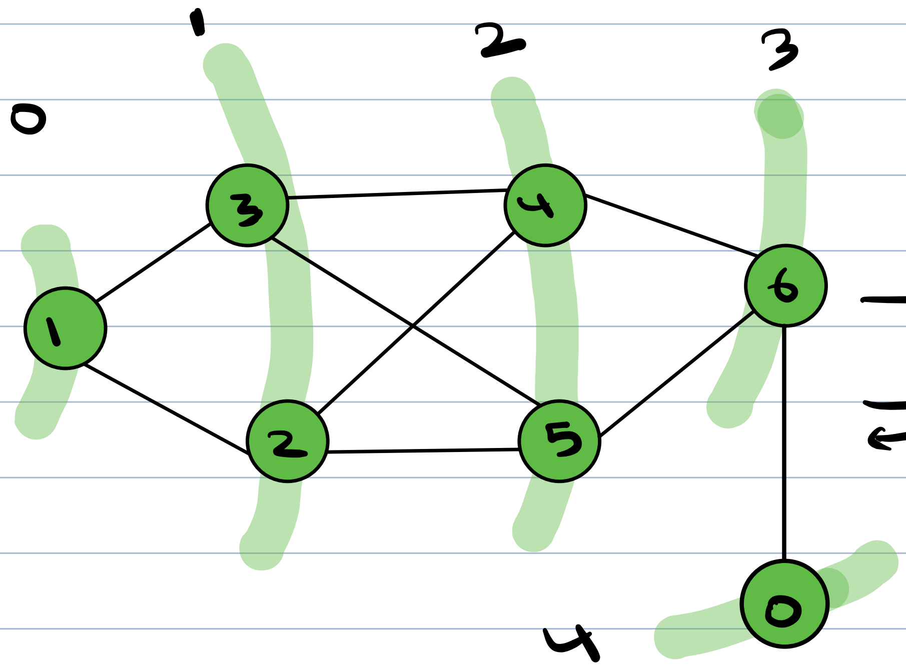
Agenda

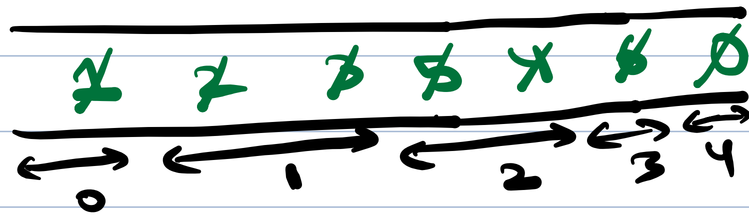
- BFS Traversal
- Multisource BFS
- Rotten Oranges
- Challenges in Flipkart Logistics
- MST : Prim's Algo

BFS : Breadth First Search

1. Level order Traversal
2. At each step, we explore neighbours
They'll provide us info about their neighbours



nodes → 0 to 6
N = 7



O/P : 1 2 3 5 4
6 0

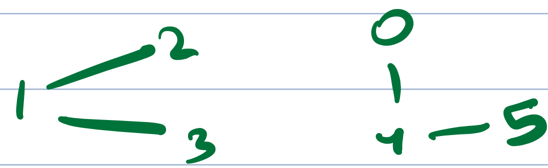
visited [7]

0	1	2	3	4	5	6
~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T

1 $\overset{2}{\text{———}}$ 5 $\overset{1}{\text{——}}$ 6

BFS : Shortest path from start node to all other nodes (unweighted graph)

```
// N nodes -> 0 to N-1
void bfs (int start, list <int> adj[N],
bool visited [N]) {
    queue <int> q
    q.enqueue (start)
    visited [start] = true

    while (! q.empty()) {

        int cur = q.front()    q.dequeue()
        // print (cur)

        for (i=0 ; i < adj[cur].size() ; i++) {
            int nbr = adj[cur][i]
            if (visited [nbr] == false) {
                q.enqueue (nbr)
                visited [nbr] = true
            }
        }
    }
}
```




```
int main () {
    // To Do -> create adj
    bool visited [N] = {false}
    for (i=0 ; i<N ; i++) {
        if (visited [i] == false)
            bfs (i, adj, visited)
    }
}
```

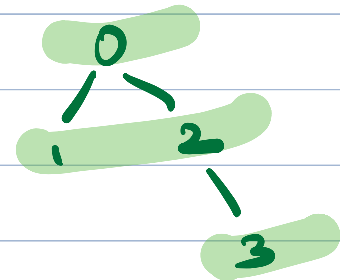
V → vertices | N → nodes

TC : O(V + 2E)

SC : O(2V) → vis[V] and queue → V

N=4 (0 to 3)

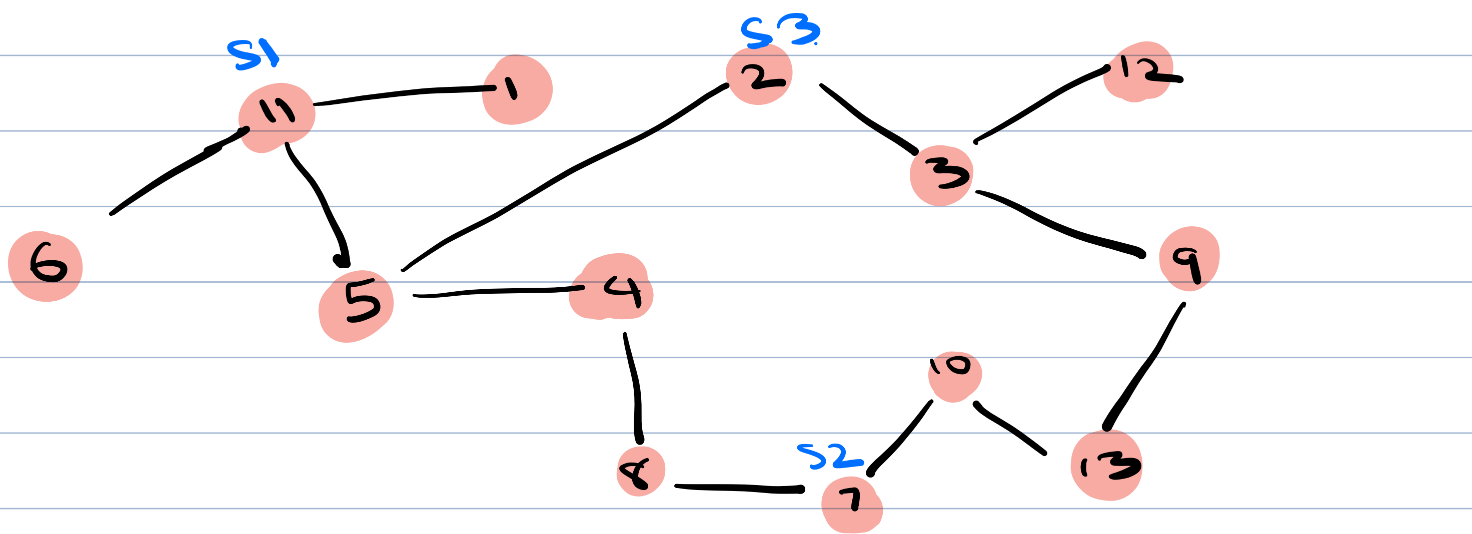
	0	1	2	3
0	0	1	1	0
1	1	0	0	0
2	1	0	0	1
3	0	0	1	0



~~0~~ ~~1~~ ~~2~~ ~~3~~

O/P : 0 1 2 3

N no. of nodes → Home, Storage Hub



Node, dist

~~(11,0)~~ ~~(7,0)~~ ~~(2,0)~~ ~~(6,1)~~ ~~(5,1)~~ ~~(4,1)~~
~~(8,1)~~ ~~(10,1)~~ ~~(3,1)~~ ~~(4,2)~~ ~~(13,2)~~
~~(12,2)~~ ~~(9,2)~~

0	1	2	3	4	5	6	7	8	9	10	11	12	13
F	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T

multisource BFS
↓
We will do BFS from all sources simultaneously

(N, 1)

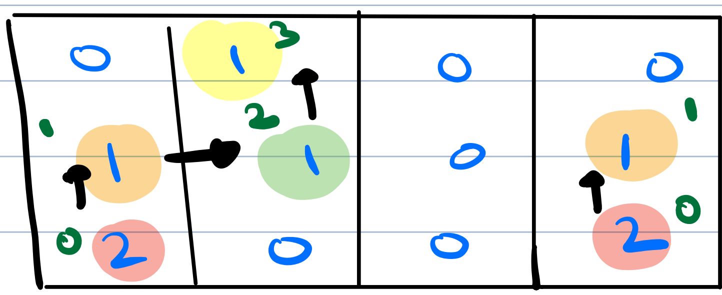
storage $\xrightarrow{1}$ N $\xrightarrow{+1}$ nbr

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

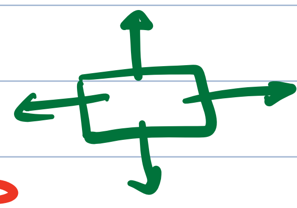
Rotten Oranges

mat[N][M] → 0 empty
→ 1 fresh
→ 2 rotten

Every minute any fresh orange adjacent to a rotten orange becomes rotten, find min time when all oranges become rotten.
If not possible, return -1.



ans=3

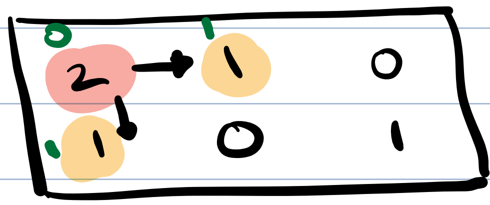


0	1	0
0	1	0

ans = -1

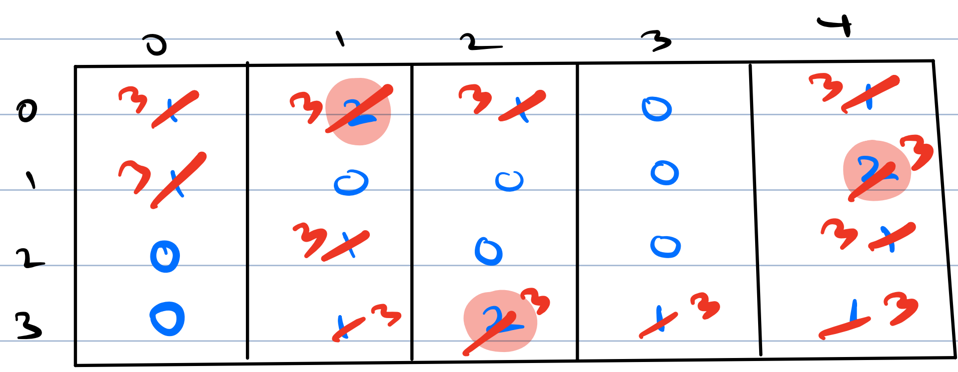
0	2
2	0

ans = 0



ans = -1

Matrix → Graph
cell → Node



(Node, time)

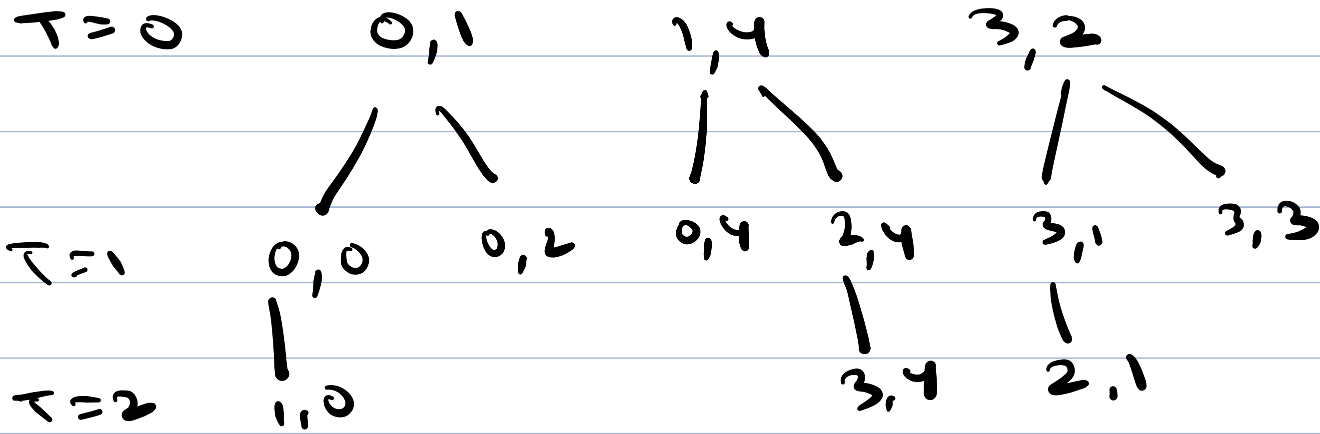
ans = 2

(i, j, t)

~~(0,1,0)~~ ~~(1,4,0)~~ ~~(3,2,0)~~ ~~(0,0,1)~~ ~~(0,2,1)~~

~~(0,4,1)~~ ~~(2,4,1)~~ ~~(3,1,1)~~ ~~(3,3,1)~~ ~~(1,0,2)~~

~~(3,4,2)~~ ~~(2,1,2)~~



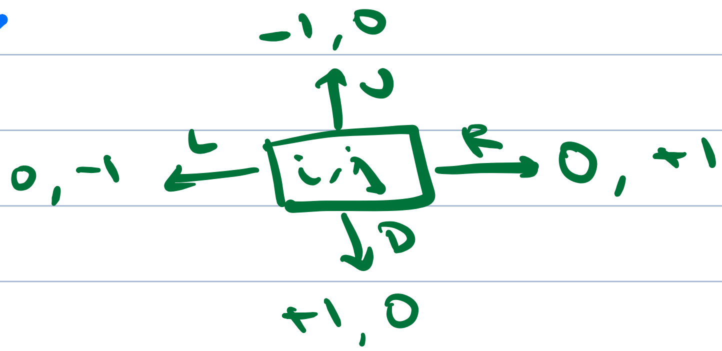

```
int rottenOranges(grid[N][M]) {

    queue<list<int>> q
                [i,j,t]

    int ans_time = 0, freshOranges = 0

    for (i=0 ; i<N ; i++) {
        for (j=0 ; j<M ; j++) {
            if (grid[i][j] == 2) {
                q.enqueue(<i,j,0>)
                grid[i][j] = 3      // visited
            }
            else if (grid[i][j] == 1) {
                freshOranges ++
            }
        }
    }
```

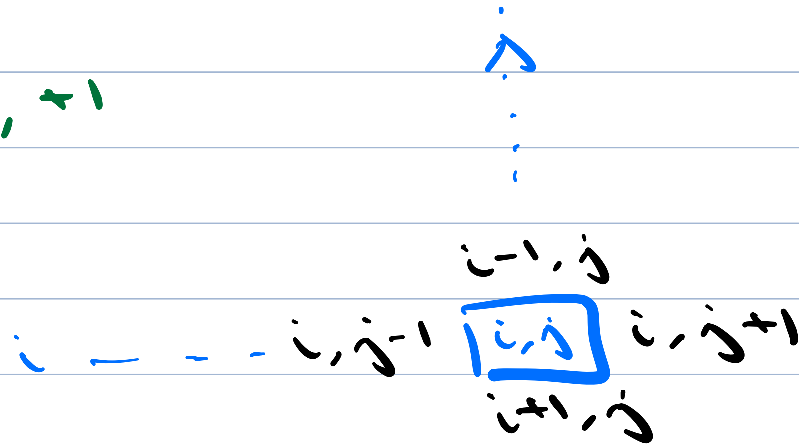
```
if (freshoranges == 0)
        return 0

while (! q.empty()) {

  list<int> cur = q.front()
  q.dequeue()

  int cur_i = cur[0], cur_j = cur[1],
  t = cur[2]

  ans.time = t

  int dx = [0, -1, 0, 1 ]
  int dy = [-1, 0, 1, 0 ]

  for (k = 0; k < 4 ; k++ ) {
    int nbr_i = cur_i + dx[k]
    int nbr_j = cur_j + dy[k]
    if (nbr_i >= 0 && nbr_i < n &&
        nbr_j >= 0 && nbr_j < m &&
        grid[nbr_i][nbr_j] == 1) {
      q.enqueue(<nbr_i, nbr_j, t+1>)
      grid[nbr_i][nbr_j] = 3
       freshoranges--
    }
  }
```

}

if (freshoranges == 0)
    return ans_time
else
    return -1

}





int dx = [0, -1, 0, 1]

int dy = [-1, 0, 1, 0]

L U R D

10:35

TC: O(V + E)

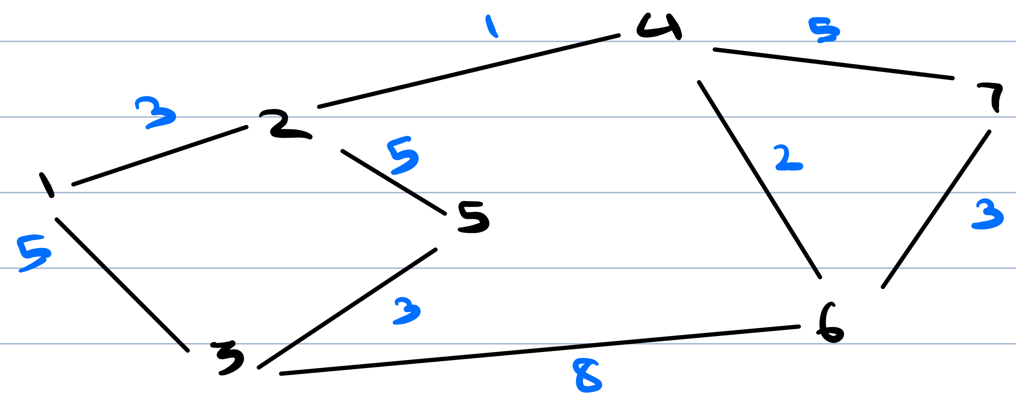
TC: O(N×M + 4×N×M)

= O(N×M)

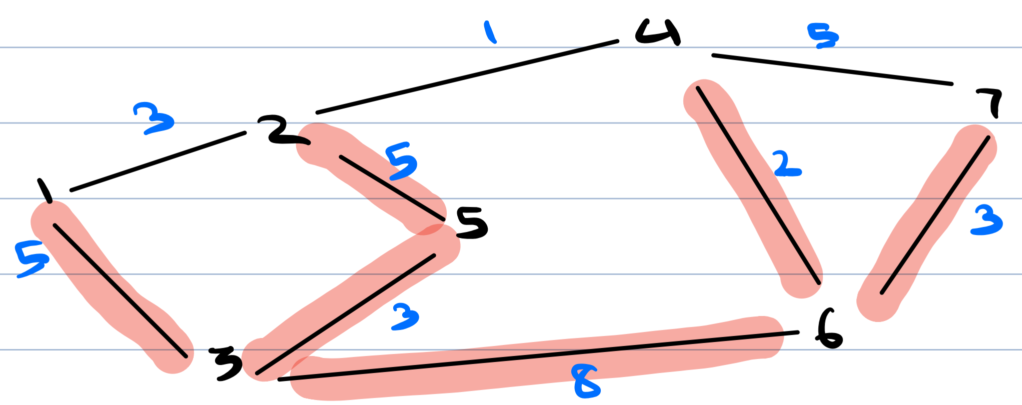
SC: O(V) = O(N×M)

1. Flipkart has N local distribution centers spread across a large city. They want to keep the centers well connected. Now some connections routes are available to you along with some cost. Find min. cost of connecting all centers.

N = 7 E = 9

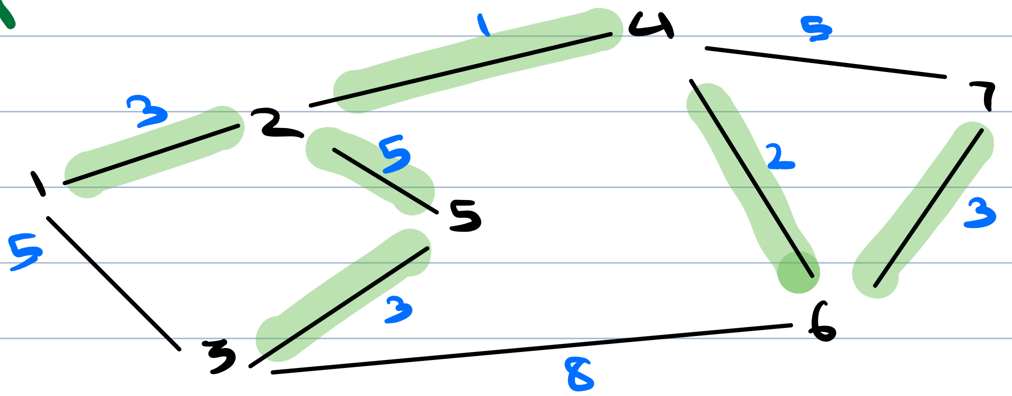


① Tree like structure
N centres → N-1 routes



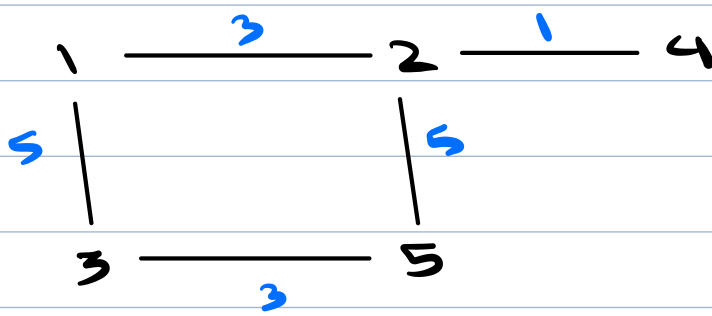
cost = 26

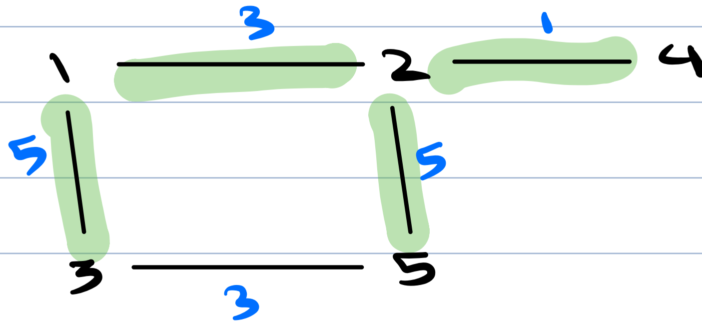
MST



ans = 17

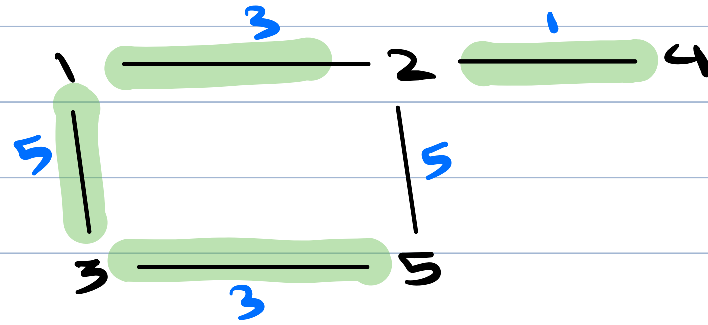
N = 5 , E = ~~4~~ 5





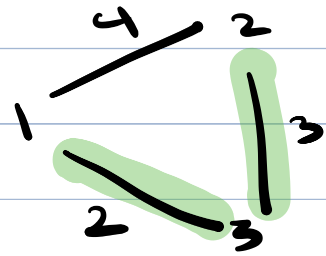
cost = 14

MST

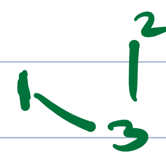


ans = 12

N = 3 , E = 3



ans = 5



The tree of N-1 Edges which spans across (covers) all vertices and connects all of them is called spanning tree.

Minimum Spanning Tree (MST) : The spanning tree with min. cost is called minimum spanning tree.

① when all the costs are unique, then we get a single unique MST

② If there is repetition in costs, then there can be multiple MSTs with min. cost.

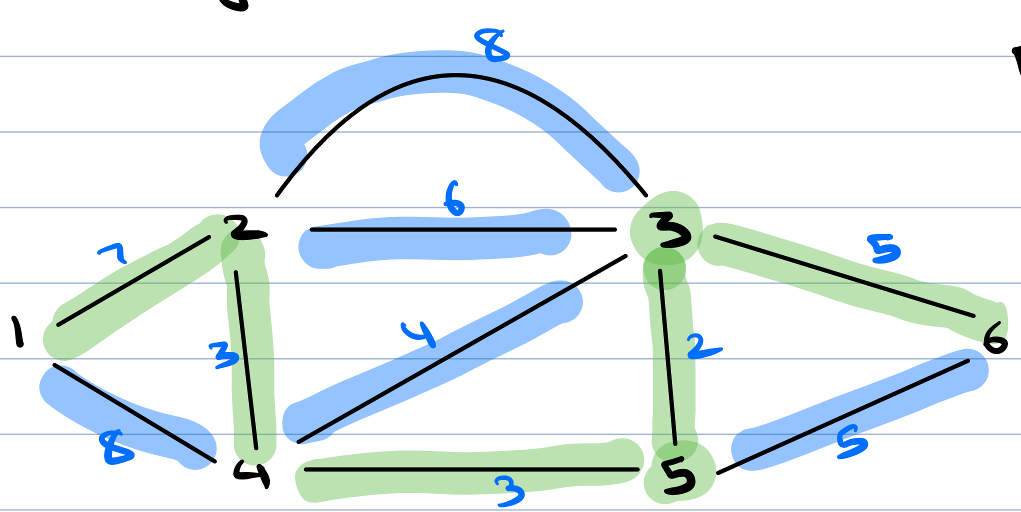
How to create MST ?

① Prim

② Kruskal (covered in DSA 4.2)

Prim's Algo

N = 6
E = 10



cost = 0

vis	1	2	3	4	5	6
	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T

1. start with 5
2. Picked 2, 3
3. cost = 2

wt	nbr
3	4
5	6
2	3

4. 3 → edges
5. Picked 3, 4

cost = 5

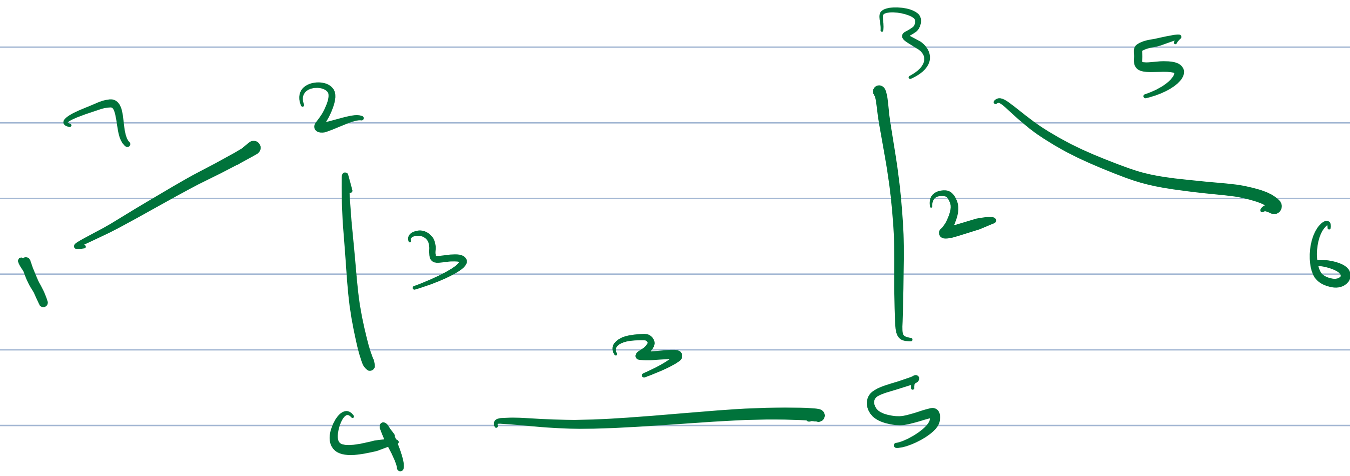
wt	nbr
3	4
5	6
2	3
6	2
8	2
4	4
5	6
3	2
8	1
7	1

6. Picked 3, 2

Cost = 8

cost = 13

cost = 20




```
int getMSTcost (int N, list<pair>
                         adj[N]){

    minHeap <pair <int, int>>  mh
                  <wt, nbr>

    int ans = 0
    bool visited [N] = false
    mh. insert ( <0,0>)
    while (! mh .empty()){

        pair<int,int> p = mh . extractMin()
        int curwt = p. first
        int cur = p.second
        if (visited [cur] == true)
                              continue;

        ans += curwt
        visited [cur] = true
```

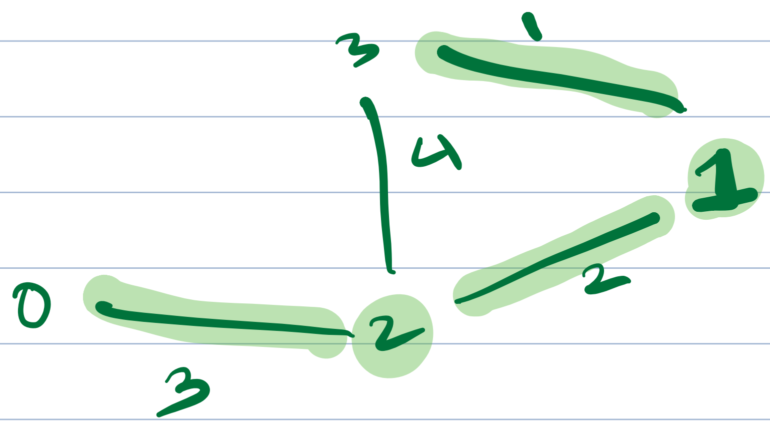
```
for (i=0; i<adj[curr].size();
i++) {
    pair<int,int> p=adj[curr][i]
    int nbr = p.first
    int cost = p.second
    if (visited[nbr] == false)
        mh.insert({cost,nbr})
    }
}

return ans
}
```

TC: O(E log E)
SC: O(V+E)

	Node
adj[N]	0 to N-1
adj[N+1]	1 to N

nbr, wt

2 : 3,6 4,3



wt, nbr

mh

vis	0	1	2	3
	T ~~F~~	~~F~~ T	~~F~~ T	~~F~~ T

ans = ~~0~~ ~~2~~ ~~5~~ 6

0,2
4,3
2,1
3,0
1,3