n] = comp(c[n], c)
    return c[n]
}
```

→ Code, cycle there are not, after adding every edge

finding component number — union

→ union find Algo

→ union find
TC: O(1) — O(N)

Small Optim.
Union + find + Path Compression
O(1) — {O(1)} → TC: O(α) ← function of N == 1
Average: O(1)

// Overall Elog E + N + E(O(1) + O(1))

→ TC: Elog E

// given a undirected graph check cycle is there or not?

1) TC: O(N+E) → BFS, components

2) Union-Path & Path Compression — { After adding every edge we will know cycle there or not }

c[N+1];

i=1; i<=N; i++){ c[i] = i }

Iterating on all Edges:

→ u v w

P_u = comp(u, c)

P_v = comp(v, c)

if (P_u != P_v) {

// both u & v belong to different component

// means we can take Edge from u-v

// update u-v in your ans list

c[max(P_u, P_v)] = min(P_u, P_v)

}

else {

// cycle is formed

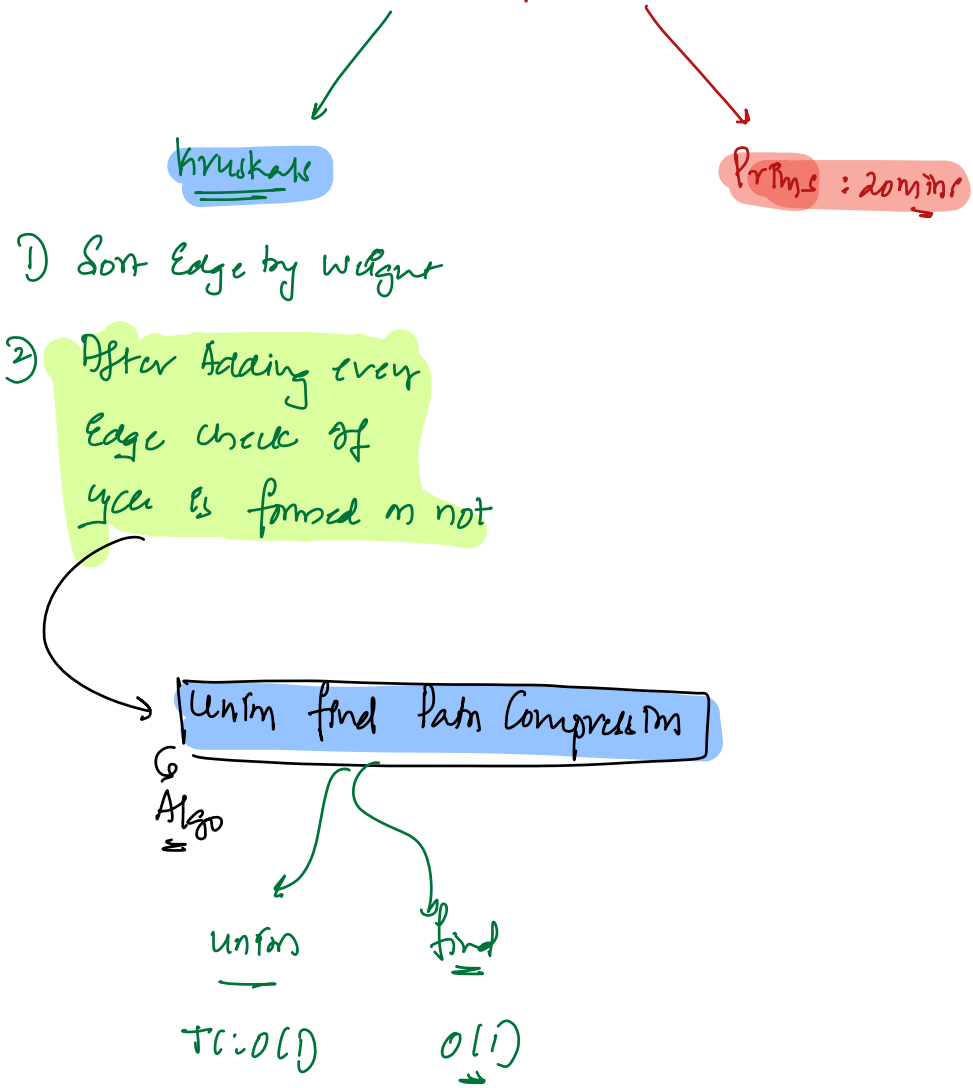
}

}

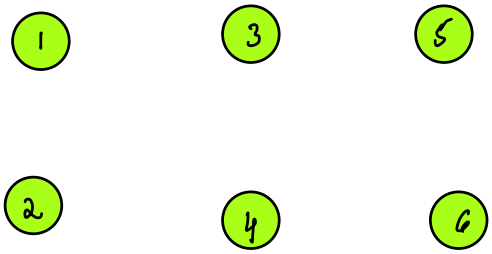
```
int comp(int n, int c[]) {
    if(c[n]==n) {return n}
    c[n] = comp(c[n], c)
    return c[n]
}
```

TC: (N + E) SC: O(N)

→ MST: (Minimum Spanning Tree)



Prims: → { Tuesday first thing }

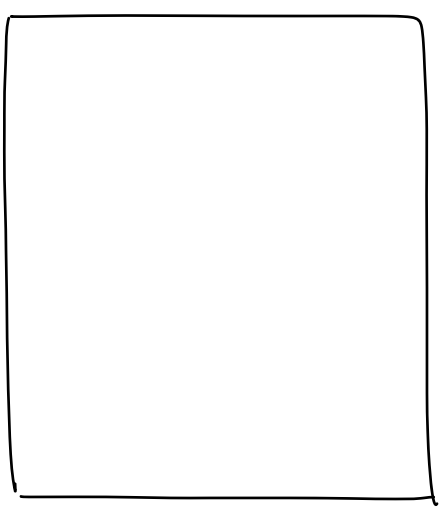


Tuesday:

→ Prims / Bipartite /

→ { Strongly Connected Comp and Directed graph cycle detn using DFS
(or)
.
Bellman Ford and Floyd Warshal }

→



→ Party → Startup

↳ D Dojo —

satyasai.2407@gmail.com

rama@scaler.com