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.

Eg 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

Observation

→ Graph must be connected.

→ Min cost ⇒ min no. of bridges/edges

(N-1) edges.

⇓

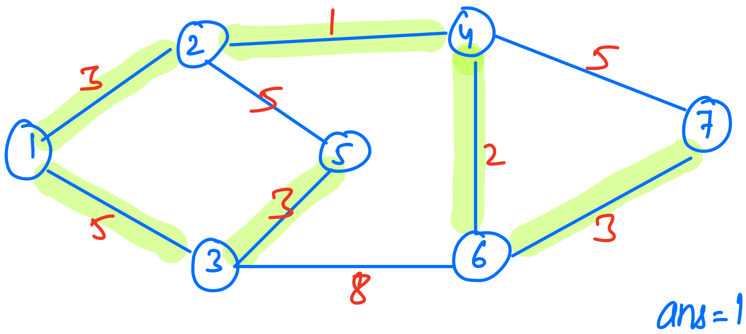
Tree

Σ wt. of select edges is minimum

⇓

Minimum Spanning Tree

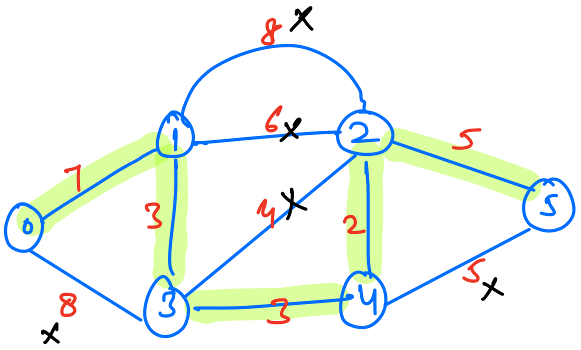
(M.S.T)



ans=17

Kruskal's Algorithm

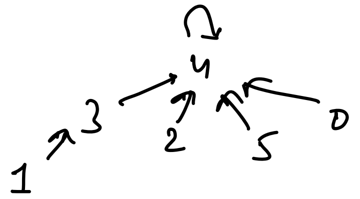
- select the edge with the minimum cost, if it is not forming a cycle, till the graph is completely connected.

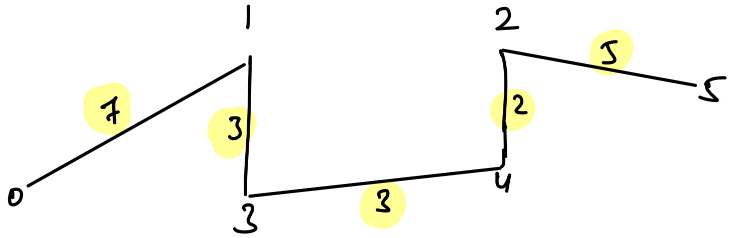


ans → 20

→ Sort the edges on the basis of their ed. wt.

✓	2	2	4
✓	1	2	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 ← 20

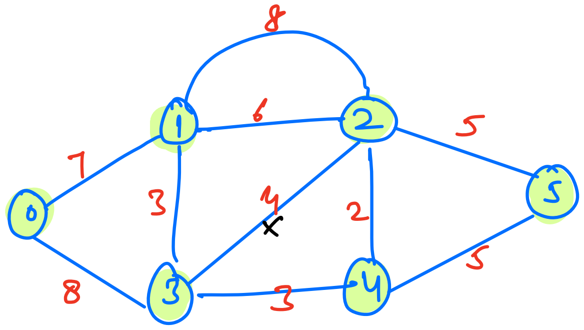
```
→   for( edge e : arr ){
        if(union(e.u, e.v) == true){
            ans += e.wt;
        }
    }
    return ans;
```

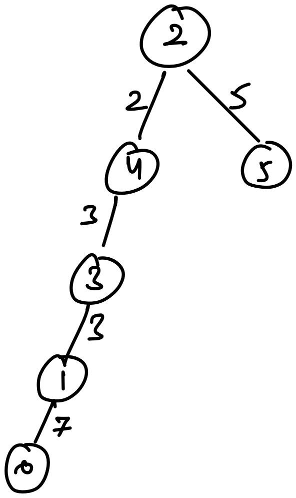
```
Edge {
    int u;
    int v;
    int wt;
}
```

T.C → O(E log E)
S.C → O(N + E)

Prim's Algorithm

→ start with any node as root node of M.S.T and keep on adding the other nodes with minimum weight.





ans = 20

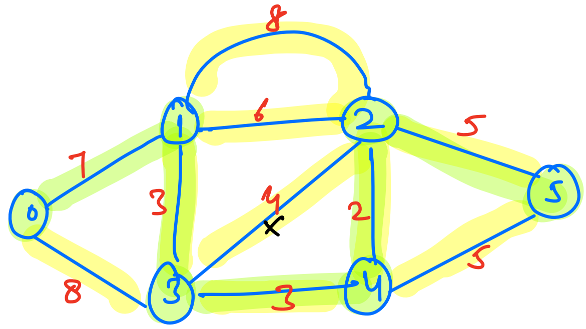
① Select any node as the root node of M.S.T and mark it visited.

② insert all the edges connected to this root node in the min Heap.

visited →

0	1	2	3	4	5
t	t	t	t	t	t
~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~

(1,7) (3,8)



Edge {
    int v;
    int wt;
}

ans = 0+2+3+3+5+7
= 20

Graph

0 → {(1,7), (3,8)}
1 → {(0,7), (3,3), (2,6), (2,8)}
2 → {(1,6), (1,8), (3,4), (4,2), (5,5)}
3 →
4 →
5 →

code →

```
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 = q.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;
```

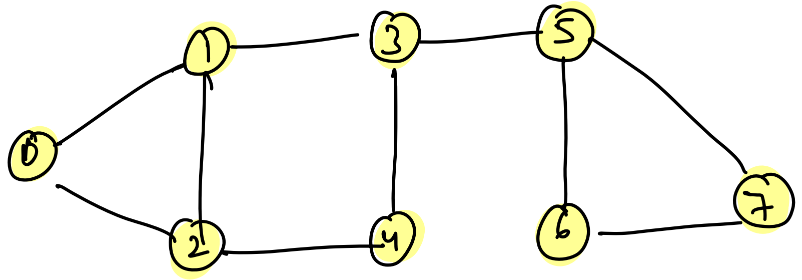
T.C → O(E log E)
S.C → O(N + E)

** :- Edges must be compared on the basis of ed. wt.

↓

vtx
ed wt

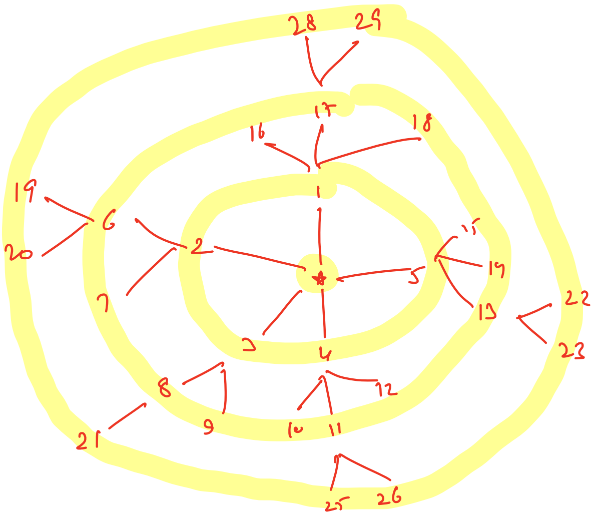
Q. Find the min no. of edges to reach v starting from u in undirected simple graph.



u = 0
v = 7

ans = 4

idea → Apply BFS



~~0~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	7
0	1		2		3	4	

ans = 4

code → todo

T.C → O(N + E)
S.C → O(N)

Dijksta's Algorithm [Single sourced shortest path]

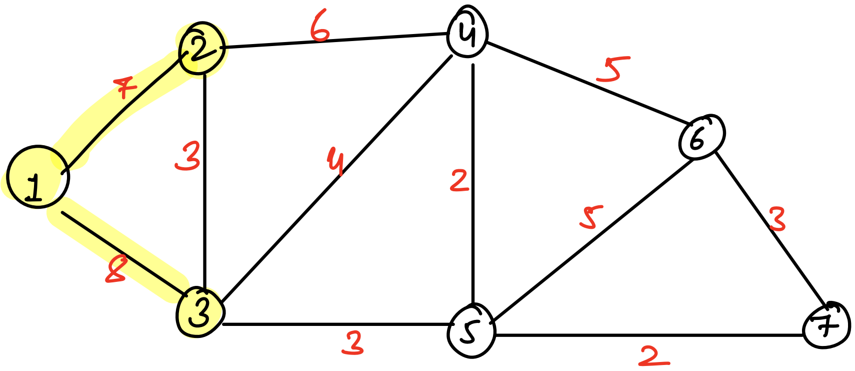
Q) There are N cities in a country, you are living in city-1. Find minimum distance to reach every other city from city-1.

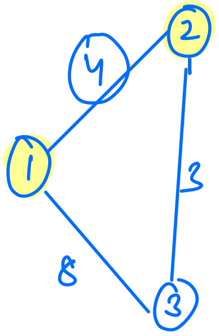
[edge wt. > 0]

Expected output

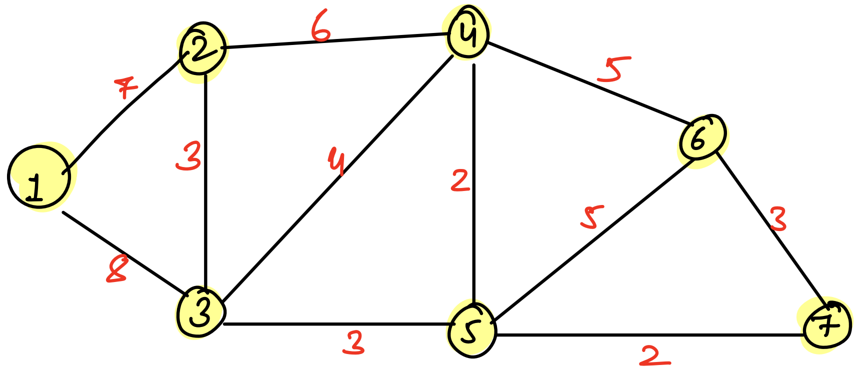
↓

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





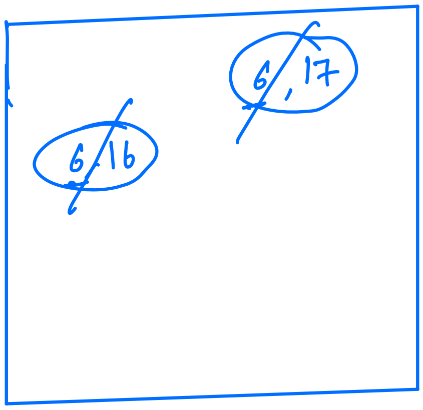
→ minimum wt. so far starting from the source.



dist →

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

∞ → not visited.



```
Pair {
      int v;   (vertex)
      int wsf; (wt. so far)
}
```

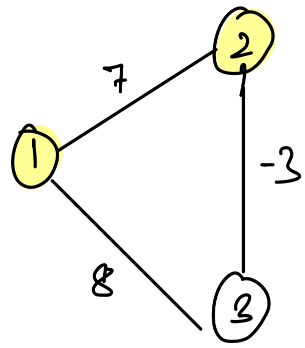
#code:→

```
dist [N+1] ,  // ∀i, 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( int nbr : graph [rp.v]){

                        if( dist [nbr] == INT-MAX) {

                                heap.insert(new Pair( nbr, rp.wsf + wt of));
                                                                 current
                                                                  edge
                        }
                }
        }
}
return dist [] ;
```

T.C → O(E log E)

S.C → O(E)

Can it work if any wt < 0 ?



0	5	4
1	2	3

	0	7	4
dist	~~∞~~	~~∞~~	~~∞~~
	1	2	3

~~3,4~~

3,8

To be continued!!

- → Hashing
- → Stack n Queue
- → Linked list
- → Trees
- → Sorting + Searching
- → Recursion

- → Arrays
- → Strings
- → D.P
- → Graph
- → Backtracking & Greedy

- → B.M
- → Trie
- → Maths

Arcesium