Today's Agenda:-

Starting 7:05

1. Minimum Spanning Tree
   a) Kruskal's Algo
   b) Prim's Algo
2. Dijkstra's Algorithm

Q:- Given N islands and cost of construction of a bridge b/w multiple pair of islands. Find minimum cost of construction required such that it is possible to travel from one island to any other island via bridges.

If not possible, return -1.

Ex:- N = 7, E = 9

1 —3— 2
1 —5— 3
2 —1— 4
2 —5— 5
3 —3— 5
4 —2— 6
3 —8— 6
4 —5— 7
6 —3— 7

Obs 1:- All islands must be connected with each other.

⇓

Graph must be connected.

Obs. 2:- Min Cost ⇒ Min. no. of bridges to construct.

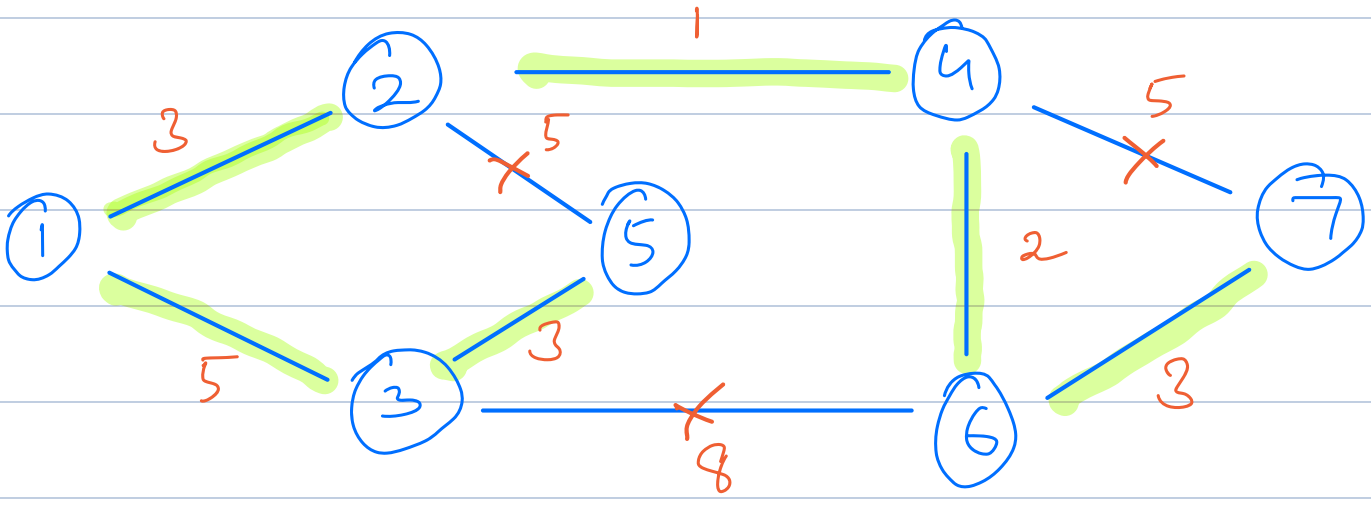
(N-1) edges.

Tree..

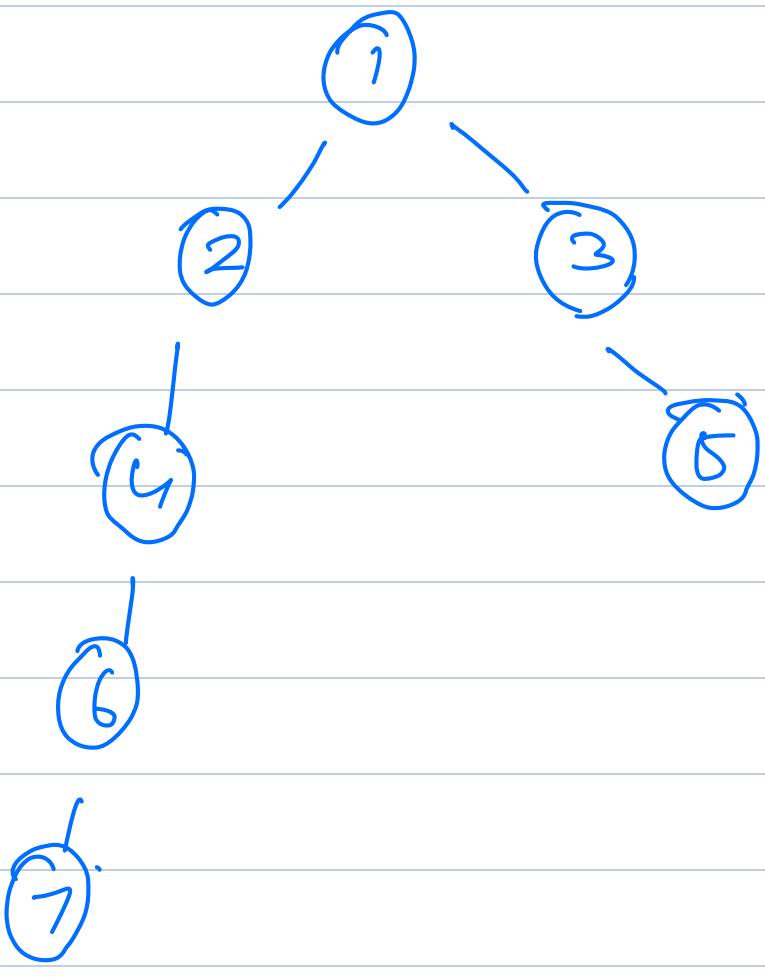
Σ Wt of selected edges is minimum

⇓

M.S.T.

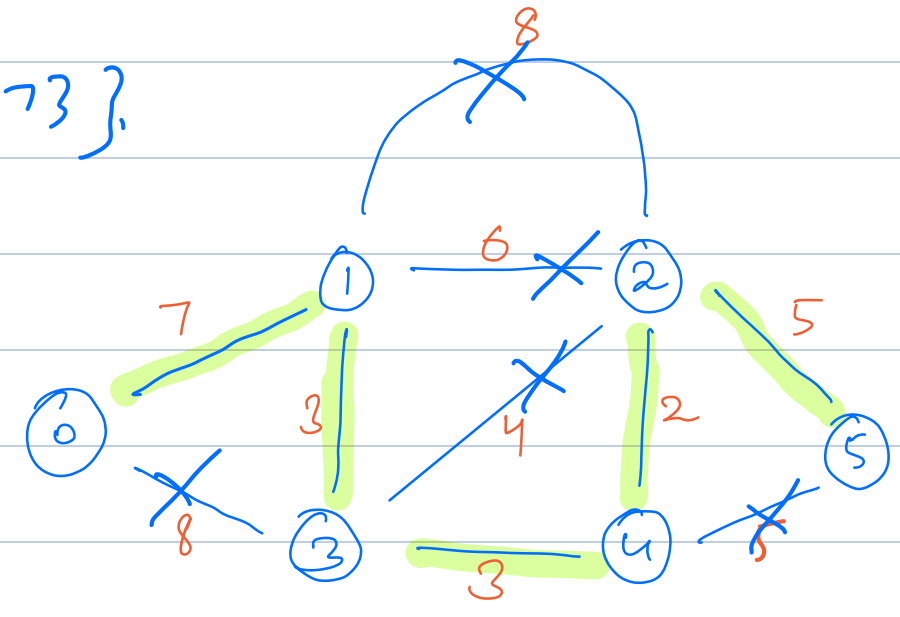


Ans = 17.



Kruskal's Algorithm

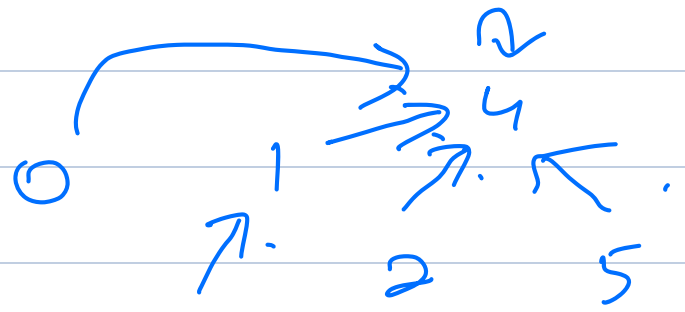
0	- { {1,7}, {3,8} }
1	- { {3,3}, {2,6}, {2,8}, {0,7} }
2	- { {1,8}, {1,6}, {3,4}, {4,2} }
3	- { {0,1}, {1,3}, {2,4}, {4,3} }
4	- { {2,2}, {3,3}, {5,5} }
5	- { {2,5}, {4,5} }

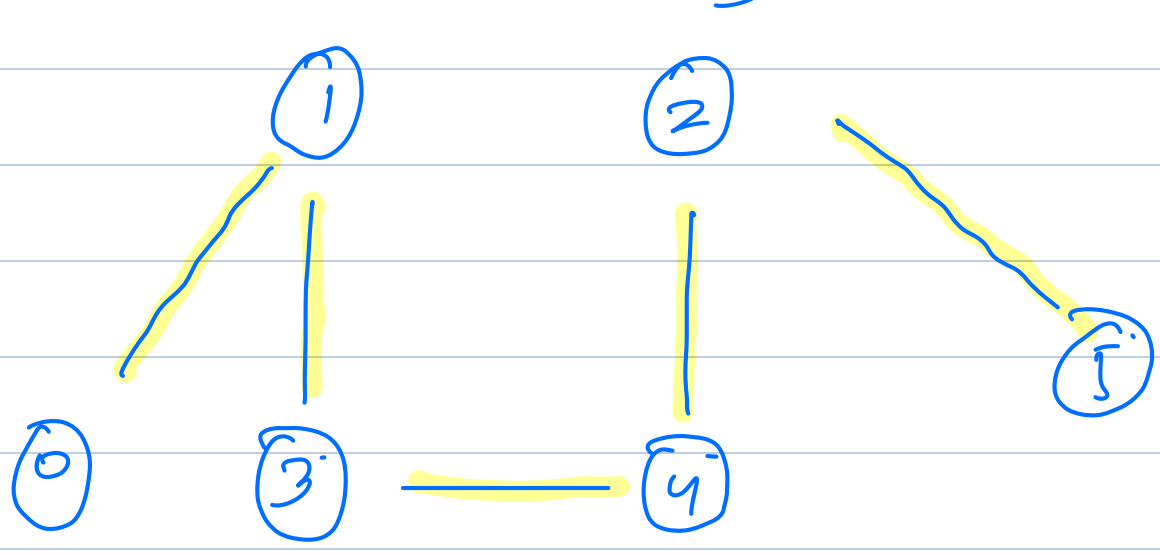


Ans = 20.

(1) Sort the edges of the graph in increasing order of wt.

2 — 2 — 4 ✓
1 — 3 — 3 ✓
3 — 3 — 4 ✓
2 — 4 — 3 ✗
2 — 5 — 5 ✓
4 — 5 — 5 ✗
1 — 6 — 2 ✗
0 — 7 — 1 ✓
0 — 8 — 3 ✗
1 — 8 — 2 ✗





Ans = 0 1 2 + 3
+ 3 + 5
+ 7.
return 20.

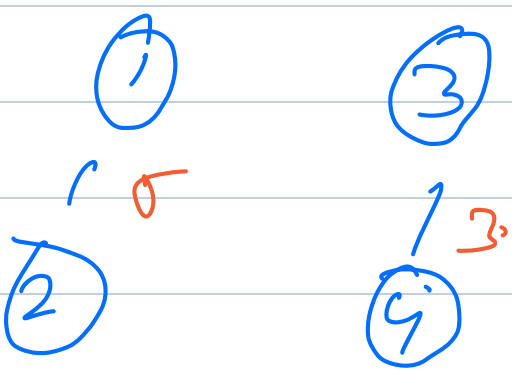
```
Edge {
  int u;
  int v;
  int wt;
}
```

List < Edge >

Step 2:-

```
for ( Edge e : arr ) {
    if (union(e.u, e.v) == true) {
        ans += e.wt;
    }
}
return ans;
```

** → Check for connected graph.

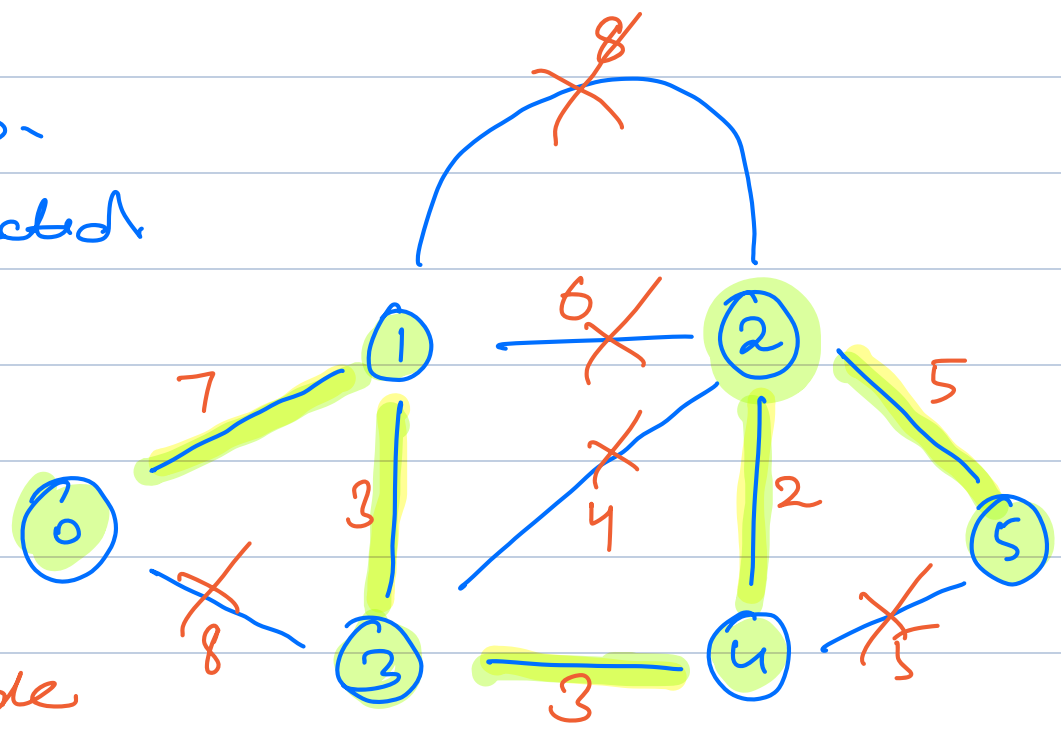


T.C. → O(E log E)

S.C. → O(E+V).

Prim's Algorithm

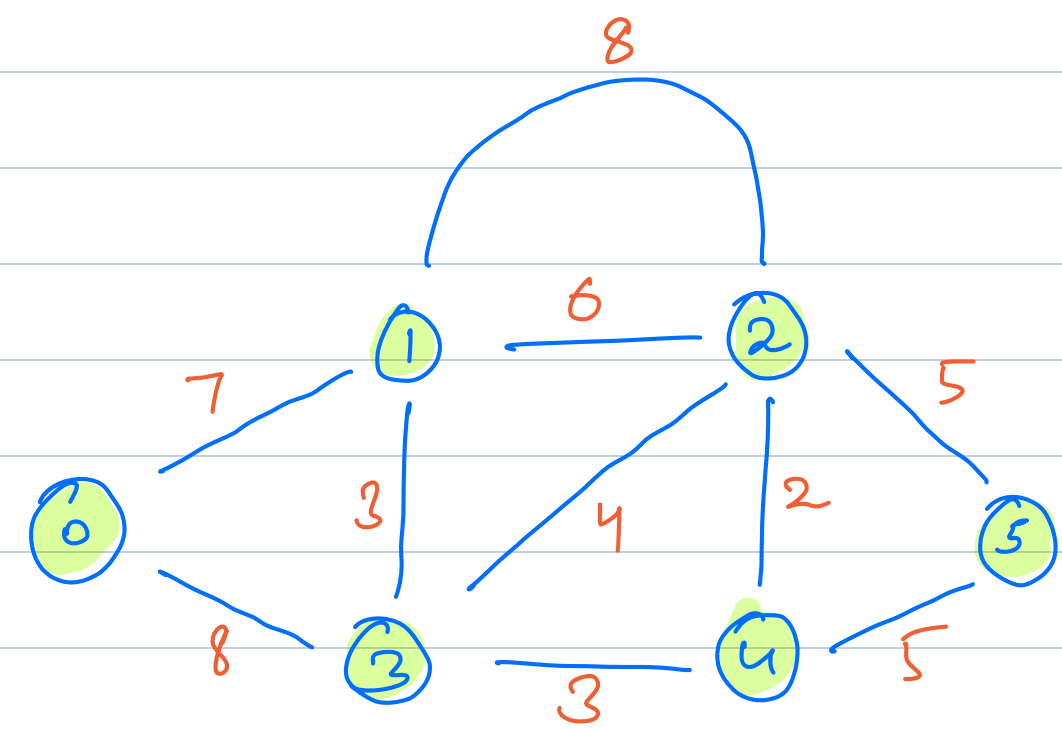
Yellow → In contest.
Green → Already selected.



→ Start with any node as root node
→ Keep on adding the other nodes with min. cost;

visited.

t	t	t	t	t	t
0	1	2	3	4	5



prority Q.

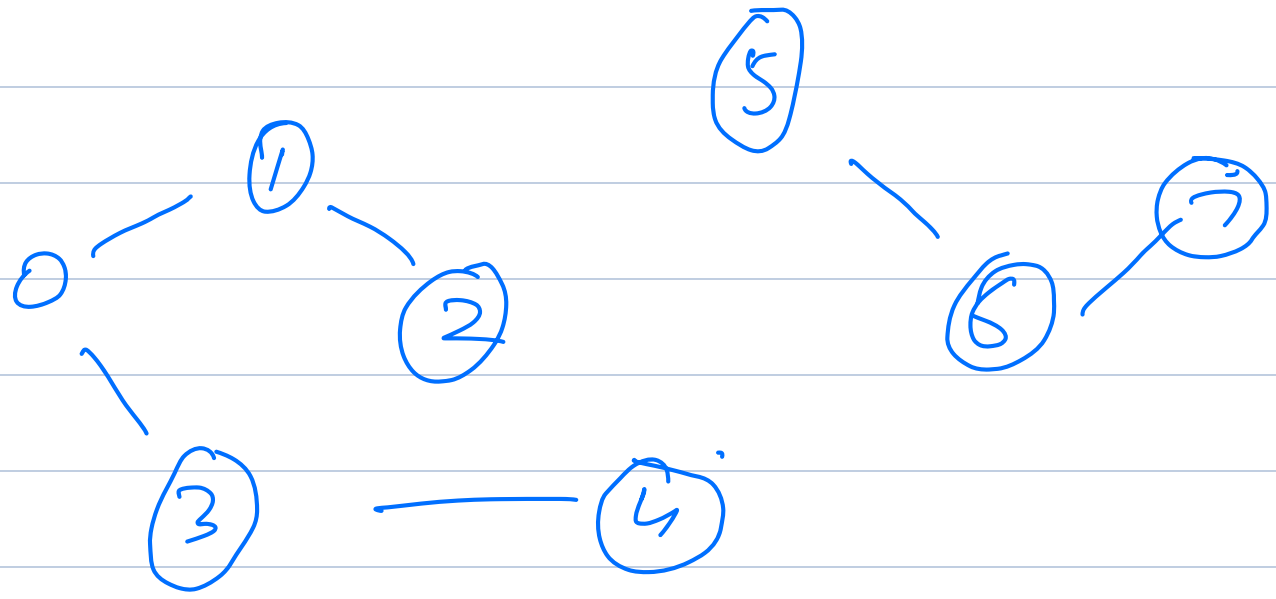
Edge {
int v;
int wt
3.

ans
0 + 7 + 3
+ 3 + 2.
+ 5

return 20;

```
# visited[N]; ∀i visited[i] = false;
minHeap<Edge> heap;

visited[0] = true;
for(Edge e : graph[0]){
    heap.insert(e);
}
int ans = 0;
while(heap.size() > 0){
    Edge re = heap.removeMin();
    if(visited[re.v] == true){
        continue;}
    else{
        visited[re.v] = true;
        ans += re.wt;
        for(Edge e : graph[re.v]){
            if(visited[e.v] == false){
            heap.insert(e);
        }
    }
}
return ans
```

Check for connectivity before moving part away.



T.C → O(E log E)

S.C → O(N+E).

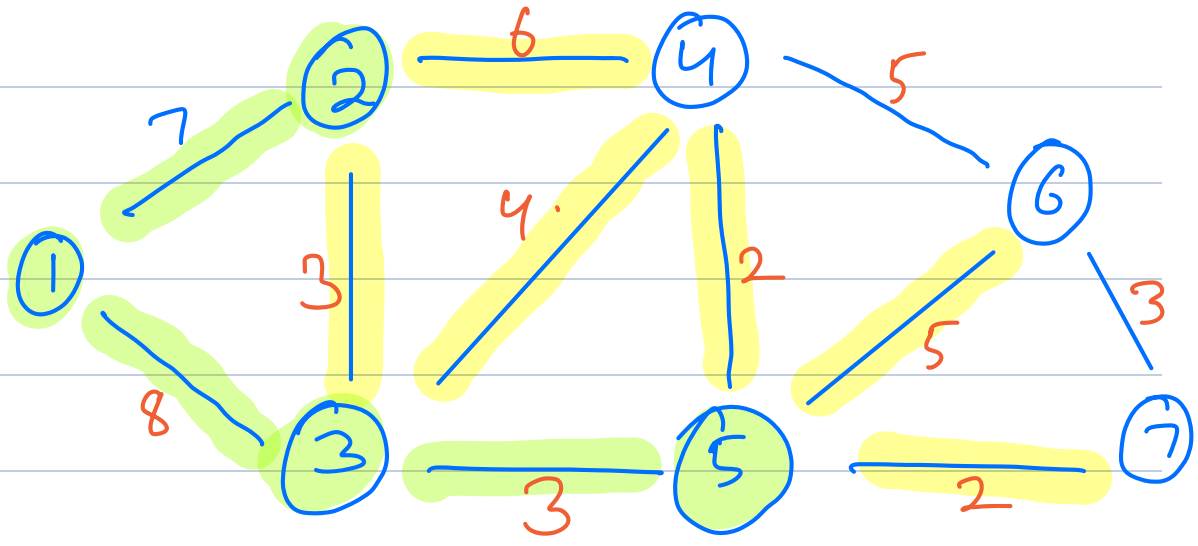
8:50

Dijkstra's Algorithm

(Single source shortest path)

(Edge wt ≥ 0)

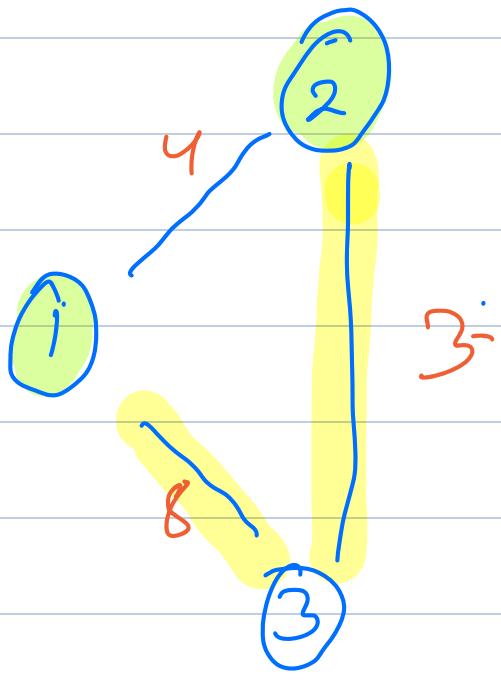
Q:- There are N cities in a country, you are living in city 1. Find min. distance to reach every other city from city-1.



Expected Output

dist:

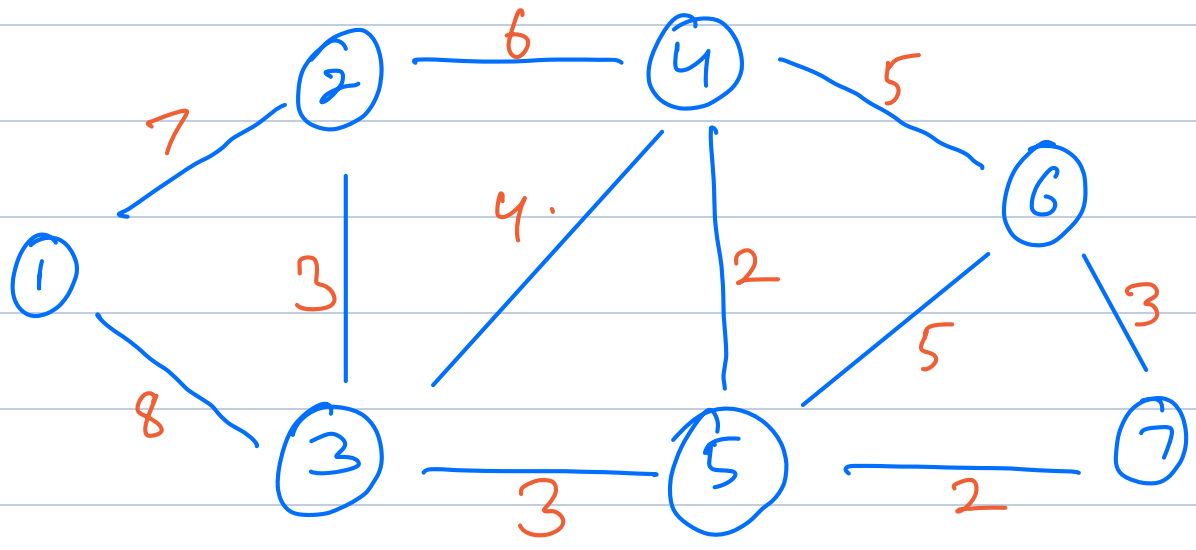
1	2	3	4	5	6	7
0	7	8	12	11	16	13



More connection

8, 4+3

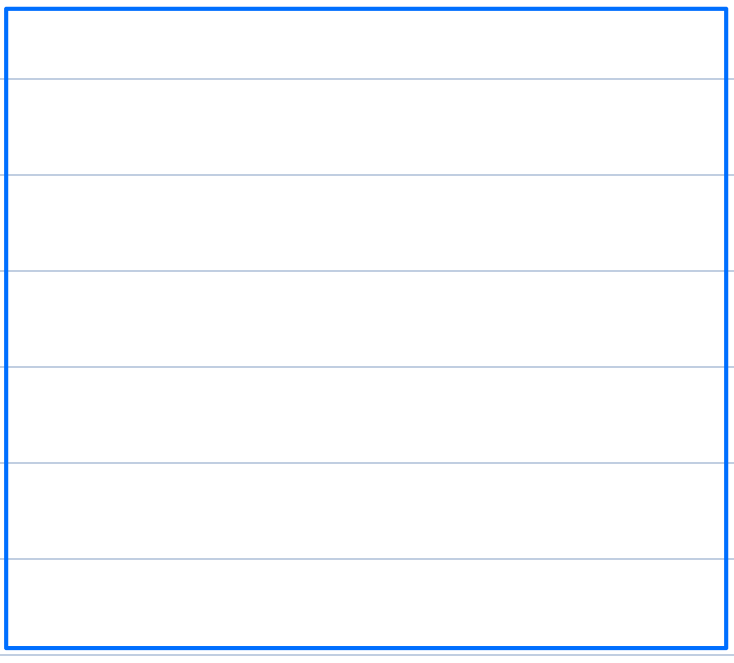
```
Pair {
    int v;
    int wsfi
};
```



dist: [∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞]

7 8 12 11 16 13

0 1 2 3 4 5 6 7



minHeap.

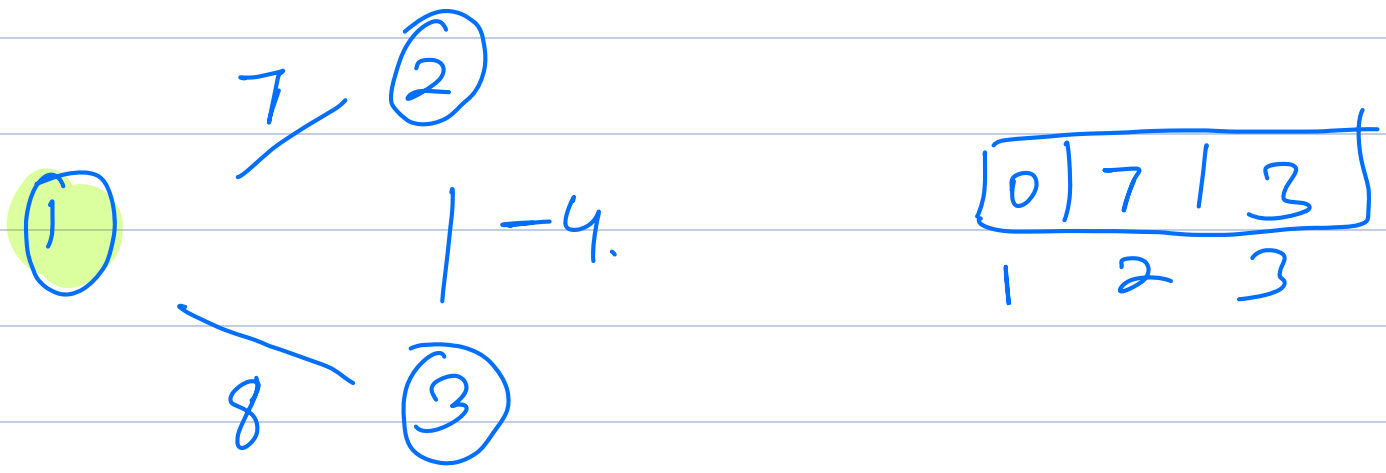
Code:

```
dist[N+1]; // dist[i] = INT_MAX;

minHeap<Pair> heap;

heap.insert(new Pair(src, 0));

while (heap.size() != 0) {
    Pair rp = heap.removeMin();
    if (dist[rp.v] != INT_MAX)
        { continue; }
    else {
        dist[rp.v] = rp.wsf;
        for (nt nbr: graph[rp.v]) {
            if (dist[nbr] == INT_MAX){
                heap.insert(new Pair
                (nbr, rp.wsf
                + wt. of curr
                edge));
    }
}
}
return dist;
```

T.C → O(ElogE).

S.C. → O(E).

Can it work for -ve edge weight ?



Djikstra fe not gong to give the correct answer.