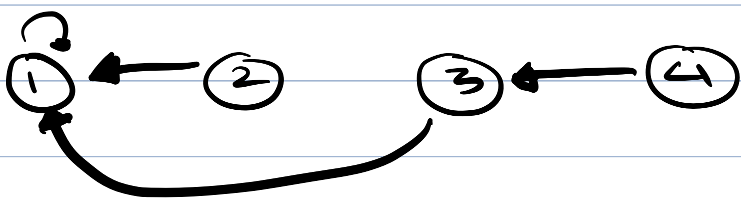
Agenda :

- Applications of DSU
- Minimum Spanning Tree → Prim's Algorithm
- BFS
- Dijkstra Algo

Applications of DSU

1. Checking if an undirected graph is connected

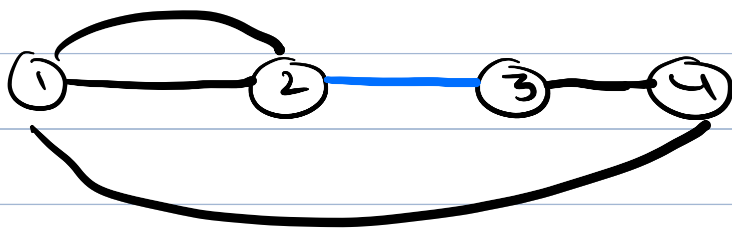
N = 4

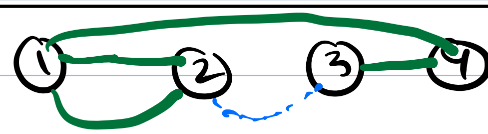


Queries

(1,2)	T
(3,4)	T
(1,2)	F
(1,4)	T
(2,3)	F

	1	2	3	4
par →	1	~~2~~ 1	~~3~~ 1	~~4~~ 3

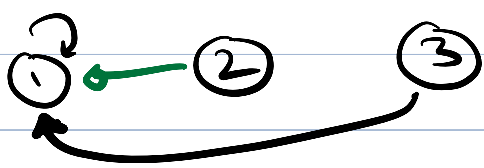




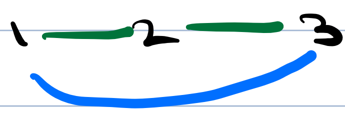
No. of sets → For every node, count unique roots

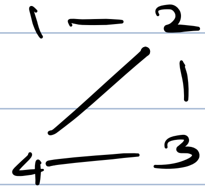
2. Cycle in an undirected graph

N = 3



(1, 2)	T
(2, 3)	T
$\boxed{(1, 3) \quad F}$	

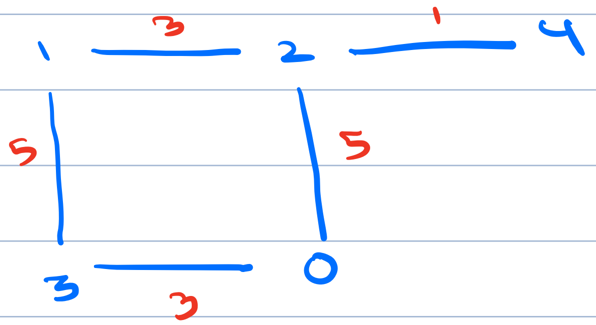




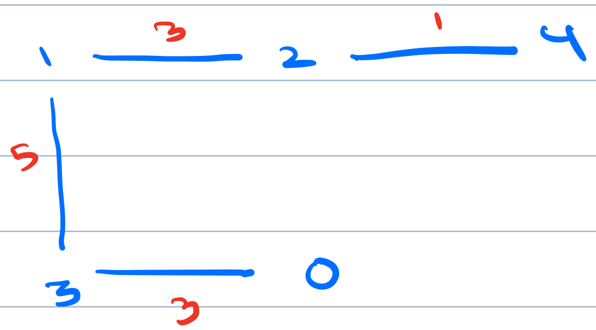
(2,6)

Q. Given N islands and cost of construction of a bridge b/w multible pair of islands. Find min. cost of construction s.t it is possible to travel from any island to any other island. If not possible, return -1.

N = 5 (4 bridges)

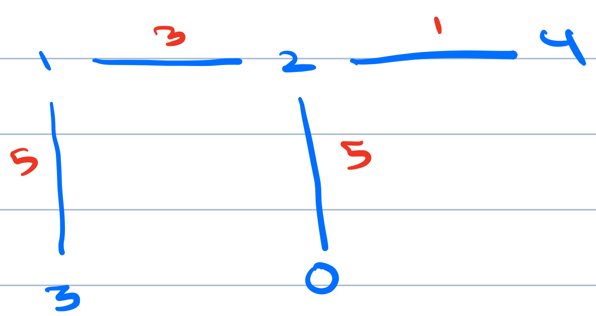


Minimum Spanning Tree



ans

Cost = 12



Cost = 14

Minimum Spanning Tree - Tree like structure generated from a connected graph s.t all nodes are connected and sum of weights of all selected edges is minimum.

2 Algorithms : 1 Prim's   2 Kruskal's

Spanning Tree : A spanning tree is a subgraph of a graph that includes all the nodes, maintains connectivity and has no cycle.

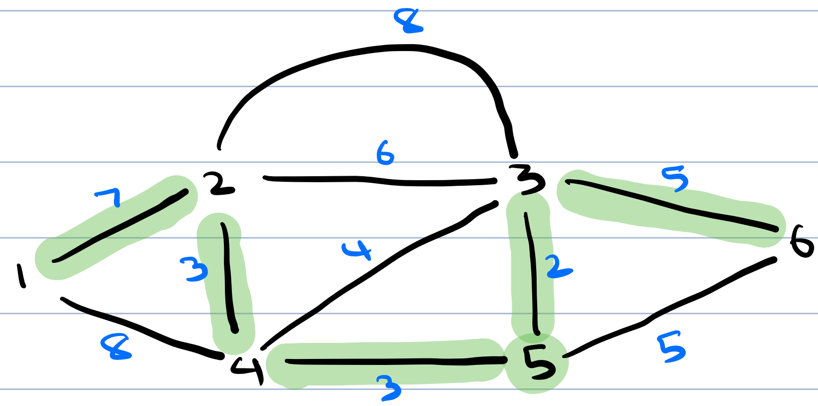
↳ It doesn't give shortest distance from one node to another. (2 - 0)

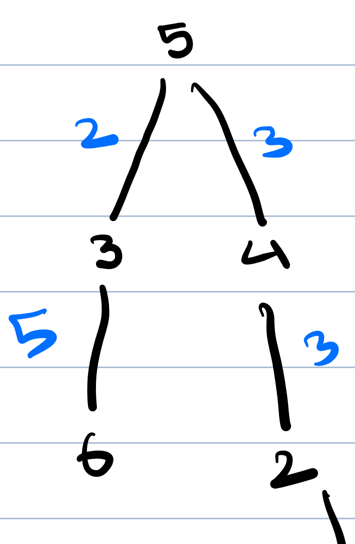
Prim's Algo

MST

$N = 6$

$E = 10$





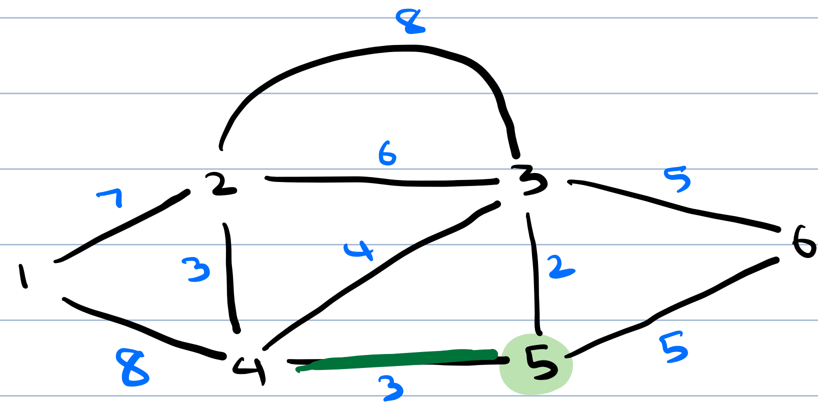
cost = ~~0~~ ~~2~~ ~~5~~ ~~8~~ ~~13~~ 20

wt, Node

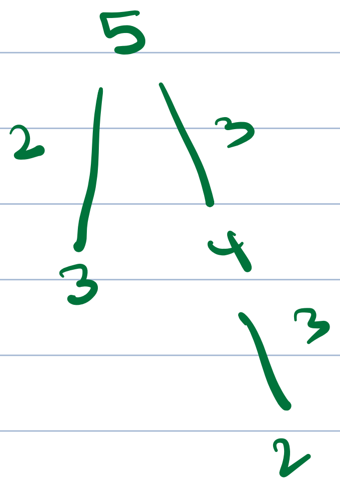
7,1	2, 3	6, 2
8, 1	3, 4	8, 2
	5, 6	4, 4
	3, 2	5, 6

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

7



cost = ~~2~~ 5

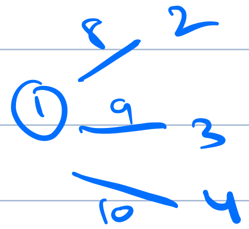


visited	1	2	3	4	5	6
	F	F	~~F~~ T	~~F~~ T	~~F~~ T	F

2, 3	5, 6
3, 4	6, 2
5, 6	8, 2
3, 2 8, 1	4, 4

w, Node

```
Minheap mh (of pairs <w, v>)
bool visited [N] -> <F>
visited [0] = true

// all options of 0 to mh
int cost = 0
while (mh.size() > 0) {

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

        cost += wt
        visited [Node] = true

        (for (v, w) in adj [Node]) {
            if (! visited [v]) {
                mh.insert (<w, v>)
            }
        }
}

return cost
```



nbr, wt
1 -> (2, 8),
(3, 9),
(4, 10)

TC : O(E log E)

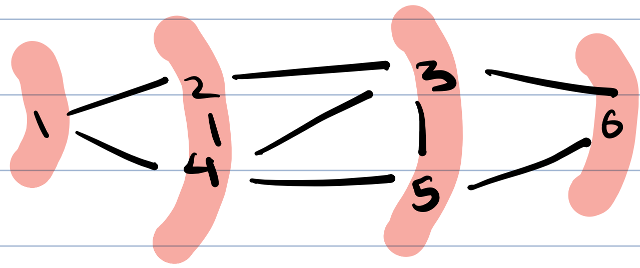
SC : O(N + E)

visited ↓ (N), Heap ↓ (E)

N/V -> nodes, E -> Edges

10:20

Find min distance to travel from u → v.



src	dst
1	6

ans → 3

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

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

src,0

~~x~~,d	nbr1, d+1	nbr 2, d+1

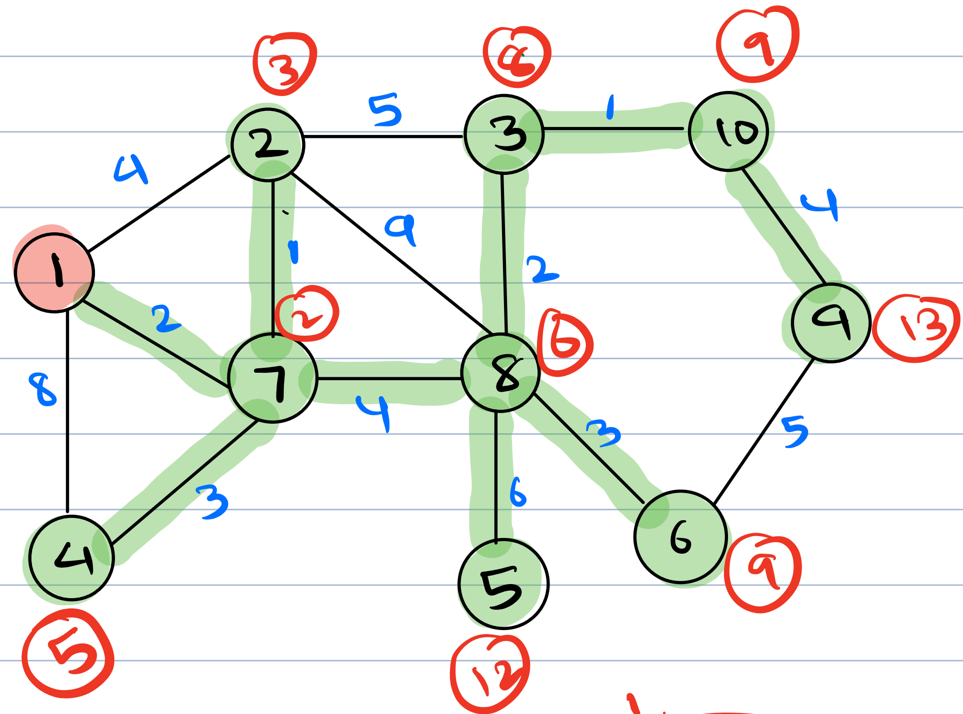
BFS gives you shortest path in an unweighted graph (undirected and directed)

↓

dist = no. of edges

$\boxed{\text{contest} \to 9 \text{ Feb} \to \text{Friday} \quad 2.5 \text{ hr} \quad \geq 60\%}$

Find shortest path from a source to all other node. (+ve edge weights)



Adj List <N,W>

1	~~2,4~~	~~4,8~~	~~7,2~~		
2	~~1,4~~	~~3,5~~	~~7,1~~	~~8,9~~	
3	~~2,5~~	~~8,2~~	~~10,1~~		
4	~~1,8~~	[redacted]	~~7,3~~		
5	~~8,6~~				
6	~~8,3~~	~~9,5~~			
7	~~1,2~~	~~2,1~~	~~4,3~~	~~8,4~~	
8	~~2,9~~	~~3,2~~	~~5,6~~	~~6,3~~	~~7,4~~
9	~~6,5~~	~~10,4~~			
10	~~3,1~~	~~9,4~~			

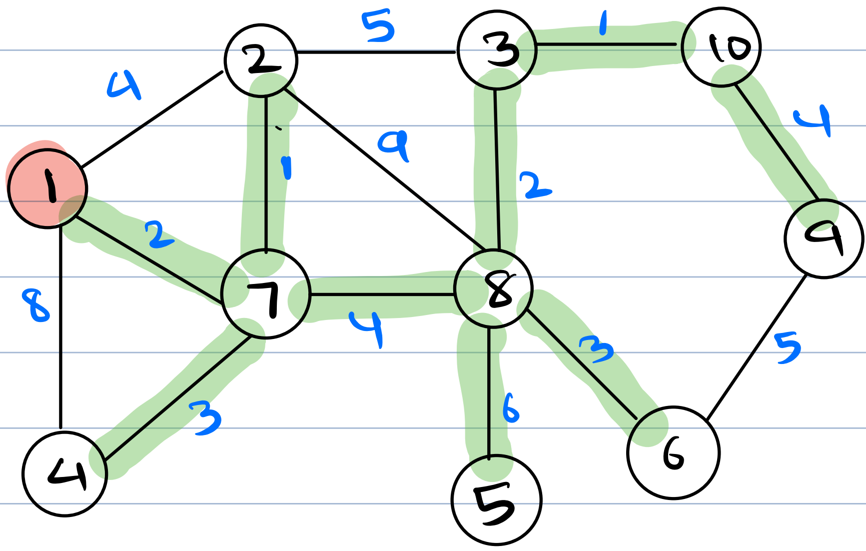
src = 1

dist

1	2	3	4	5	6	7	8	9	10
0	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	~~4~~ 3	8	~~8~~ 5	12	9	2	6	~~14~~ 13	9

wt, Node

4, 2	3,2	12, 5
2, 7	5, 4	9, 6
6, 4	6, 8	9, 10
	8, 3	14, 9
		13, 9



1	2	3	4	5	6	7	8	9	10
0	3	8	5	12	9	2	6	13	9

source → insert all options w, n → pick out best → explore further options

```
min_heap (mh)      -> <w, v>
dist [N]  =  < INT_MAX >
dist [src] = 0
mh. insert (<0, src>)

while (! mh.empty()) {
    d , Node = mh. extractMin()
    if (d == dist [Node]) {
        for( v, w   in   adj [Node] )
            // check if we can reach
                v via Node
            if (dist [v] > d + w) {
                dist [v] = d + w
                mh.insert (<dist[v], v>)
            }
    }
}
```

$src \xrightarrow{d} Node \xrightarrow{w} v$

$$TC : O(V + \underset{\substack{\uparrow \\ N}}{E} \log E)$$

$$\Big\downarrow E \simeq V^2$$

$$TC : O(E \log V)$$

	min	max
Edges =	$N-1$	$\frac{N(N-1)}{2}$
$E \simeq$	N	N^2

$$E \log E = E \log V^2 \rightarrow 2E \log V \rightarrow E \log V$$