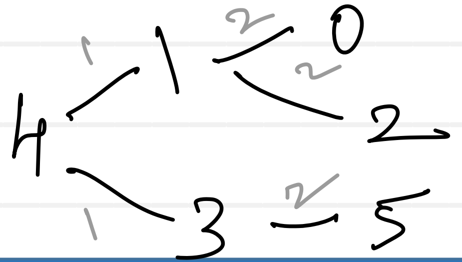
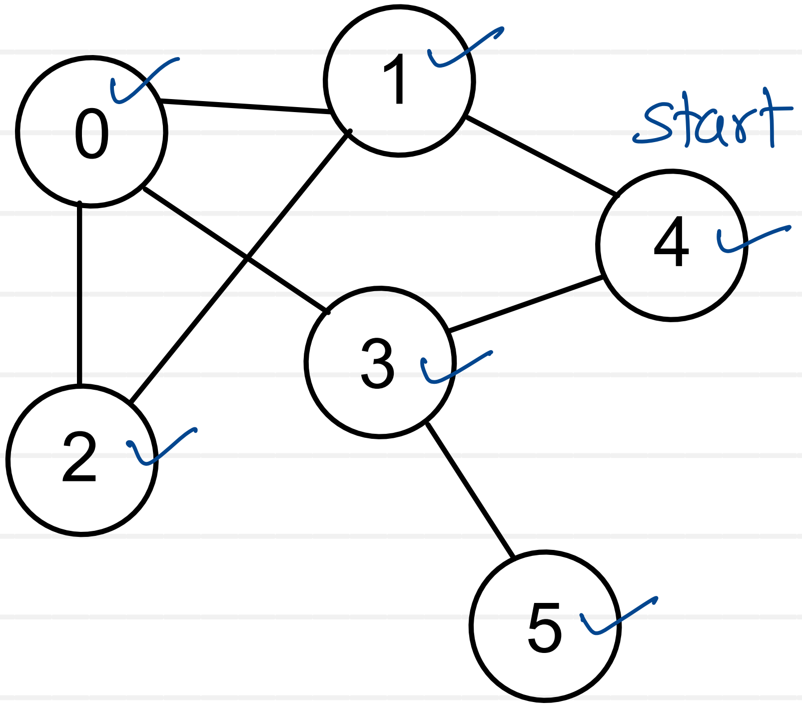
Single Source Path Length

1. Create path length array to keep length of vertex from start vertex.
2. push start on queue & mark it. length[start] = 0
3. pop the vertex.
4. push all its non-marked neighbors on the queue, mark them.
5. For each such vertex calculate length as length[neighbor] = length[current] + 1
6. print current vertex to that neighbor vertex edge.
7. repeat steps 3-6 until queue is empty.
8. Print path length array.

$\boxed{T(n) = O(V^2)}$





Queue

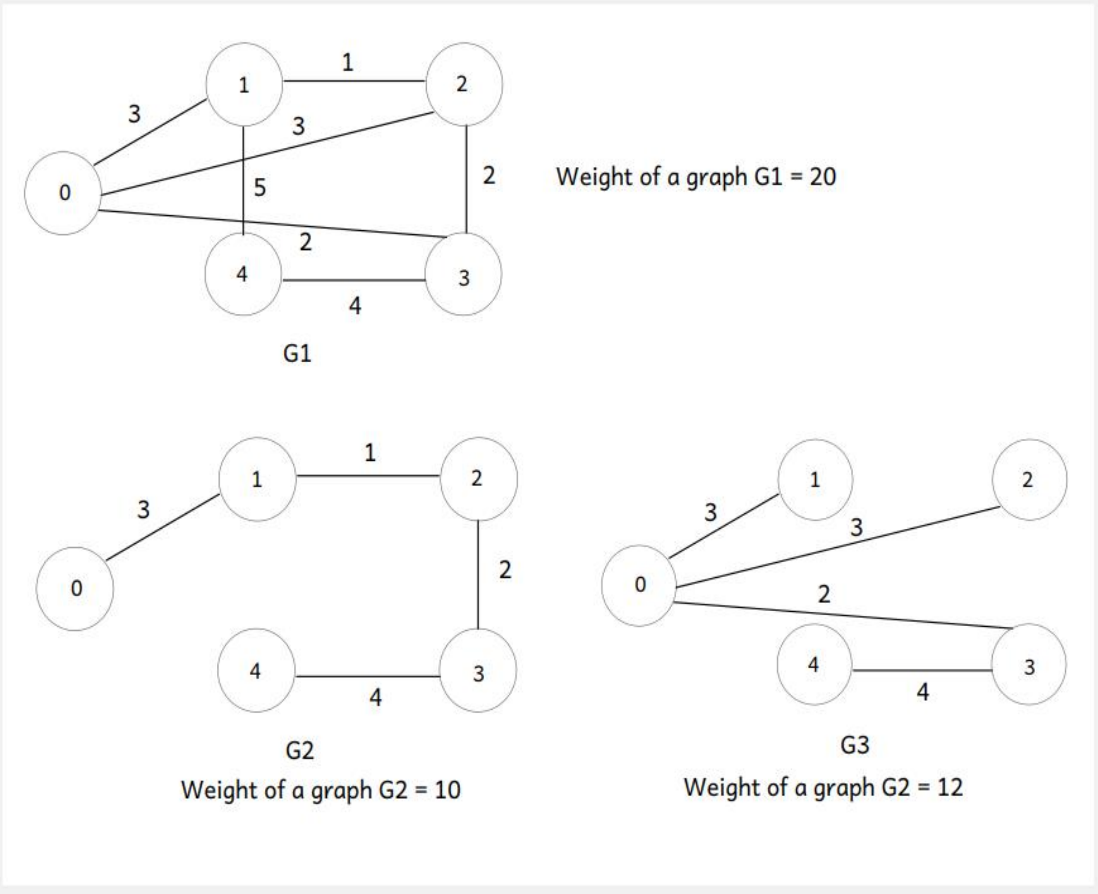
~~5~~
~~2~~
~~0~~
~~3~~
~~1~~
~~4~~

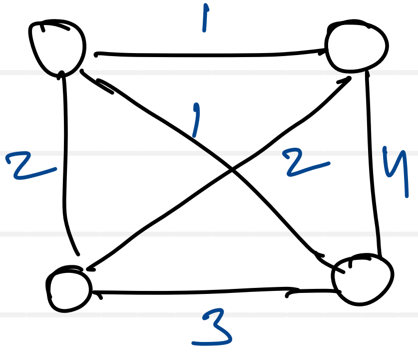
Length	2	1	2	1	0	2
	0	1	2	3	4	5

Path length tree : (4-1), (4-3), (1-0) (1-2) (3-5)

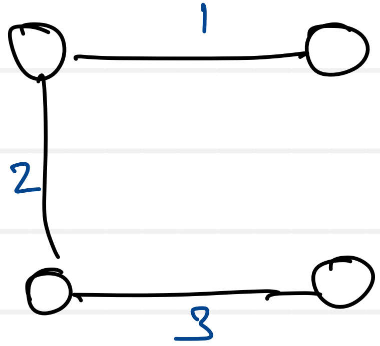
Spanning Tree

- Tree is a graph without cycles. Includes all V vertices and V-1 edges.
- Spanning tree is connected sub-graph of the given graph that contains all the vertices and sub-set of edges.
- Spanning tree can be created by removing few edges from the graph which are causing cycles to form.
- One graph can have multiple different spanning trees.
- In weighted graph, spanning tree can be made who has minimum weight (sum of weights of edges). Such spanning tree is called as Minimum Spanning Tree.
- Spanning tree can be made by various algorithms.
  - BFS Spanning tree
  - DFS Spanning tree
  - Prim's MST — Minimal Spanning Tree
  - Kruskal's MST

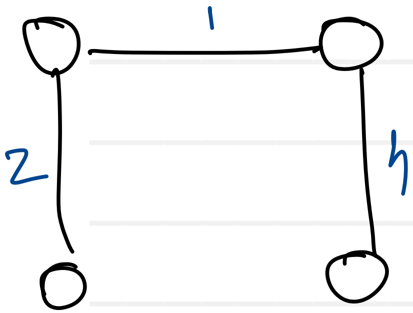




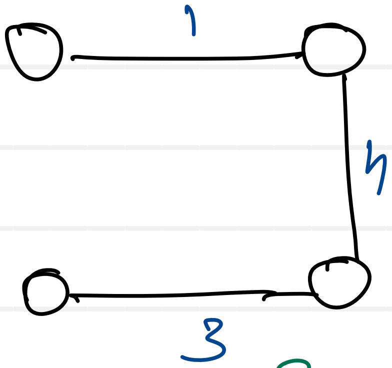
Weight = 13



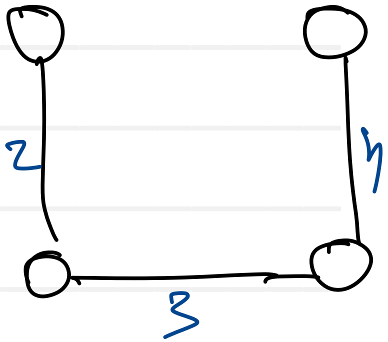
Wt = 6



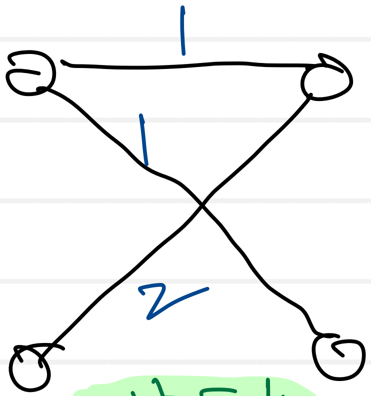
wt = 7



wt = 8

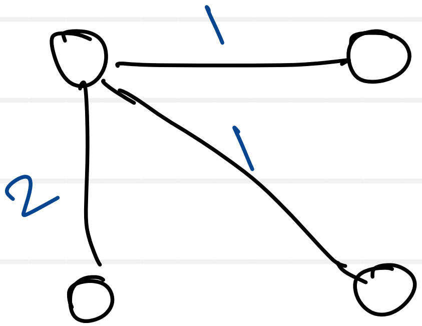


wt = 9



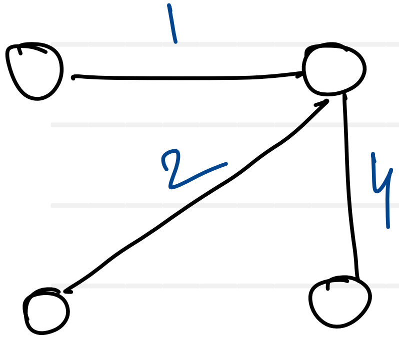
wt = 4

Minimum Spanning Tree

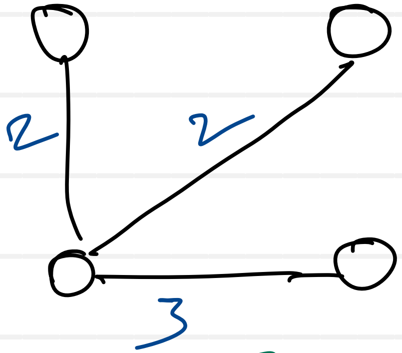


wt = 4

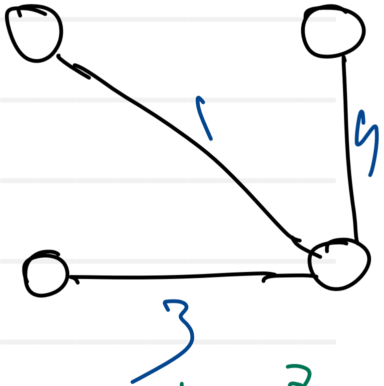
Minimum Spanning Tree



Wt = 7

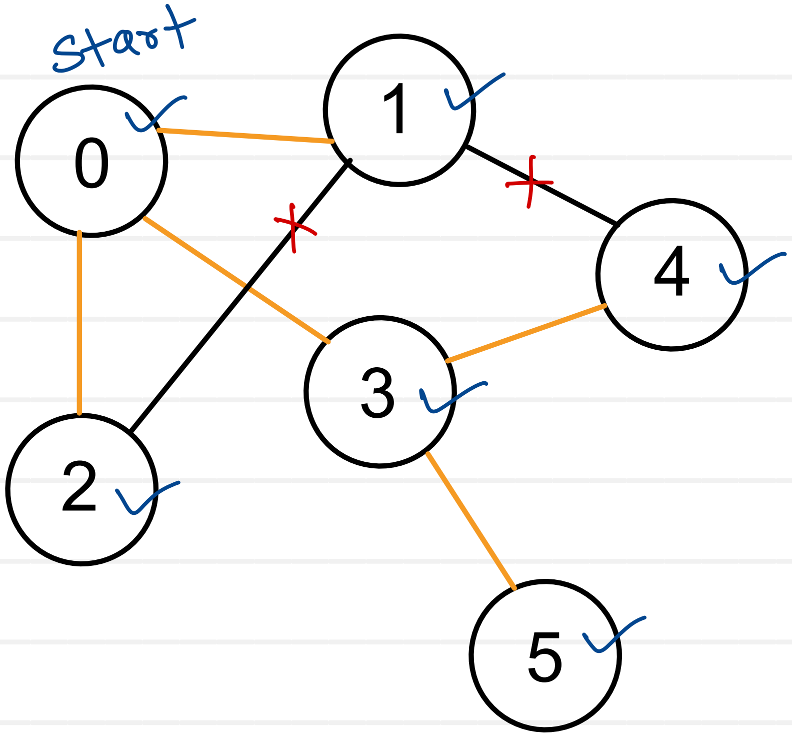


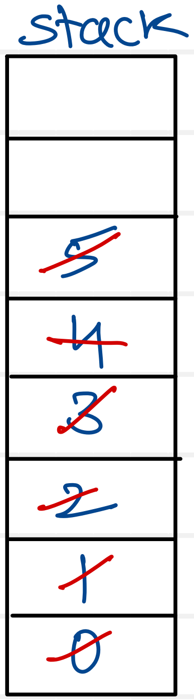
wt = 7



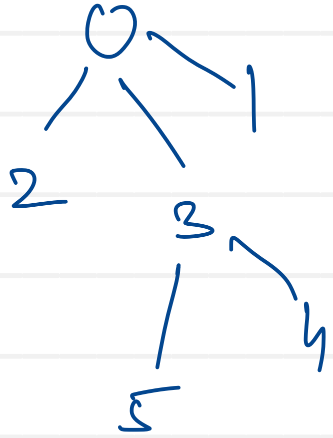
wt = 8

DFS Spanning Tree





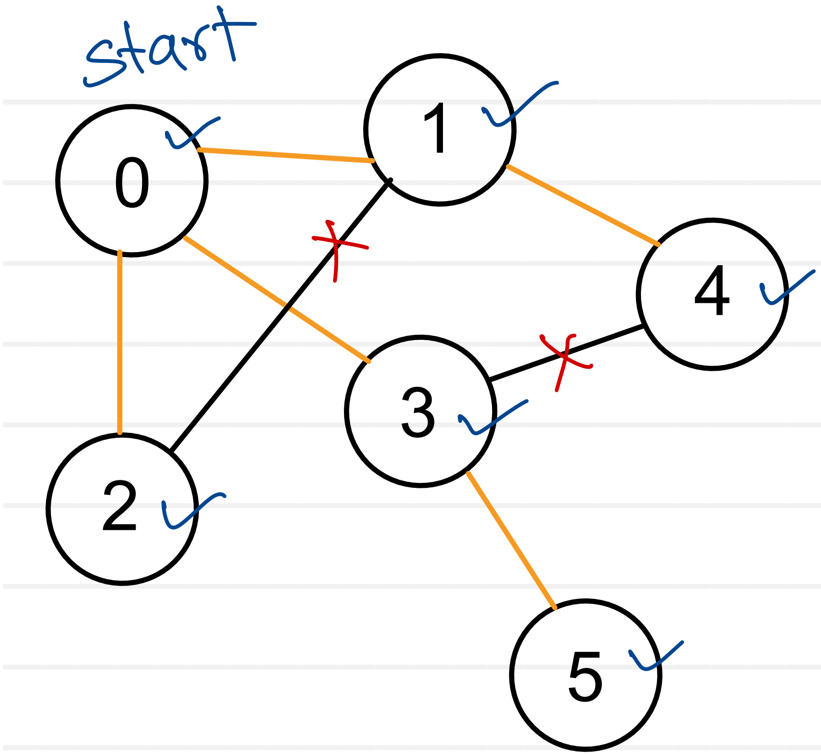
1. push starting vertex on stack & mark it.
2. pop the vertex.
3. push all its non-marked neighbors on the stack, mark them and also print the vertex to neighboring vertex edges.
4. repeat steps 2-3 until stack is empty.

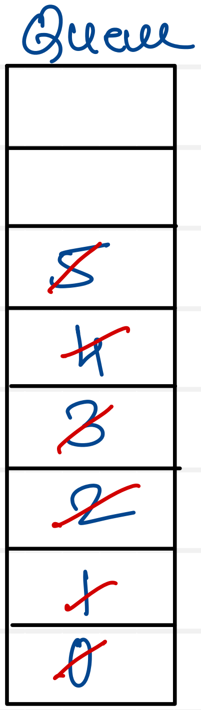


Spanning tree : (0,1) (0,2) (0,3) (3,4) (3,5)

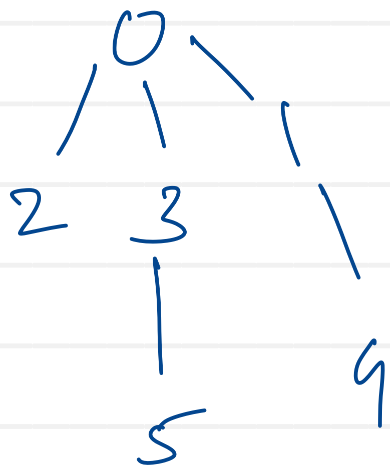
$\boxed{T(V) = O(V^2)}$

BFS Spanning Tree





1. push starting vertex on queue & mark it.
2. pop the vertex.
3. push all its non-marked neighbors on the queue, mark them and also print the vertex to neighboring vertex edges.
4. repeat steps 2-3 until queue is empty.

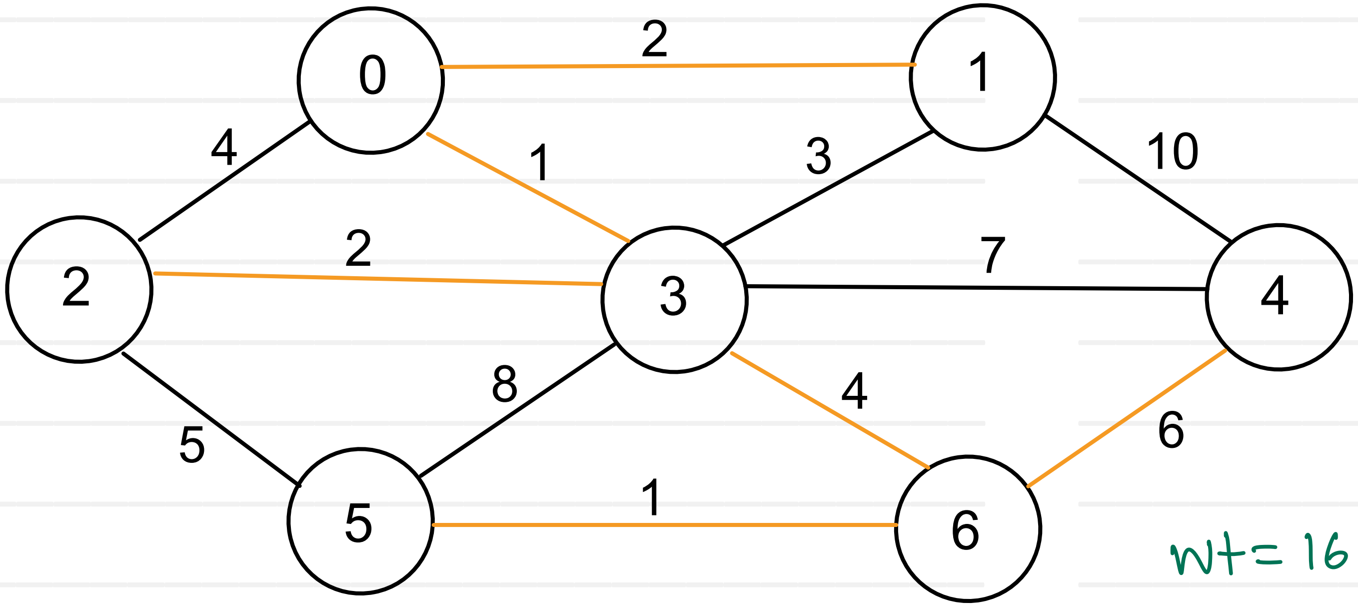


Spanning tree : (0-1) (0-2) (0-3) (1-4) (3-5)

$\boxed{T(V) = O(V^2)}$

Kruskal's Algorithm (MST)

1. Sort all the edges in ascending order of their weight.
2. Pick the smallest edge.
   Check if it forms a cycle with the spanning tree formed so far.
   If cycle is not formed, include this edge.
   Else, discard it.
3. Repeat step 2 until there are (V-1) edges in the spanning tree.



wt = 16

s	d	wt	
0	3	1	✓
6	5	1	✓
0	1	2	✓
3	2	2	✓
1	3	3	✗
2	0	4	✗
3	6	4	✓
2	5	5	✗
4	6	6	✓
3	4	7	
3	5	8	
1	4	10	

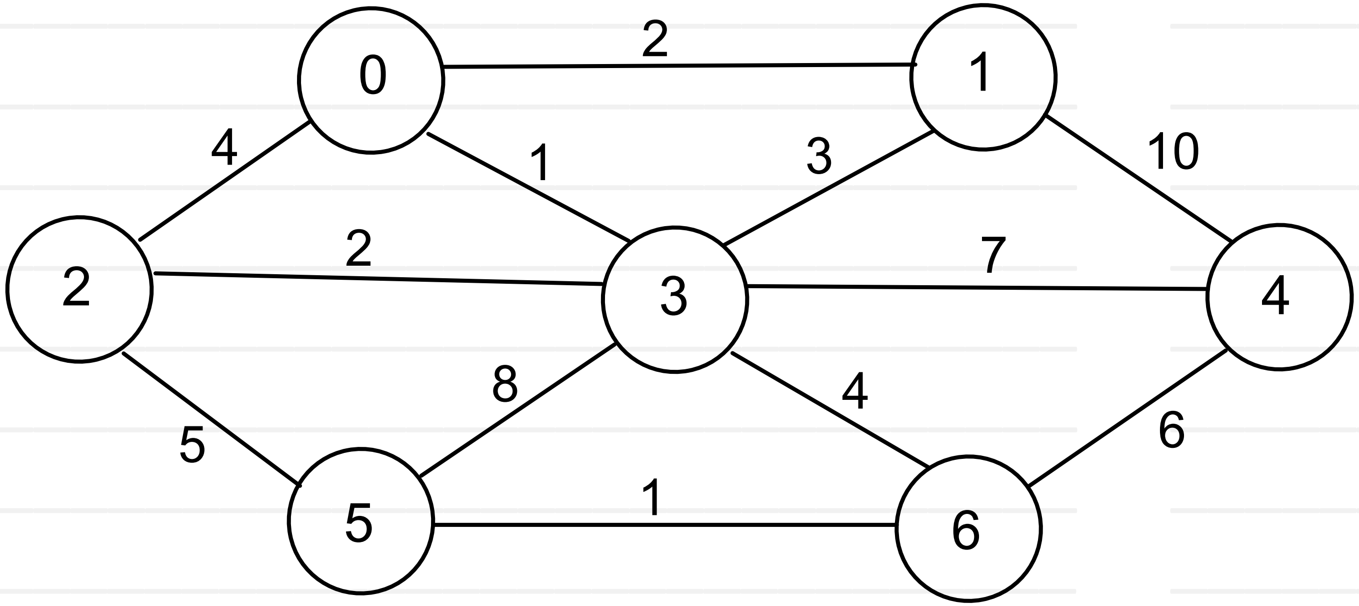
V = 7

V-1 edges

6

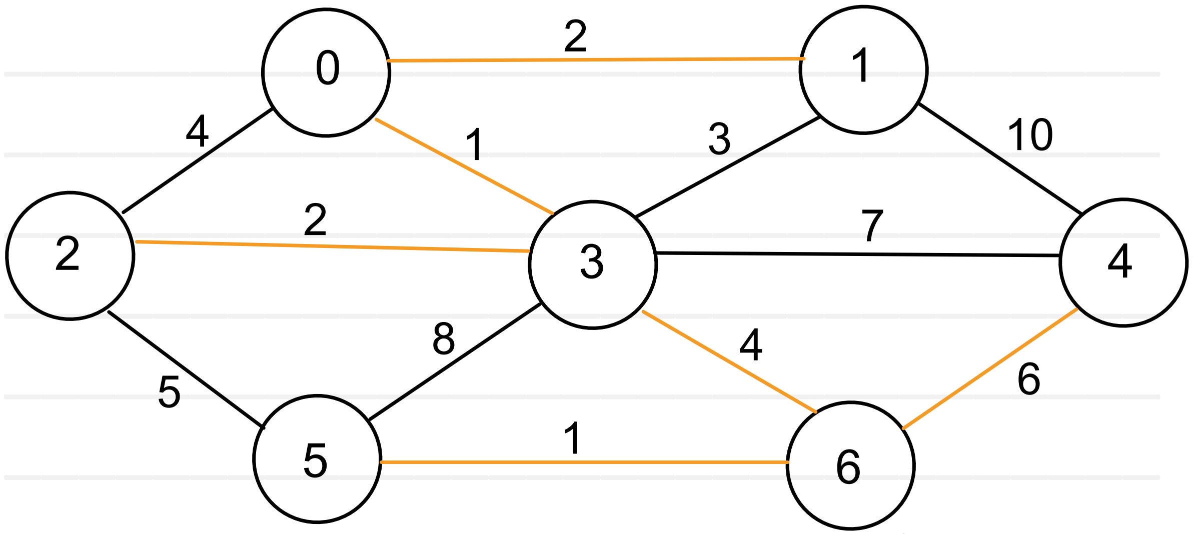
Union Find Algorithm

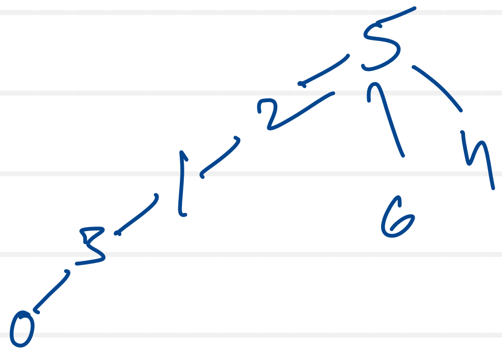
1. Consider all vertices as disjoint sets (parent = -1).
2. For each edge in the ~~graph~~ MST
    1. Find set(root) of first vertex. (src)
    2. Find set(root) of second vertex. (dest)
    3. If both are in same set(same root), cycle is detected.
    4. Otherwise, merge(Union) both the sets i.e. add root of first set under second set
       src dest



Union Find Algorithm

	0	1	2	3	4	5	6
Parents	3	2	5	1	5	-1	5



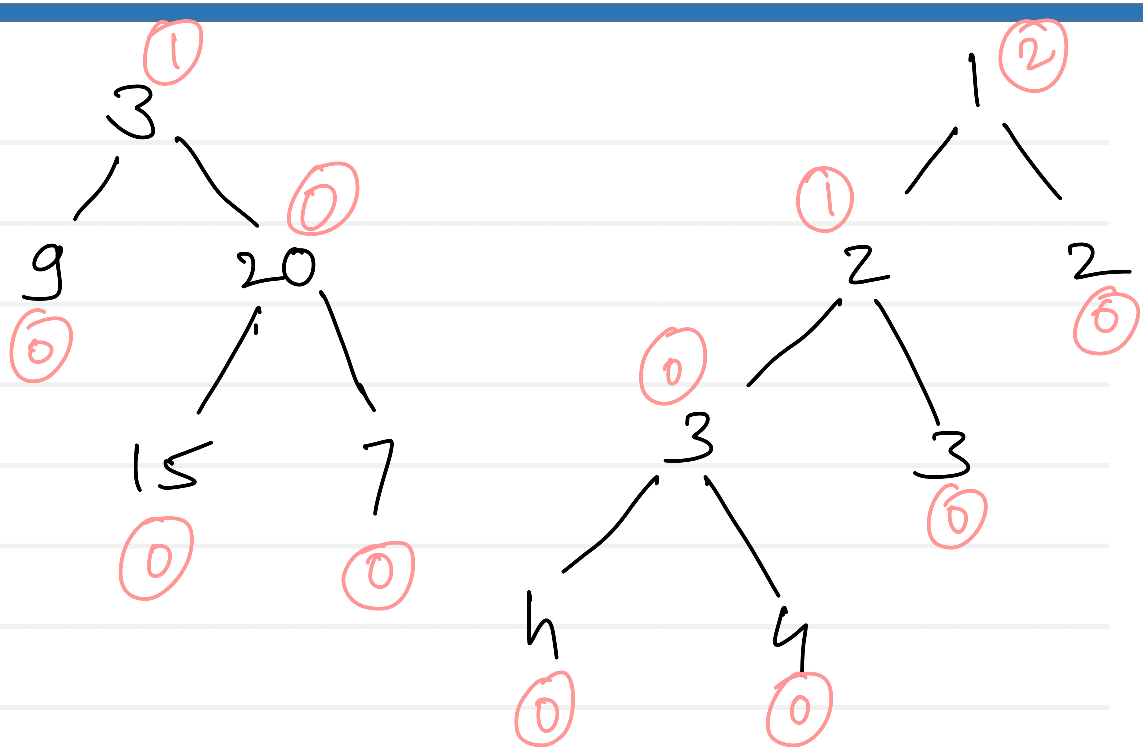


sr	dr	s	d	wt
0	3	0	3	1
6	5	6	5	1
3	1	0	1	2
1	2	3	2	2
2	2	1	3	3
2	2	2	0	4
2	5	3	6	4
5	5	2	5	5
4	5	4	6	6
		3	4	7
		3	5	8
		1	4	1

```
int find(int v, int parent[]){
    while(parent[v] != -1)
        v = parent[v];
    return v;
}
```

```
void union(int sr, int dr, int parent[]){
    parent[sr] = dr;
}
```

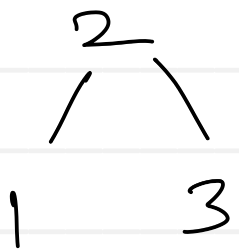
Balanced binary tree

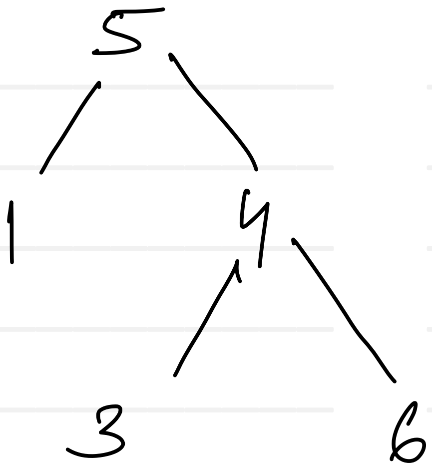


```
boolean isBalanced(root) {
    return height(root) != -1;
}
```

```
int height(Node trav) {
    if(trav == null) return 0;
    int hl = height(trav.left);
    int hr = height(trav.right);

    if(hl == -1 || hr == -1)
        return -1;

    if(Math.abs(hl - hr) > 1)
        return -1;

    int max = hl > hr ? hl : hr;
    return max + 1;
}
```

Validate binary search tree

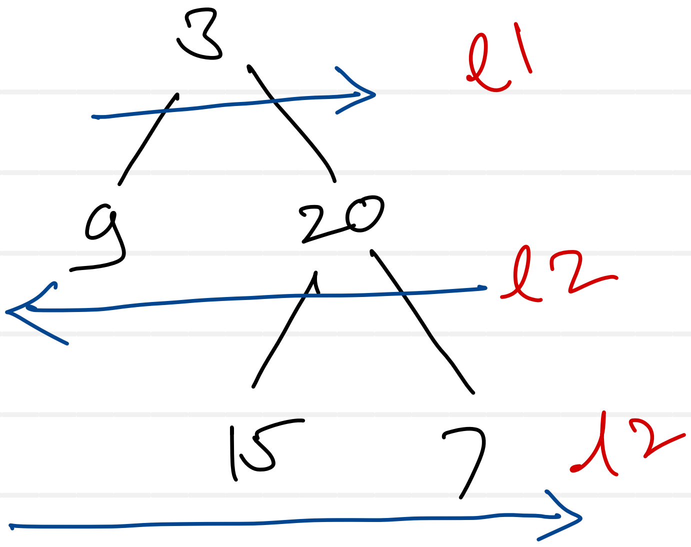


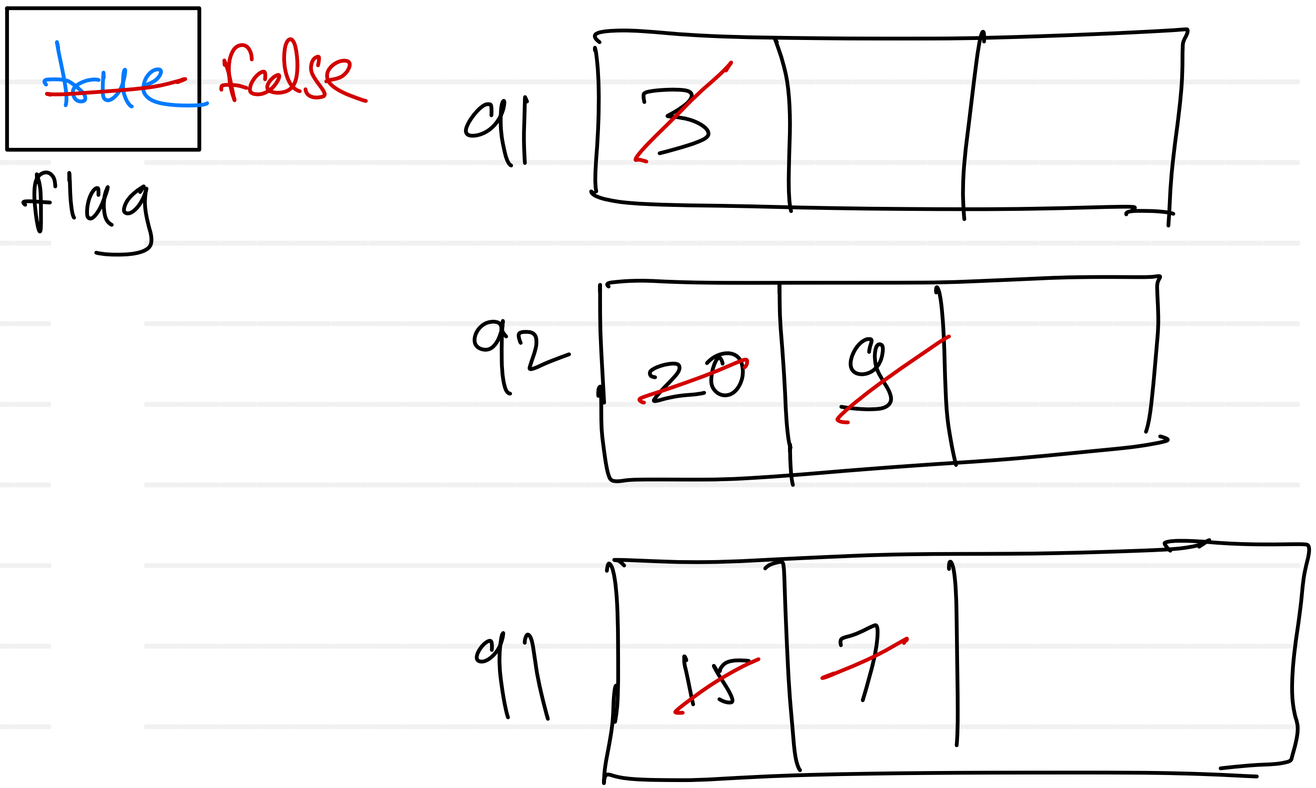


	l	r
2	1	3
1	null	null
3	null	null

	l	r
5	1	4 ✗

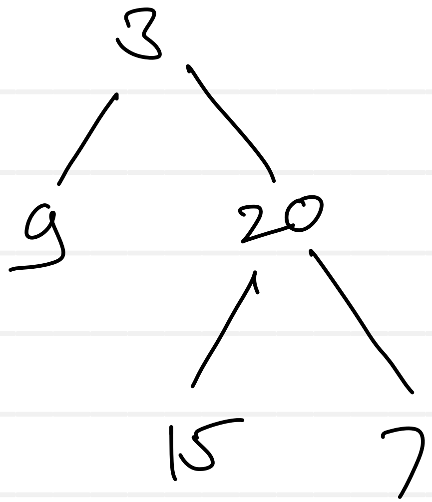
Binary tree zigzag level order traversal





Output: [3], [20, 9], [15, 7]

Minimum depth of binary tree



Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com