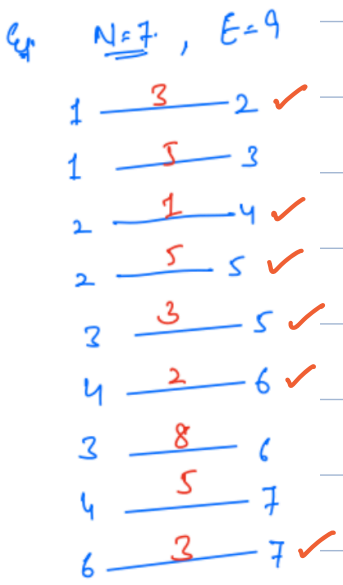
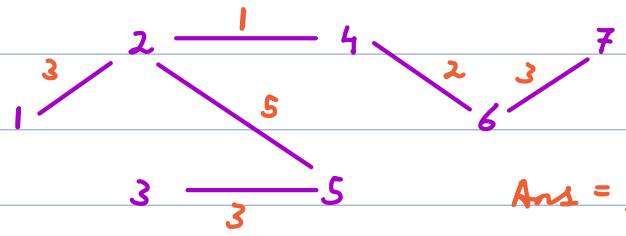
Scenario:
Suppose Flipkart has N local distribution centers spread across a large metropolitan city. These centers need to be interconnected for efficient movement of goods. However, building and maintaining roads between these centers is costly. Flipkart's goal is to minimize these costs while ensuring every center is connected and operational.

Goal: You are given number of centers and possible connections that can be made with their cost. Find minimum cost of constructing roads between centers such that it is possible to travel from one center to any other via roads.

Example:





Ans = 17

cost ∝ #roads

⇒ min #roads required = N-1

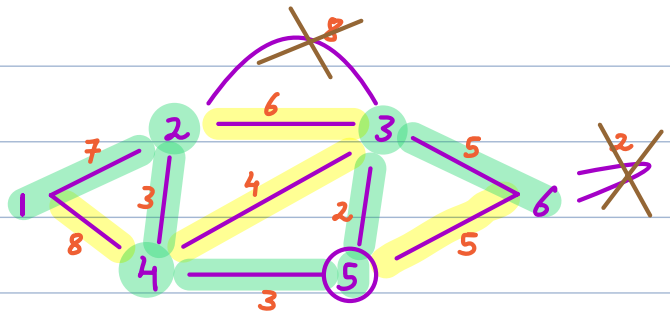
⇒ connected graph with N nodes & N-1 edges = tree

Minimum Spanning Tree (MST)

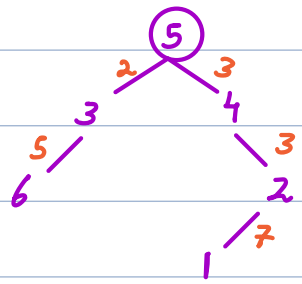
1) Multiple MST possible for a graph.
If wt are unique ⇒ unique MST.

2) Prim's Algo ✓
Kruskal's Algo (DSA 4.2 Module)

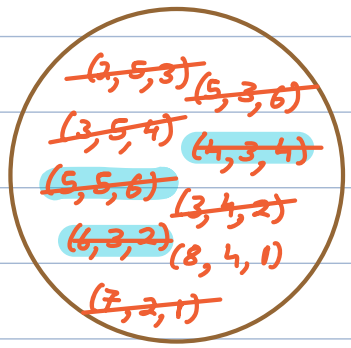
Prim's Algo (Greedy)



1) Ignore self loops & multi edges.
2) Start with any node as root node.
3) Insert the connected edges of selected node edge in min heap (wt, u, v). (wt: weight)
4) Select the min wt edge that connect a node & repeat from step 3 till all nodes are connected.



Ans = 20



minHeap

```
// minHeap → h
∀ i, vst[i] = false          cost = 0
vst[1] = true   // root
for (e : Adj[1]) {    // e.wt, e.v
    h.add ({e.wt, 1, e.v})
}
while (! h.isEmpty ()) {
    t = h.getMin ()   // {wt, u, v}
    if (vst[t.v])   continue
    vst[t.v] = true      cost += t.wt
    for (e : Adj[t.v]) {    // e.wt, e.v
        if (! vst[e.v])
            h.add ({e.wt, t.v, e.v})   // t.v — e.v
```

```
    }
}
return cost
```

$TC = O(E \log(E))$

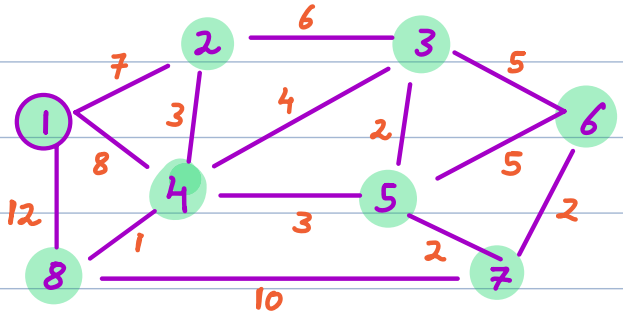
$SC = O(E + N)$ // $E >= N-1$

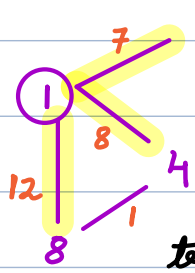
$= O(E)$

Dijkstra's Algo (Single Source shortest Path)

o/p → d[i] → min distance of node 'i' from source node.

d[1] = 0

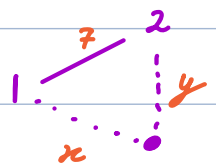




If min wt edge from 1 is (7, 2)

⇒ min distance to travel to 2 = 7 ?

Yes



if ((x + y < 7) && x > 0 && y > 0)

⇒ x < 7

1	2	3	4
d = [0	7	12	8
5	6	7	8
11	15	13	9]

~~(7, 2)~~ ~~(12, 3)~~
~~(8, 4)~~ ~~(10, 4)~~
~~(12, 8)~~ ~~(12, 3)~~ (16, 6)
~~(11, 5)~~ ~~(9, 8)~~ ~~(12, 7)~~
(19, 7) ~~(13, 5)~~
(17, 6) ~~(15, 6)~~

MinHeap {~~wt~~, v}

(d[u] + wt)

∀i, d[i] = Int_Max

h.add ({0, 1}) // source

```
while (!h.isEmpty()) {
    e = h.getMin()  // dist, v
    if (d[e.v] != IntMax)  continue
    d[e.v] = e.dist
    for (edge : Adj[e.v]) {  // wt, v
        if (d[edge.v] == IntMax) {
            h.add({ d[e.v] + edg.wt, edge.v })
    }
}
return d
```

SC = O(E)

TC = O(E log(E))