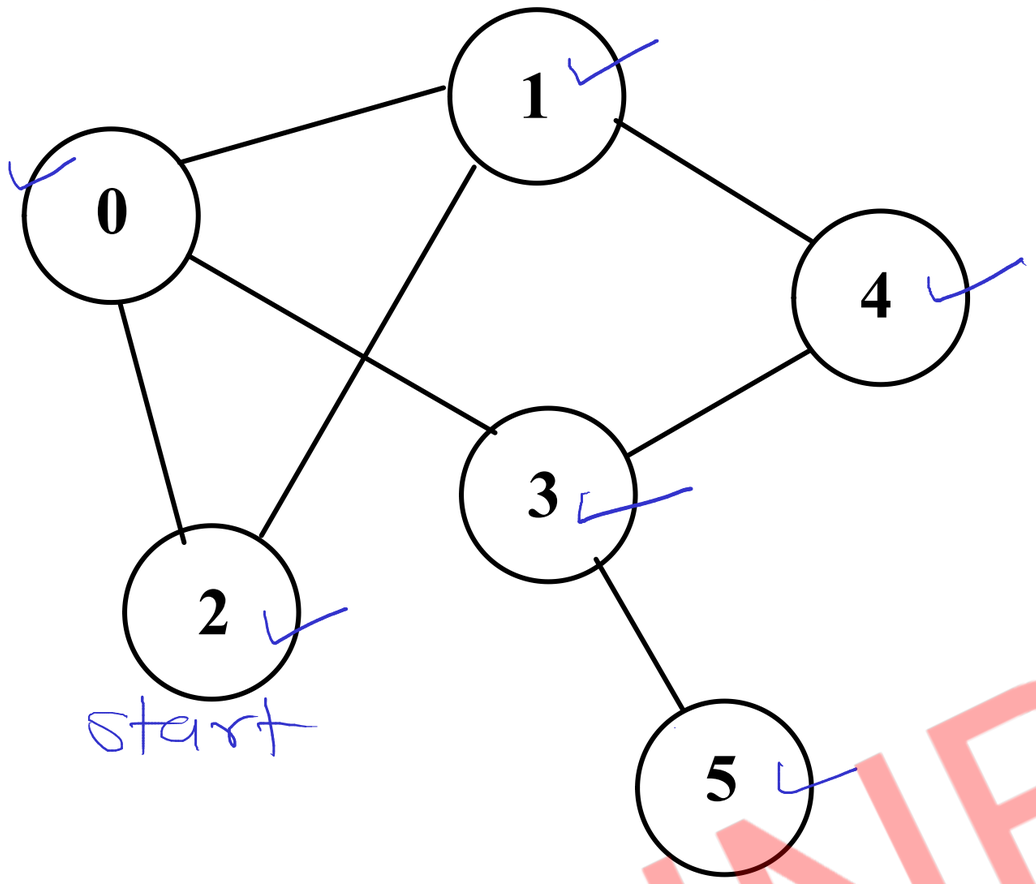
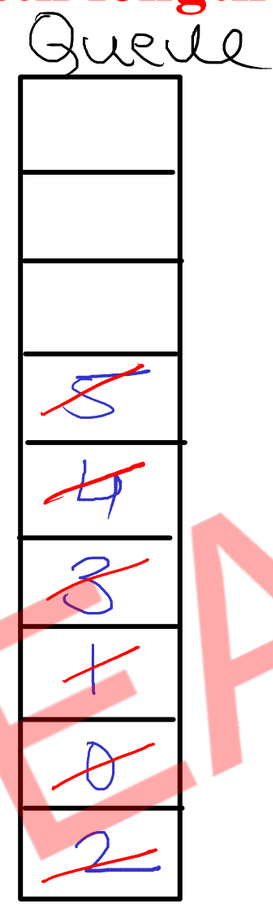
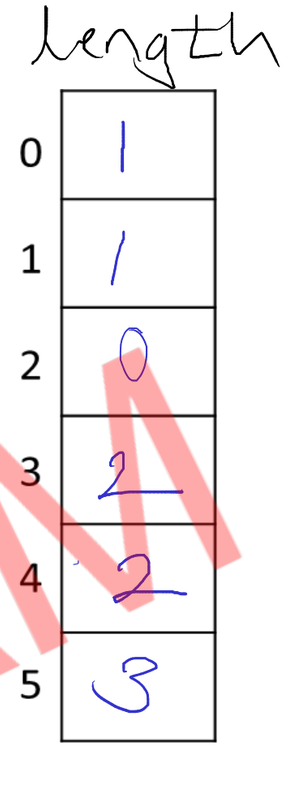
Single Source Path length







Edges

(2-0)
(2-1)

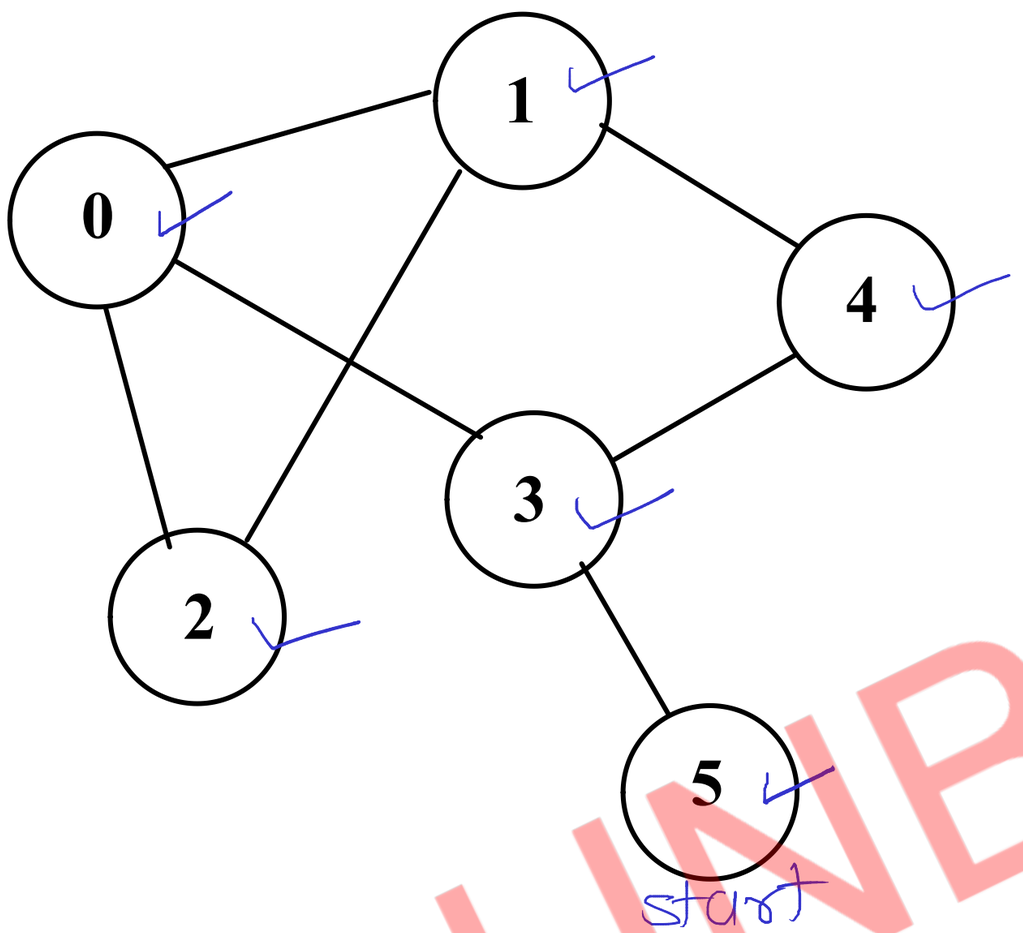
(0-3)

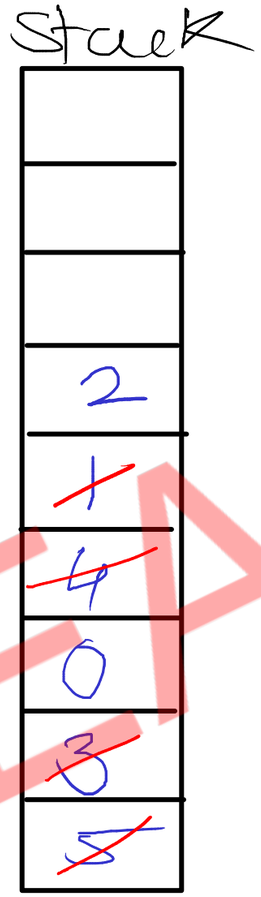
(1-4)

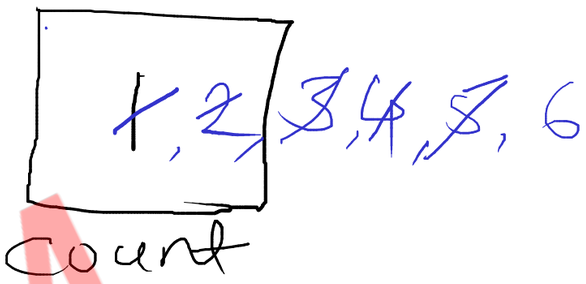
(3-5)

//1. Create path length array to keep length of vertex from start vertex.
//2. push start on queue & mark it.
//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.

Check Connected-ness



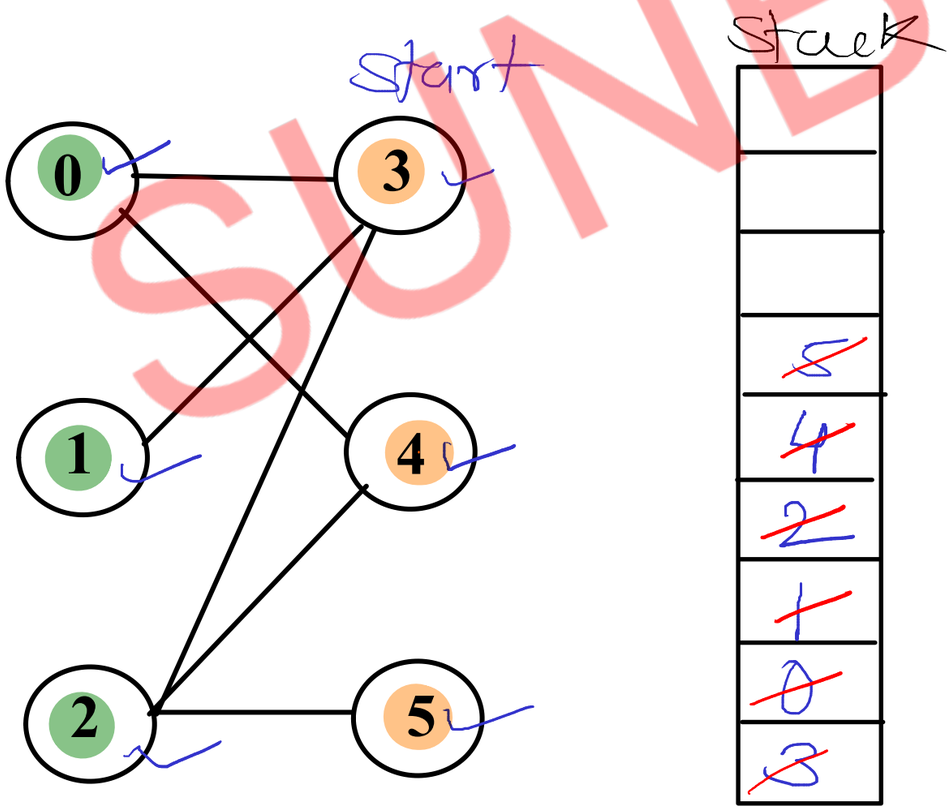




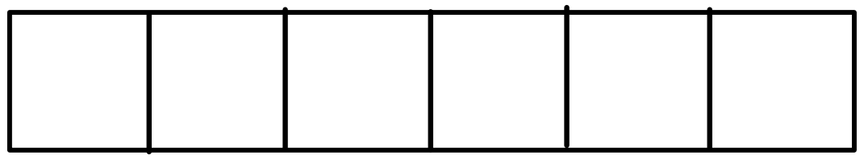
//1. push start on stack & mark it.
//2. begin counting marked vertices from 1.
//3. pop and print a vertex.
//4. push all its non-marked neighbors on the stack, mark them and increment count.
//5. if count is same as number of vertex, graph is connected (return).
//6. repeat steps 3-5 until stack is empty.
//7. graph is not connected (return)

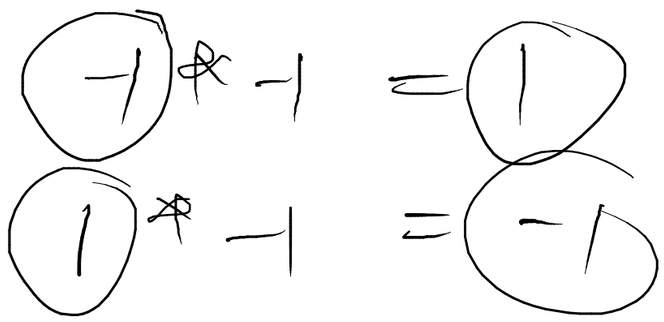
Check Bipartite-ness

//1. keep colors of all vertices in an array. Initially vertices have no color.
//2. push start on queue & mark it. Assign it color1.
//3. pop the vertex.
//4. push all its neighbors on the queue
//5. For each such vertex if no color is assigned yet, assign opposite color of current vertex (c1-c2, c2-c1).
//6. If vertex is already colored with same of current vertex, graph is not bipartite (return).
//7. repeat steps 3-6 until queue is empty.



color1 = -1, color2 = 1, no color = 0

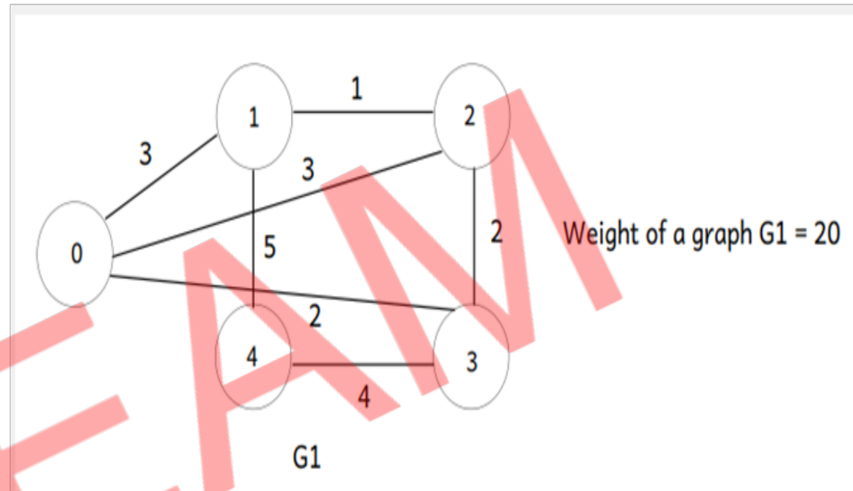


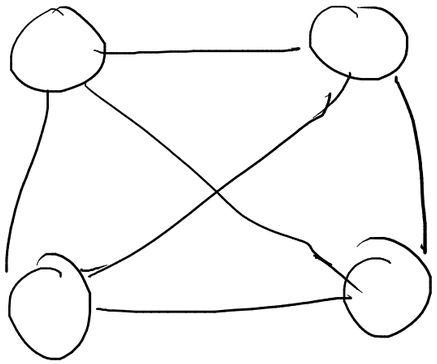


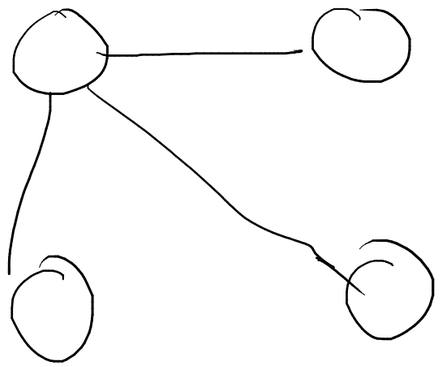
Spanning Tree

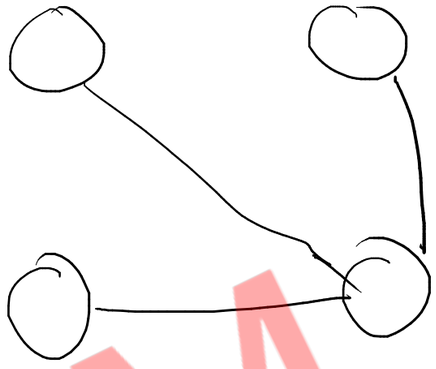
- Tree is a graph without cycles. Includes all V vertices and V-1 edges.
- Spanning tree is connected sub-graph of the given graph that contains all the vertices and sub-set of edges.
- Spanning tree can be created by removing few edges from the graph which are causing cycles to form.
- One graph can 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
  - Kruskal's MST

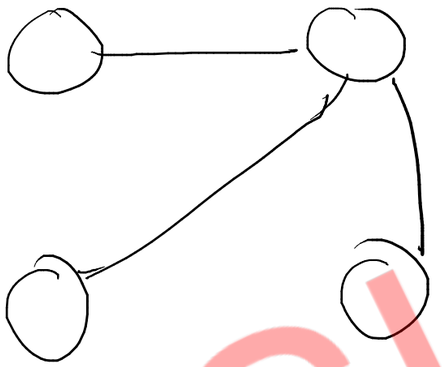
Minimum Spacing Tree

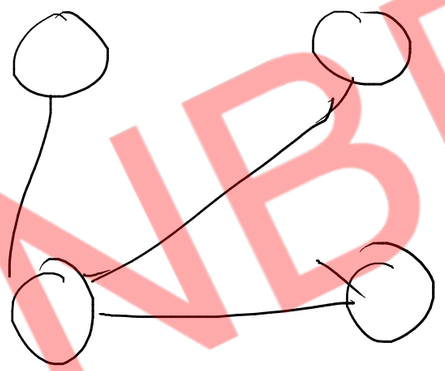


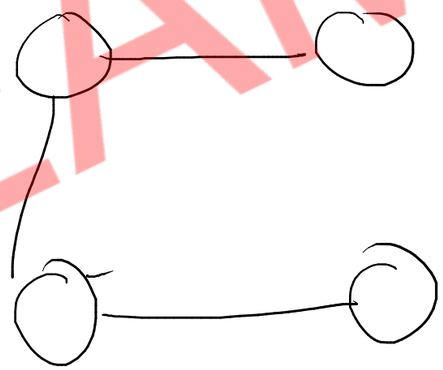


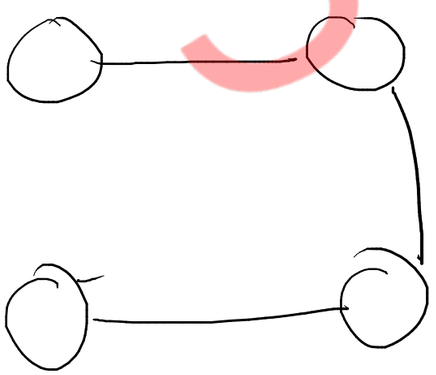


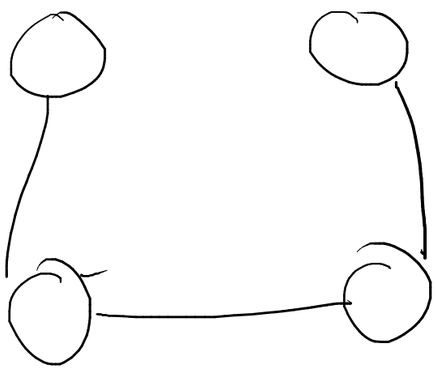


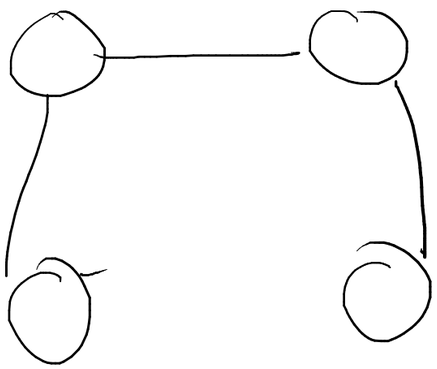




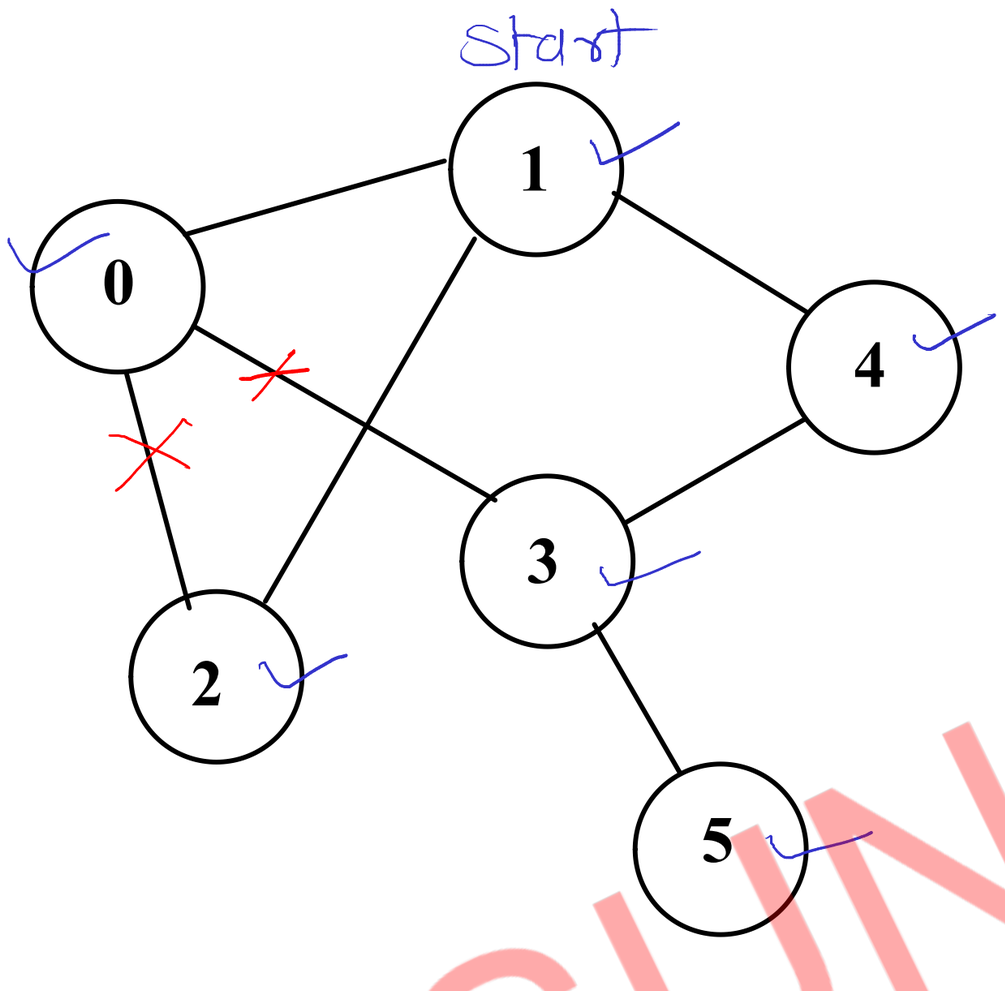


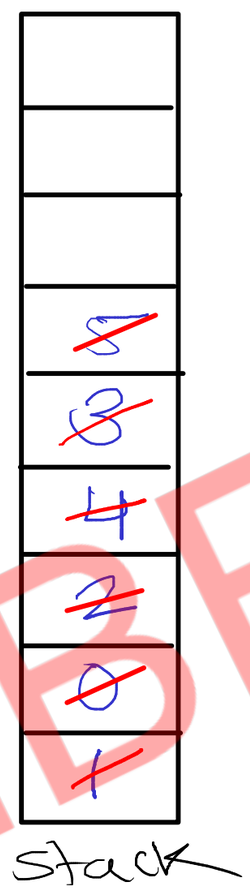


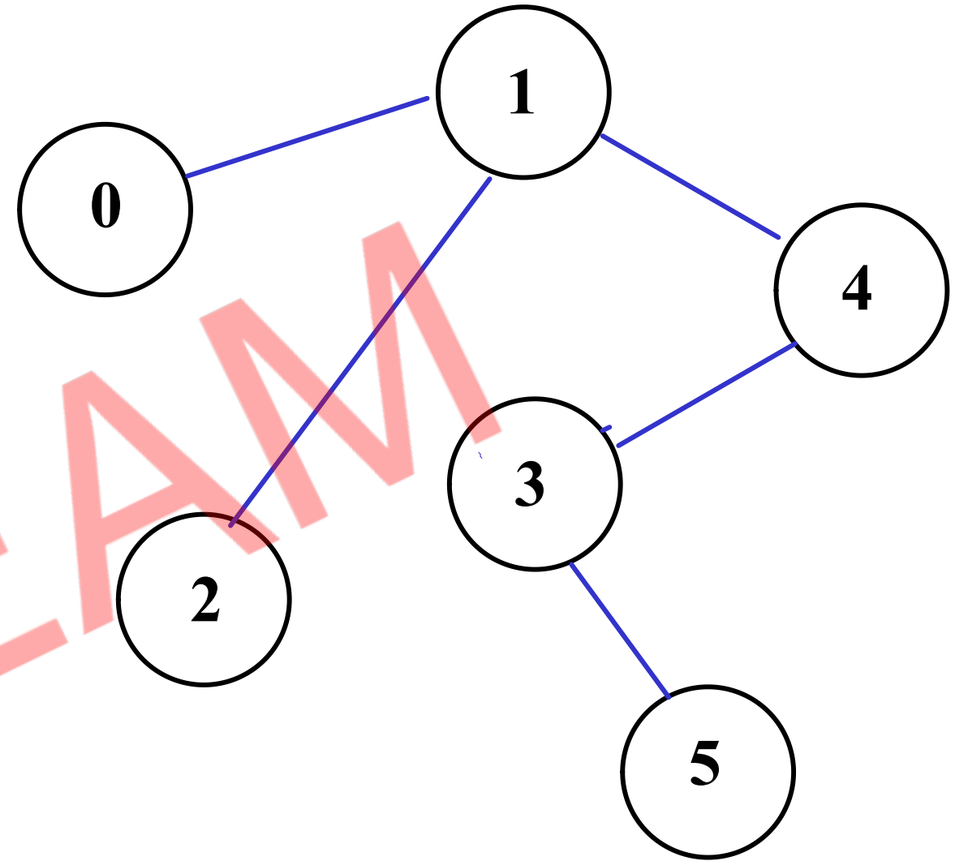




DFS Spanning Tree



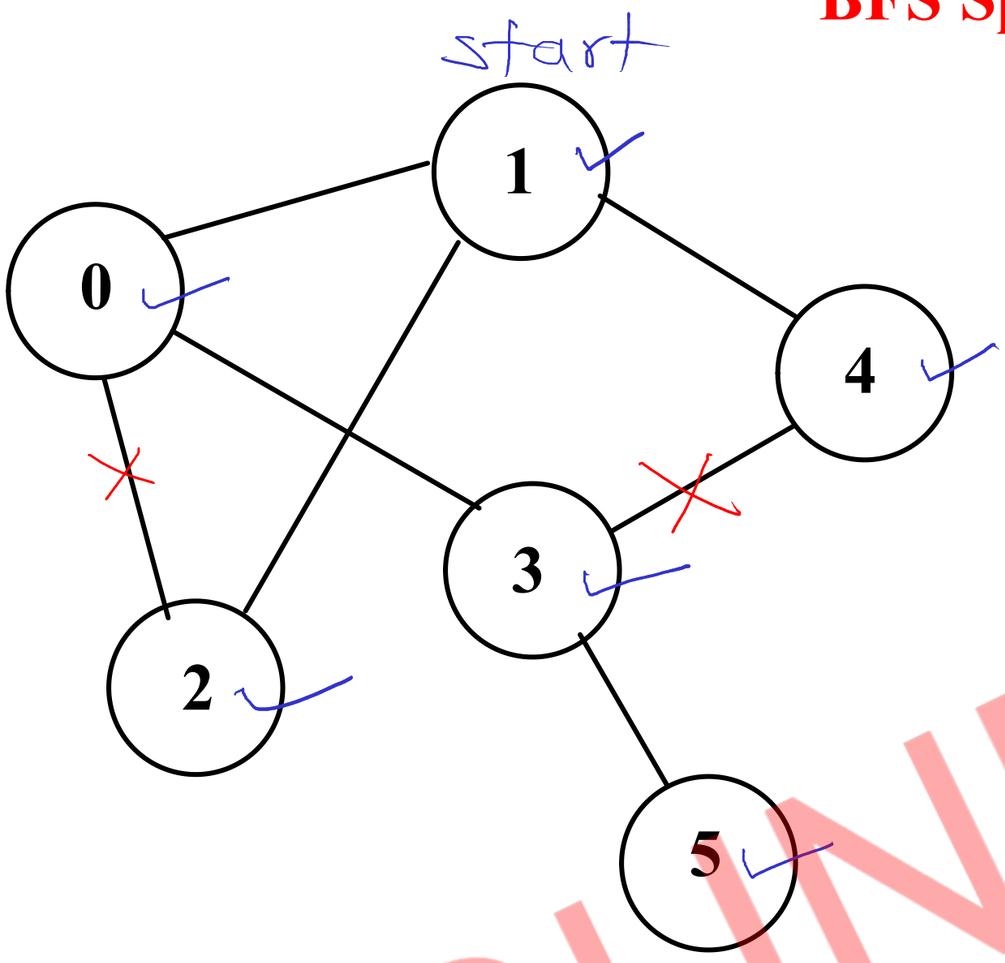


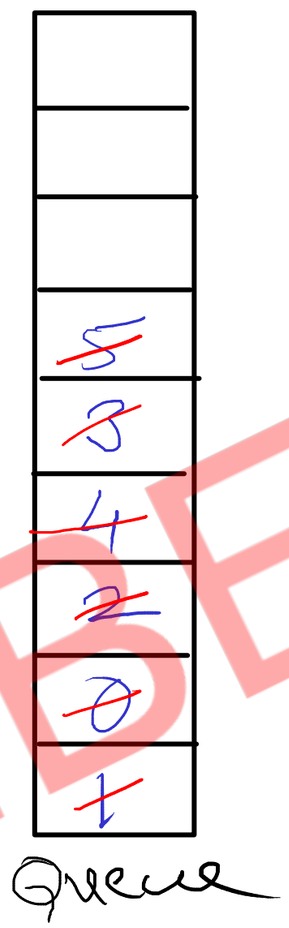


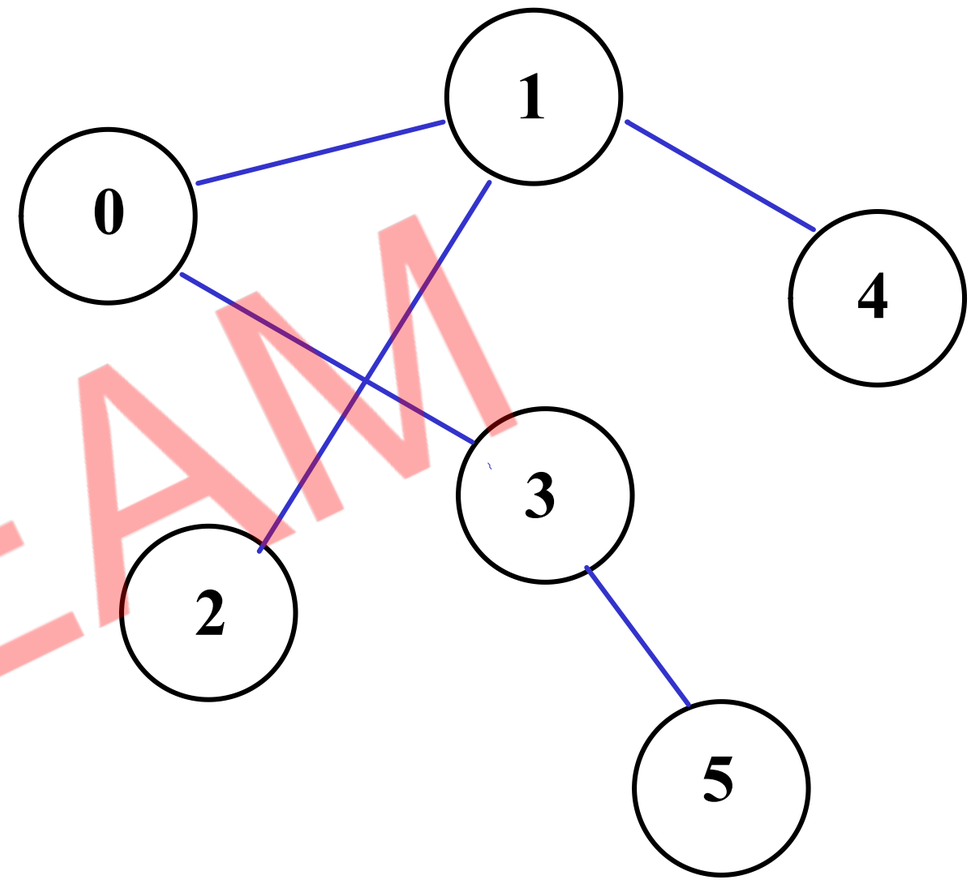
Spanning Tree : (1-0) (1-2) (1-4) (4-3) (3-5)

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

BFS Spanning Tree





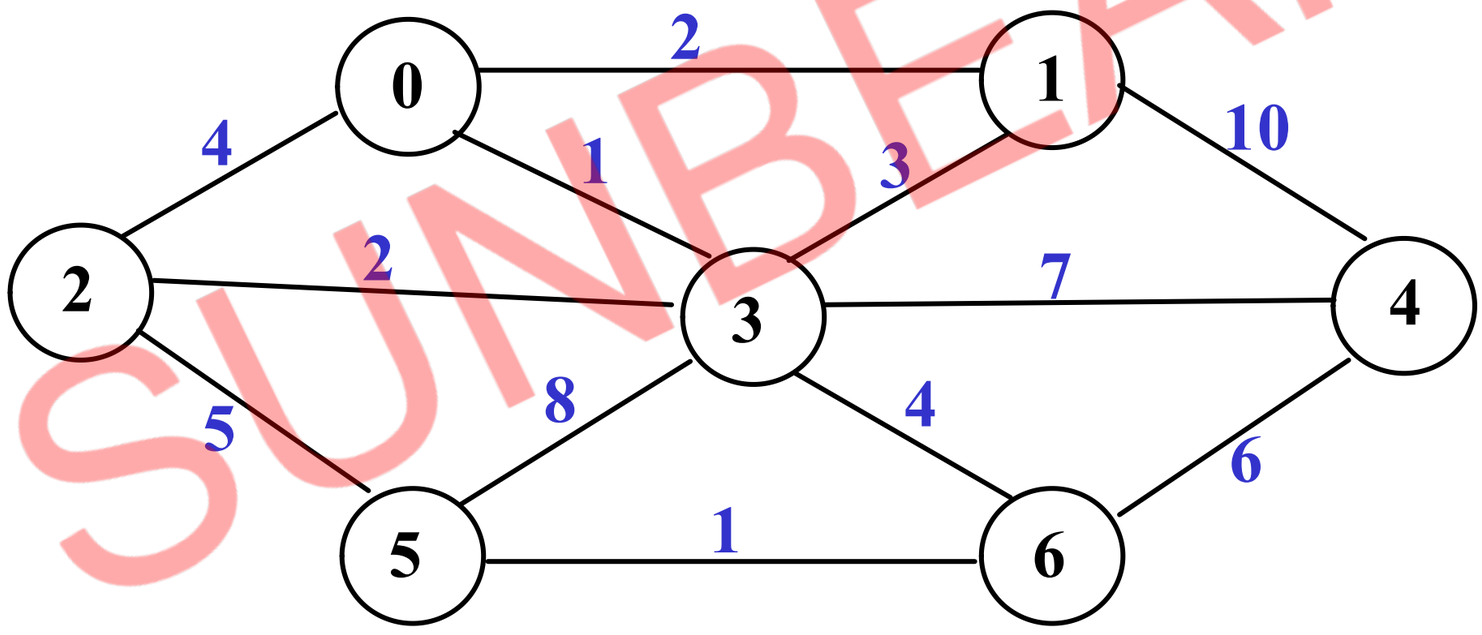


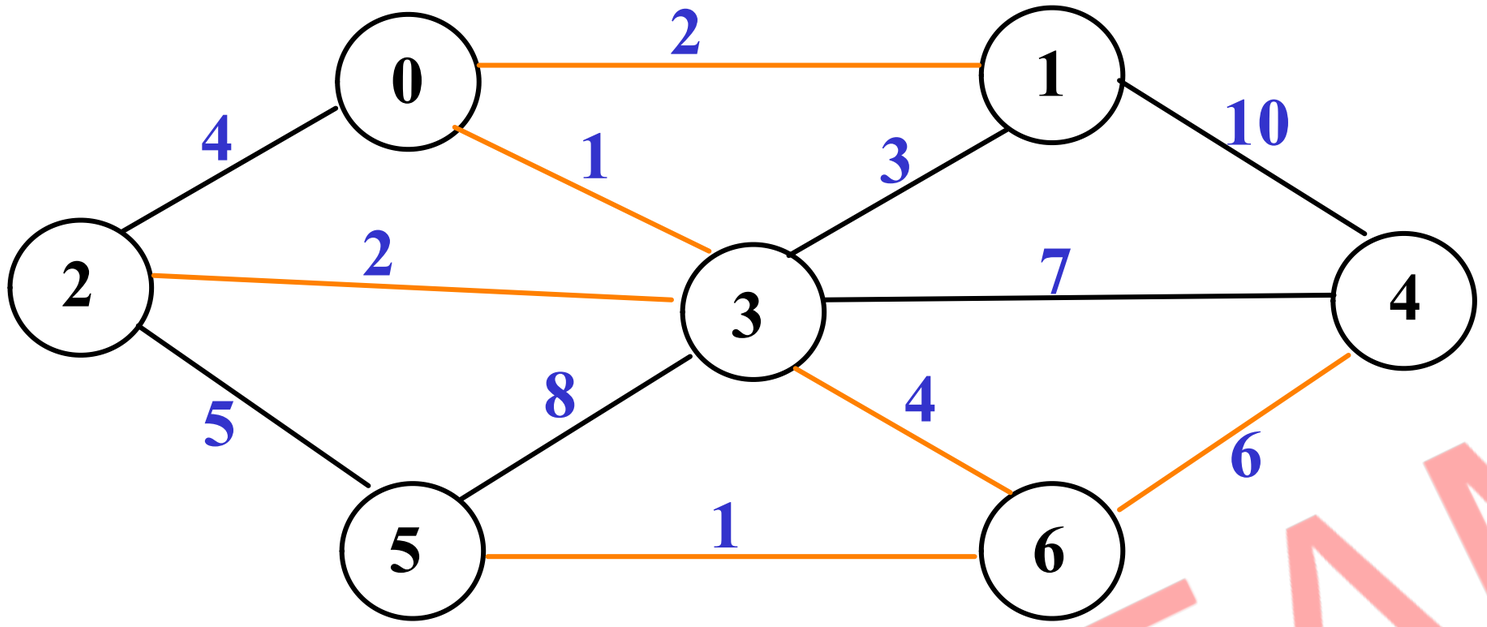
Spanning Tree : (1-0)(1-2)(1-4)(0-3)(3-5)

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

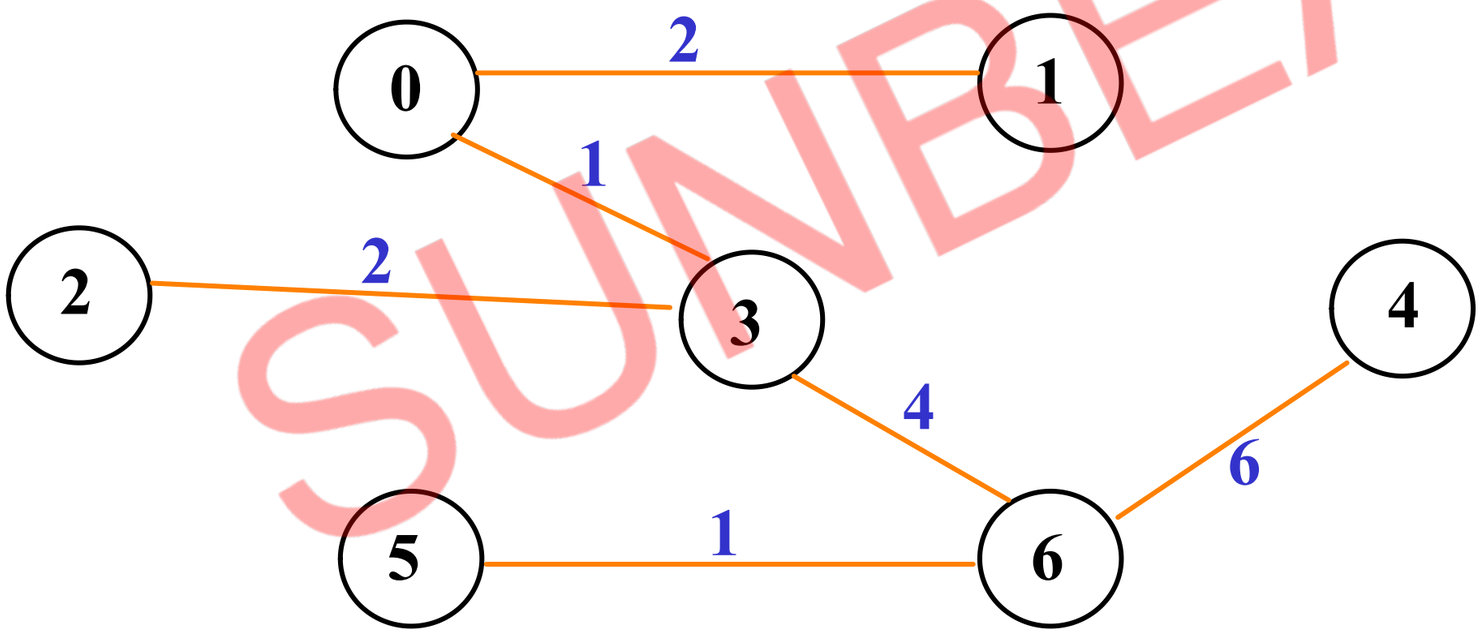
Kruskal's MST

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.





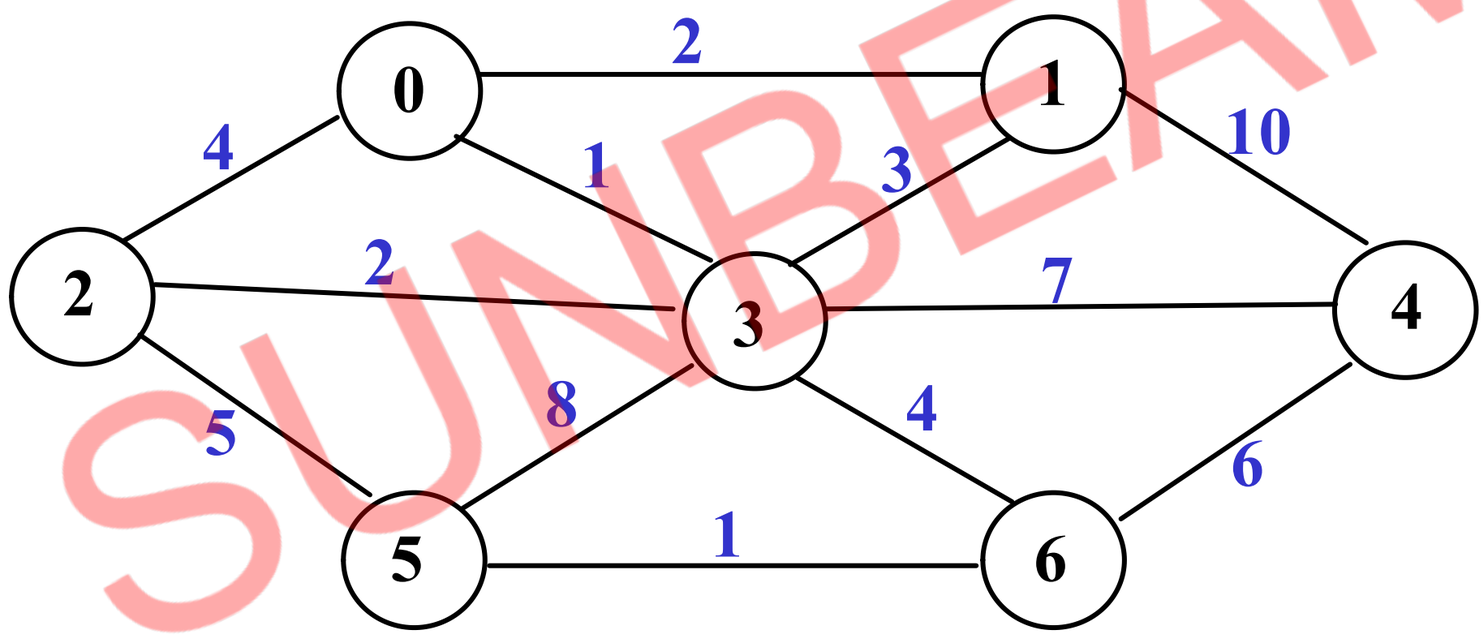
0 3 - 1 ✓
6 5 - 1 ✓
0 1 - 2 ✓
3 2 - 2 ✓
1 3 - 3 ✗
2 0 - 4 ✗
3 6 - 4 ✓
2 5 - 5 ✗
4 6 - 6 ✓
3 4 - 7
3 5 - 8
1 4 - 10



Weight = 16

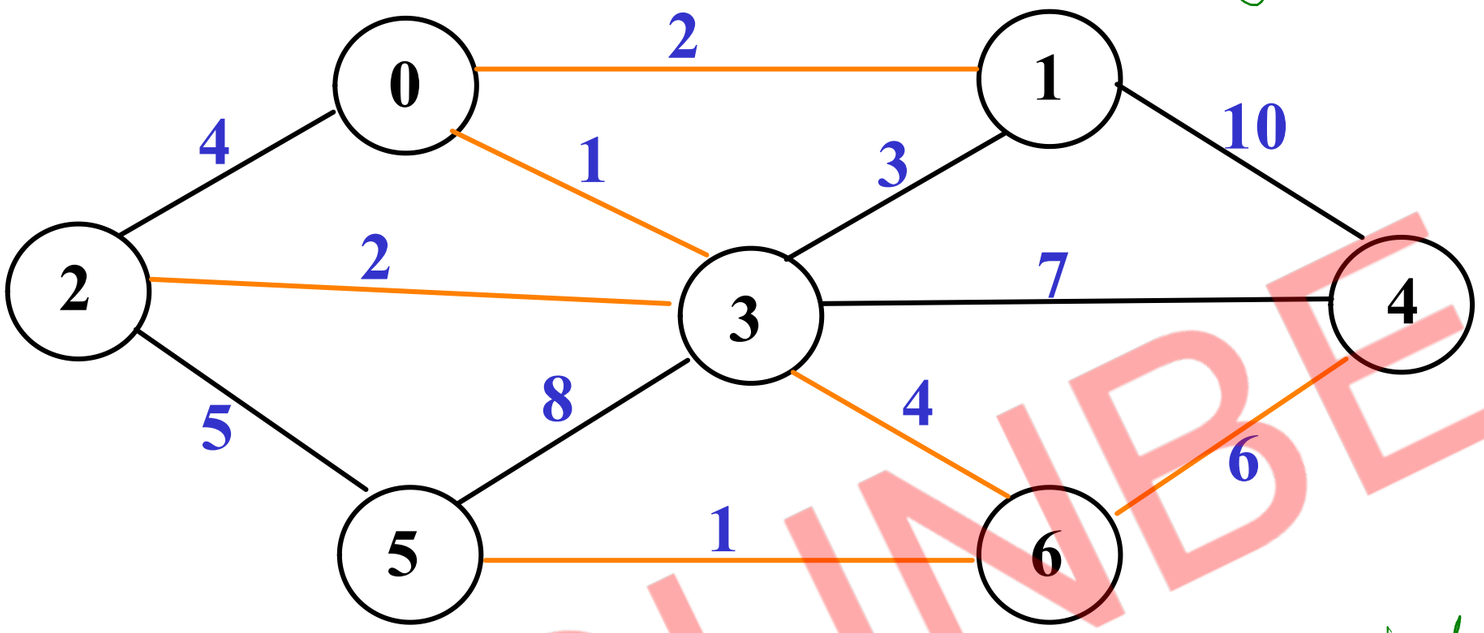
Union Find Algorithm

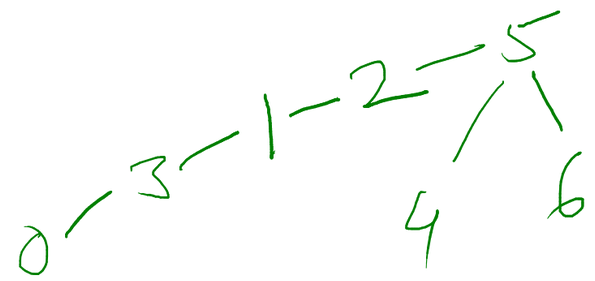
1. Consider all vertices as disjoint sets (parent = -1).
2. For each edge in the graph
    1. Find set(root) of first vertex.
    2. Find set(root) of second vertex.
    3. If both are in same set(same root), cycle is detected.
    4. Otherwise, merge(Union) both the sets i.e. add root of first set under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



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

Parent:





sr	dr	sd
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 x
2	2	2 0 - 4 x
2	5	3 6 - 4
5	5	2 5 - 5
4	5	4 6 - 6
		3 4 - 7
		3 5 - 8
		1 4 - 10

```
int find(int v, int parent[7])
{
    while(parent[v] != -1)
        v = parent[v]
    return v;
}
```

```
void union(int sr, int dr,
           int parent[7])
{
    parent[sr] = dr;
}
```