Lecture :- Graph 3

Agenda

- Applications of DSU
- Prims Algorithm
- Krushkal Algorithm
- Dijakstra's Algorithm.

Applications of DSU

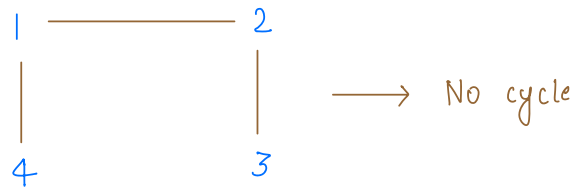
* Detect cycle in an undirected / directed graph.

Example: n = 4 (undirected graph)

edges : (1, 2)

(2, 3)

(1, 4)



---

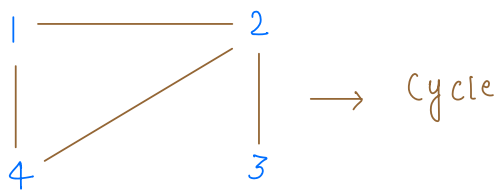
n = 4

edges : (1, 2)

(2, 3)

(1, 4)

(2, 4)

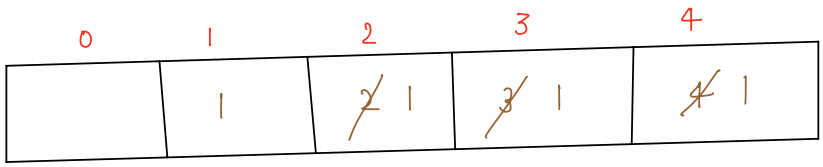


Approach

n = 4

edges: (1, 2) (1, 4) (2, 3) (2, 4)

parent[] =



Edges	① ② ③ ④	
(1, 2)	p[1] = 1 p[2] = 2	(1 2) (3) (4) — No cycle
(2, 3)	p[2] = 1, p[1] = 1 p[3] = 3	(1 2 3) (4) — No cycle
(1, 4)	p[1] = 1 p[4] = 4 p[4] = 1	(1 2 3 4) — No cycle
(2, 4)	p[2] = 1, p[1] = 1 p[4] = 1, p[1] = 1	cycle

Code:

```
boolean isCycle(n, edges[][]) {                    ☆☆☆
    parent[n+1];
O(n) —— for(i=1; i<=n; i++) {
        parent[i] = i;
    }
O(e) — for(i=0; i< edges.length; i++) {
        u = edges[i][0];
        v = edges[i][1];
O(1) —— rootU = getRoot(u);
O(1) —— rootV = getRoot(v);
        if(rootU == rootV) {
            return true;
        }
O(1) —— union(u,v);
    }
    return false;
}
```

TC: $O(n+e)$

SC: $O(n) + O(n)$

↑ parent array ↑ stack space

* Minimum spanning tree [App. of DSU]

Qu Given n islands and cost of construction of bridge b/w multiple pair of islands. find min cost of construction required such that it is possible to travel from one island to another island via bridges. If not possible, return -1.

ip n = 5

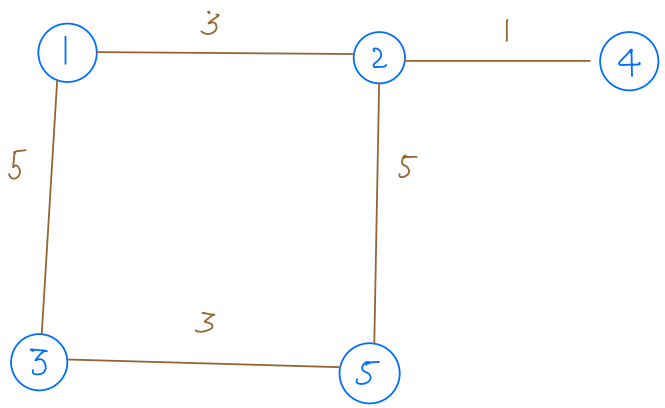
1 —3— 2

1 —5— 3

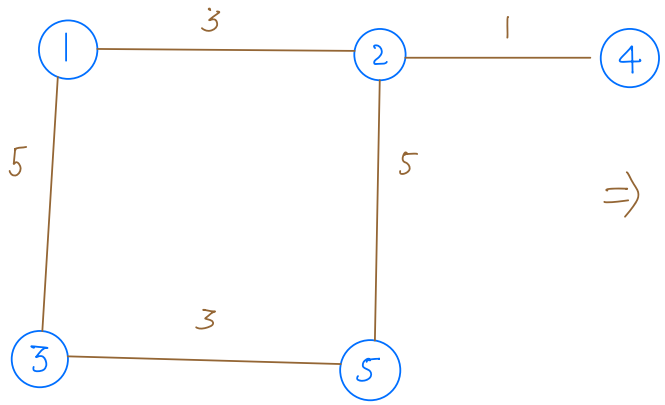
2 —1— 4

2 —5— 5

3 —3— 5

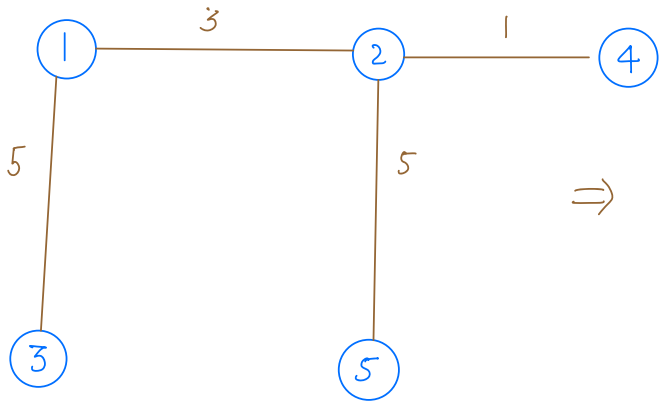


Min edges to connect n nodes = n-1 [Tree]



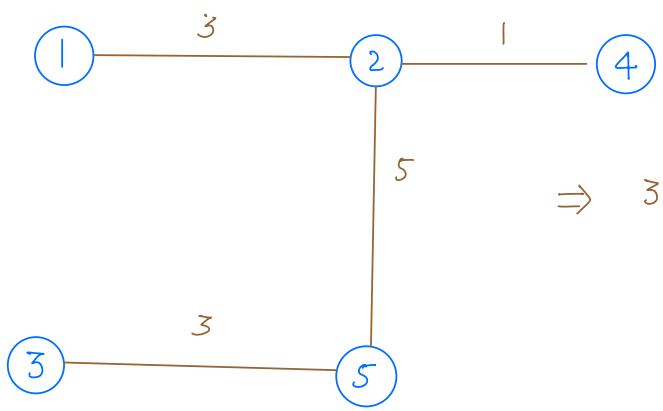
$\Rightarrow$ 3 + 5 + 3 + 5 + 1 = 17 Ans

Case1:



$\Rightarrow$ 3 + 5 + 5 + 1 = 14

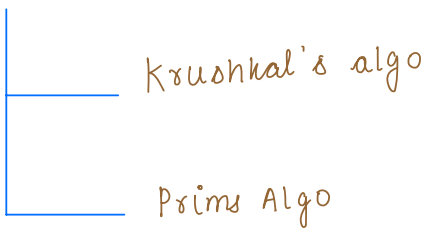
Case2:



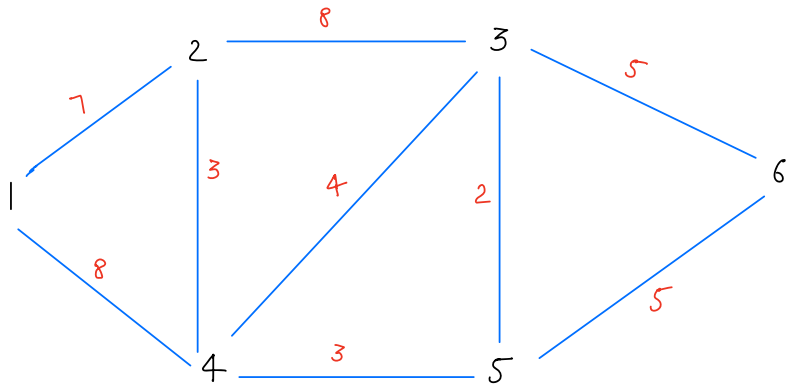
$\Rightarrow$ 3 + 3 + 5 + 1 = 12 Ans

Minimum spanning tree

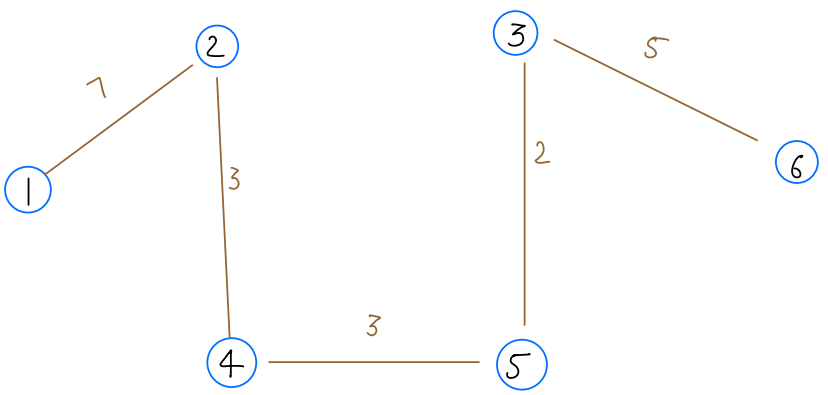
Tree generated from a connected graph such that all nodes are connected and sum of weights of all selected edges weight is minimum.



Kruskal's Algorithm



Approach:



$\Rightarrow 7 + 3 + 3 + 2 + 5 =$ 20 Ans

parent[] =

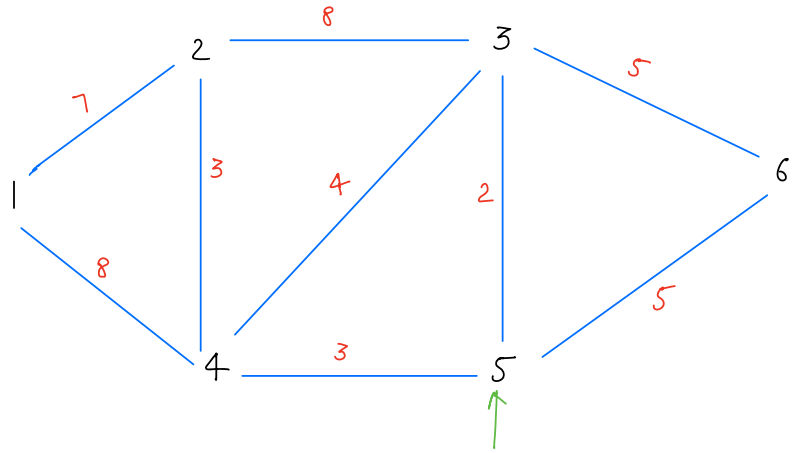
0	1	2	3	4	5	6
	1	~~2~~ 1	~~3~~ 2	~~4~~ 2	~~5~~ 3	~~6~~ 2

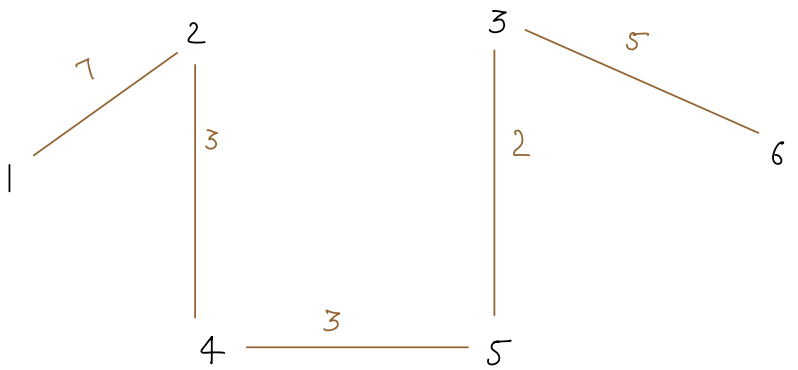
Code:

```
int Kruskal(n, edges[][]) {
    // Graph
    1. Sort edges wrt weight.
    int ans = 0;
    for(i=0; i<edges.length; i++) {
        u = edges[i][0];
        v = edges[i][1];
        wt = edges[i][2];
        rootU = getRoot(u);
        rootV = getRoot(v);
        if(rootU == rootV) {
            continue;
        }
        union(u, v);
        MST ---- { graph[u].add(new Pair(v, w));
                   graph[v].add(new Pair(u, w)); }
        ans += wt;
    }
    return ans;
}
```

ans → min cost

Prims Algorithm





$\Rightarrow 7+3+2+3+5 = 20$ Ans

Minheap: (v, wt)

~~(3,2)~~ ~~(6,5)~~ ~~(4,3)~~ (2,8) ~~(4,4)~~ ~~(6,5)~~ (1,8)

~~(2,3)~~ ~~(1,7)~~

visited:

0	1	2	3	4	5	6
	~~f~~ t	~~f~~ t	~~f~~ t	~~f~~ t	~~f~~ t	~~f~~ t

code:

```
int prims(n, edges[][]) {

    PriorityQueue<Pair> pq;
    vst[n+1];
    // Insert any random node along with
       their edges. and wt.

    while( !pq.isEmpty()) {
        Pair p = pq.poll();
        int u = p.v;
        int wt = p.wt;
        if ( !vst[u]) {
            ans += wt;
            vst[u] = t;
            for( (v, w) : graph[u]) {
                if ( !vst[v]) {
                    pq.add( new Pair (v, w));
                }
            }
        }
    }
```

```
class Pair {
    int v;
    int wt;
}
```

TC: $O(E \log E)$

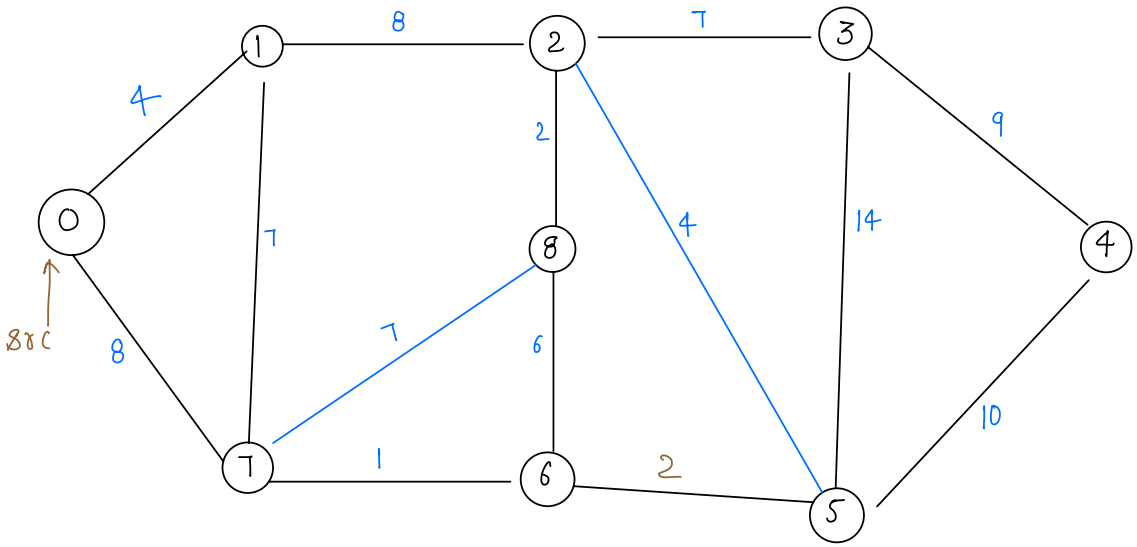
SC: $O(e + n)$

e ↑ heap, n ↑ visited

Break :- 8:23 - 8:33

Dijakstra's Algorithm

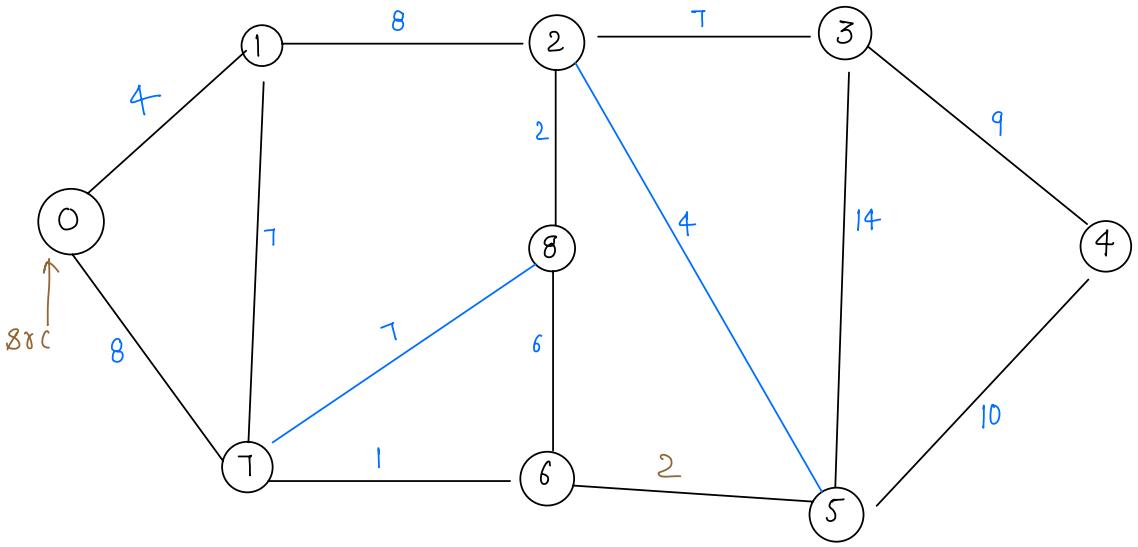
[Min distance from src to all other vertices]

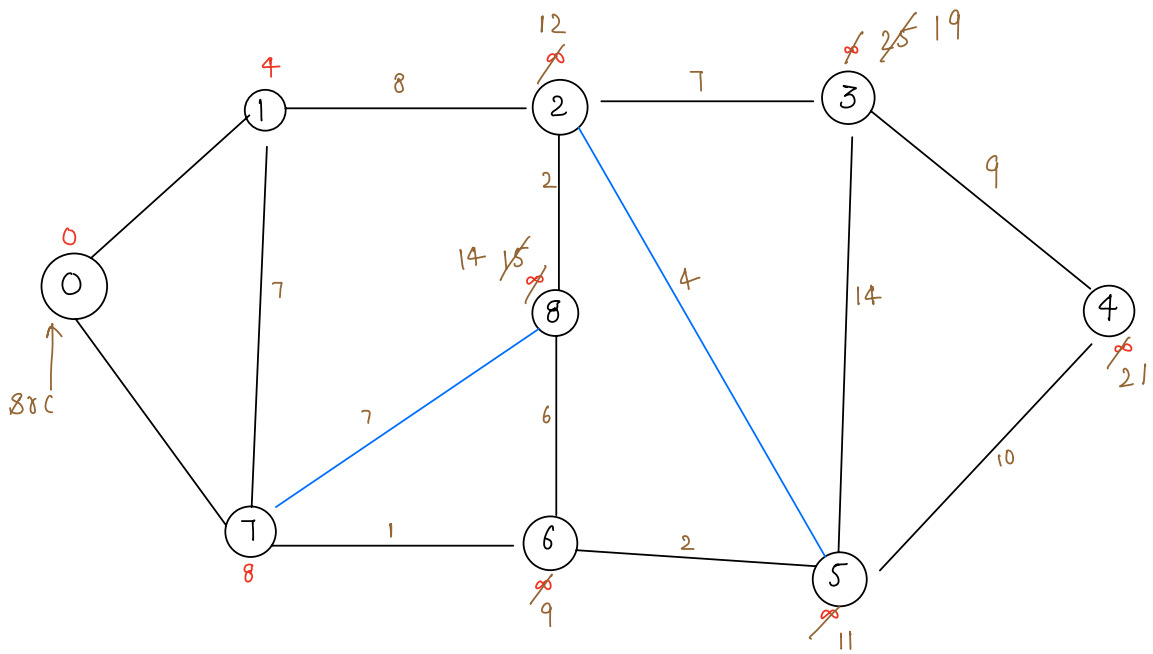


output:

	Distance
0 to 1	4
0 to 2	12
0 to 3	19
0 to 4	21
0 to 5	11
0 to 6	9
0 to 7	8
0 to 8	14

Example:





dist[] =

0	1	2	3	4	5	6	7	8
0	4	∞ 12	∞ 25 19	∞ 21	∞ 11	∞ 9	8	∞ 15 14

visited:

0	1	2	3	4	5	6	7	8
t	f t	f t	f t	f t	f t	f t	f t	f t

minheap: | ~~(1,4)~~ (7,8) ~~(2,12)~~ (6,9) ~~(8,15)~~ (5,11) ~~(3,25)~~ ~~(4,21)~~ ~~(8,19)~~ ~~(3,19)~~ |

```
int[] dijkastra( List<Pair> graph[], int src, n) {
    int dis[n+1]; — O(n)
    for(i=0; i<=n; i++) {
        if( i != src) {
            dis[i] = ∞;
        }
    }
    vis[n+1]; vis[src]=true; — O(n)
    PriorityQueue<Pair> pq; — O(n)
    for(Pair p: graph[src]) {
        pq.add( new Pair( p.v, p.wt));
    }
    while( !pq.isEmpty()) {
        Pair p = pq.poll();
        v = p.v;
        wt = p.wt;
        if( vis[v] == true) {
            continue;
        }
```

```
            vis[v] = true;
            for( Pair p : graph[v]) {
                int currDis = dis[v] + p.wt;
                if( currDis < dis[p.v] ) {
                    dis[p.v] = currDis;
                    pq.add( new Pair(p.v, p.wt));
                }
            }
        }
        return dis;
    }
```

Relaxation ←

TC: $O(V^2)$

SC: $O(V+E)$