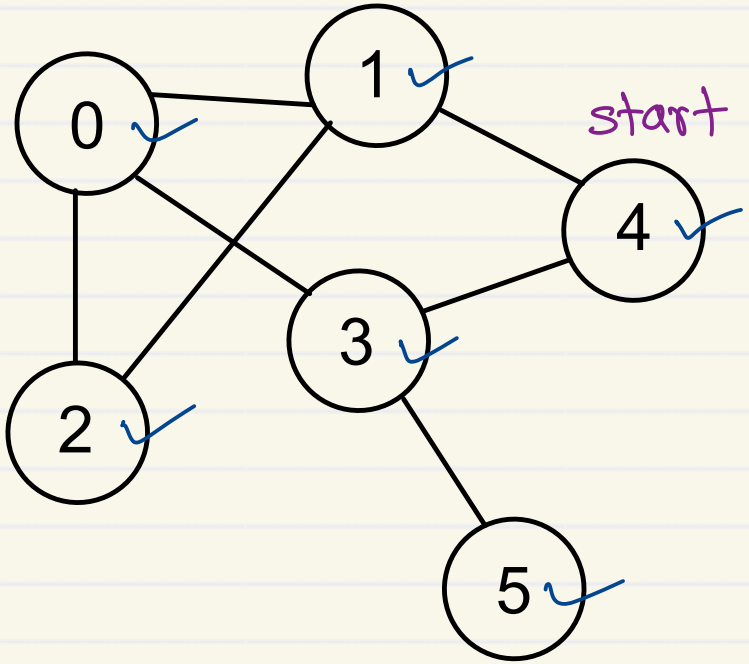
Sunbeam Institute of Information Technology
Pune and Karad

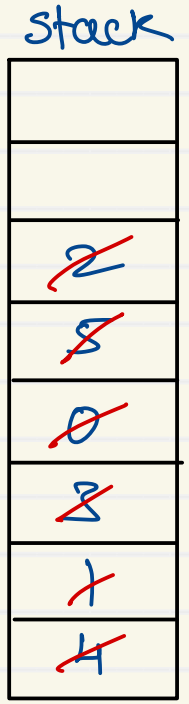
Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

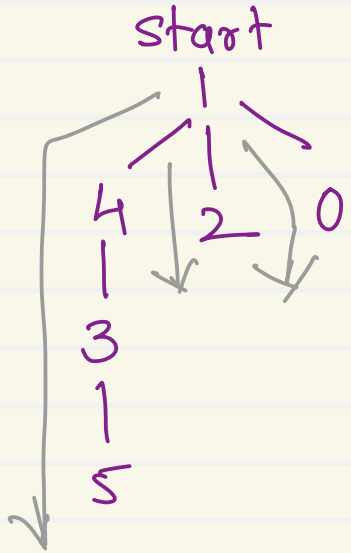
DFS traversal



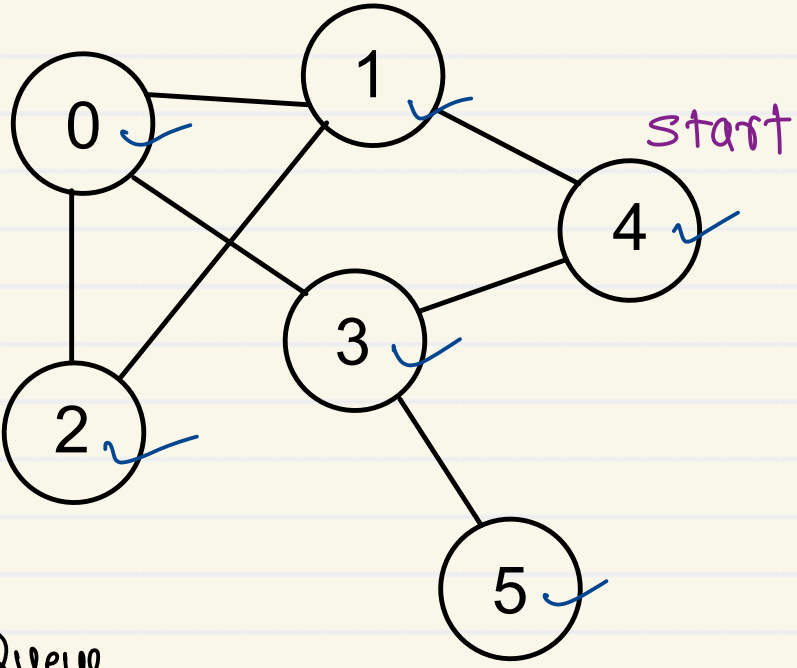


Traversal : 4, 3, 5, 0, 2, 1

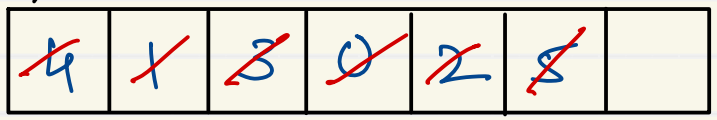
1. Choose a vertex as start vertex.
2. Push start vertex on stack and mark it.
3. Pop vertex from stack.
4. Visit/print popped vertex.
5. Push all non marked neighbours of the vertex on the stack and mark them.
6. Repeat step 3 - 5 until stack is empty.



BFS traversal



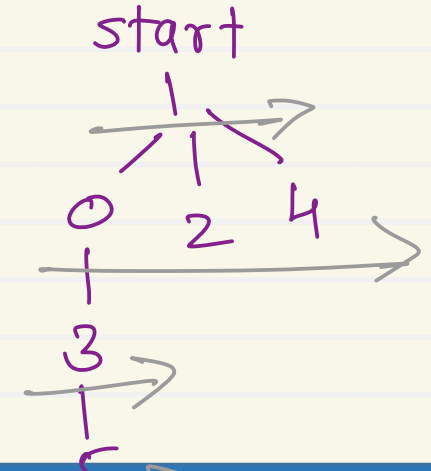
Queue



Traversal : 4, 1, 3, 0, 2, 5

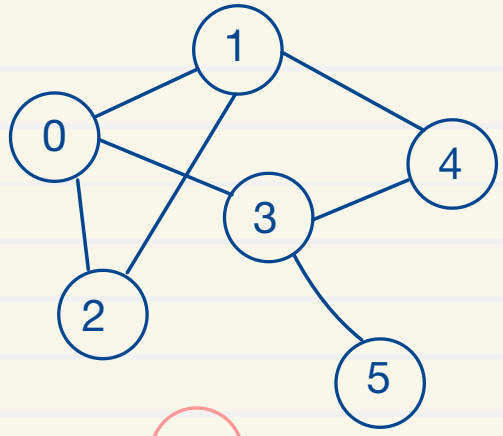
1. Choose a vertex as start vertex.
2. Push start vertex on queue and mark it.
3. Pop vertex from queue .
4. Visit/print popped vertex.
5. Push all non marked neighbours of the vertex on the queue and mark them.
6. Repeat step 3 - 5 until queue is empty.

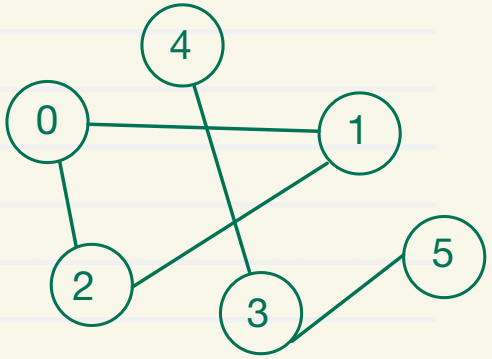
- BFS is also referred as level wise search algorithm



Graph

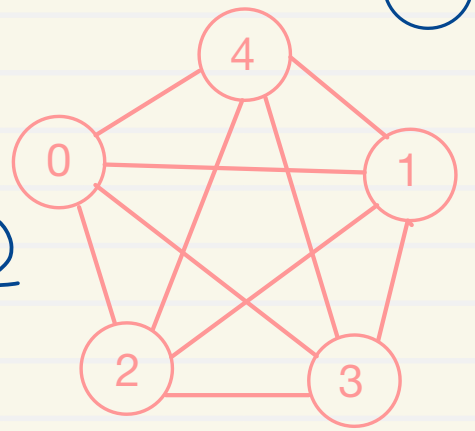
- **Connected graph**
  - From each vertex some path exists for every other vertex.
  - Can traverse the entire graph starting from any vertex.

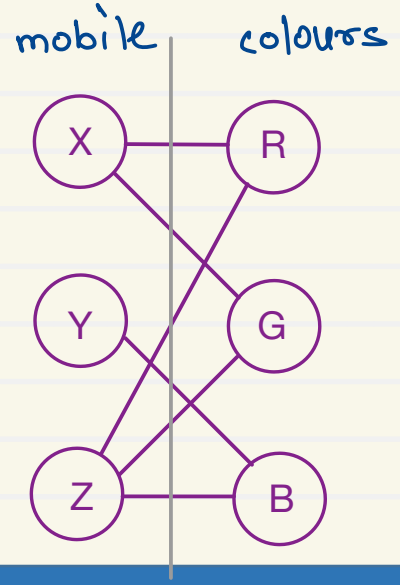




- **Complete graph**
  - Each vertex of a graph is adjacent to every other vertex.
  - Un-directed graph : number of edges = n (n - 1) / 2
  - Directed graph : number of edges = n (n - 1)

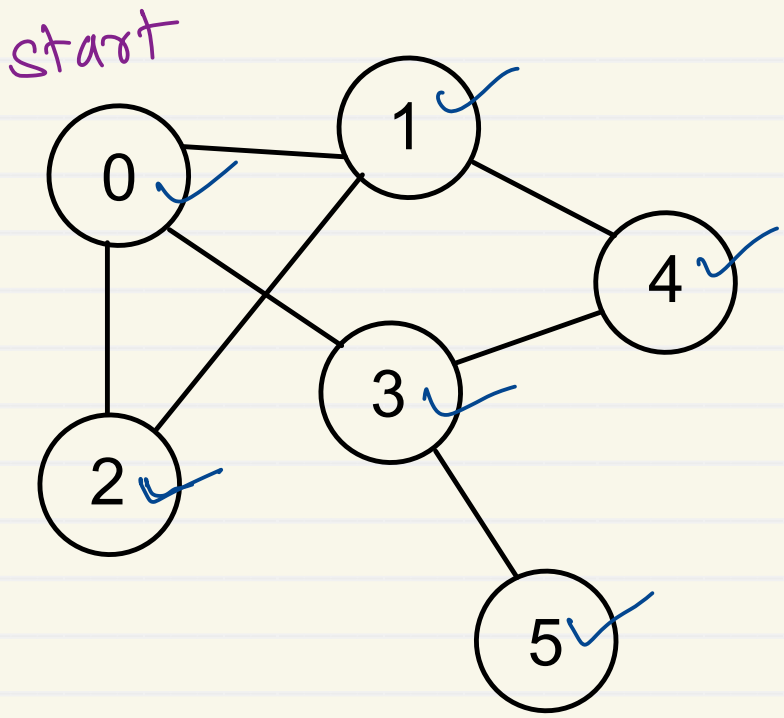
$\frac{5(5-1)}{2}$

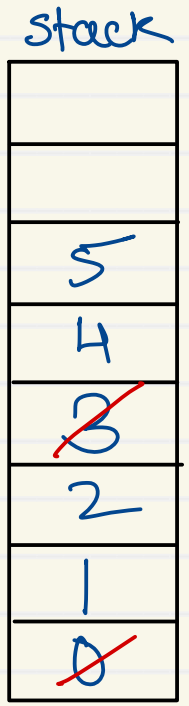




- **Bi-partite graph**
  - Vertices can be divided in two disjoint sets.
  - Vertices in first set are connected to vertices in second set.
  - Vertices in first set are not directly connected to each other.

Check connected ness

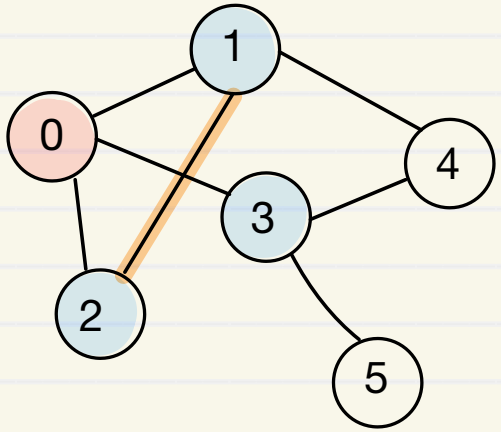




count = 1, 2, 3, 4, 5, 6

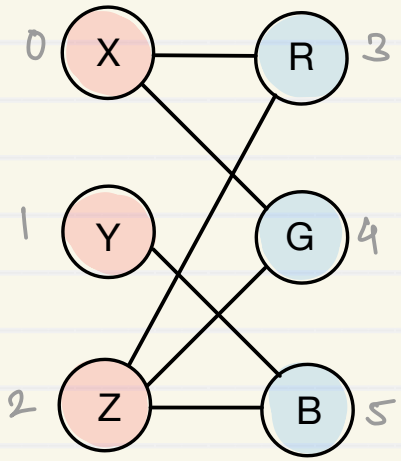
1. Push starting vertex on stack and mark it.
2. Begin counting marked vertices from 1.
3. Pop a vertex from stack.
4. Push all it's non marked neighbours on the stack, mark them and increment count.
5. If count is same as number of vertices, graph is connected (return).
6. Repeat steps 3 - 5 until stack is empty.
7. Graph is not connected (return)

Check Bipartite ness



Queue

0	1	2	3		



Queue

X	R	G	Z	B	Y

1. Keep colours of all vertices in an array. Initially vertices have no colours.
2. Push start on queue and mark it. Assign it colour 1.
3. Pop the vertex.
4. Push all it's non marked neighbours on the queue, mark them.
5. For each such vertex if no colour is assigned yet, assign opposite colour of current vertex.
6. If vertex is already coloured with same of current vertex, graph is not bipartite.
7. Repeat steps 3-6 until queue is empty.

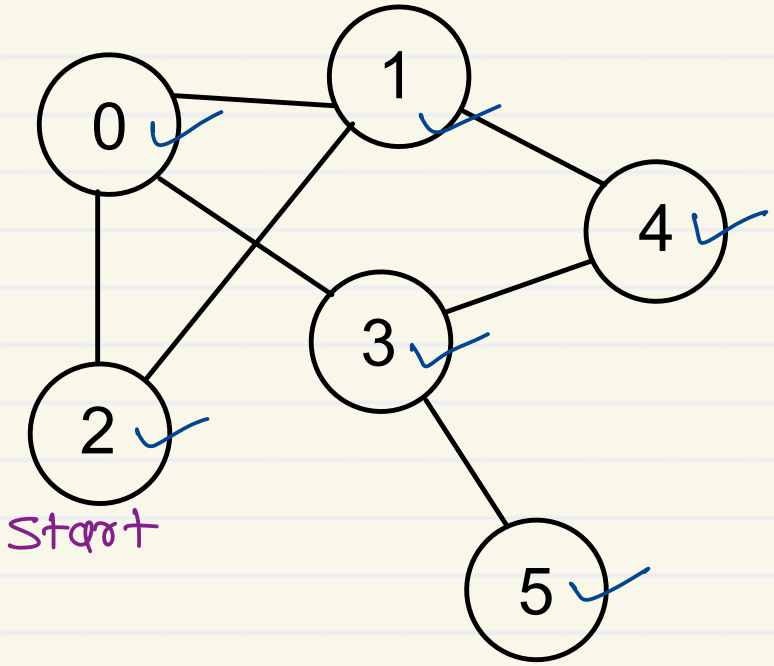
c1
c2

colour1 = +1
No colour = 0
colour2 = -1

+1 * -1 = -1

-1 * -1 = +1

Single source path length



1. Create path length array to keep length of vertex from start vertex.
2. Consider length of start vertex as 0.
3. Push start vertex on queue and mark it.
4. Pop the vertex.
5. Push all it's non marked neighbours on the queue, mark them.
6. For each such vertex calculate it's length as:
   length[neighbour] = length[current] + 1
7. Repeat steps 3 - 6 until queue is empty.
8. Print path length array.

Queue

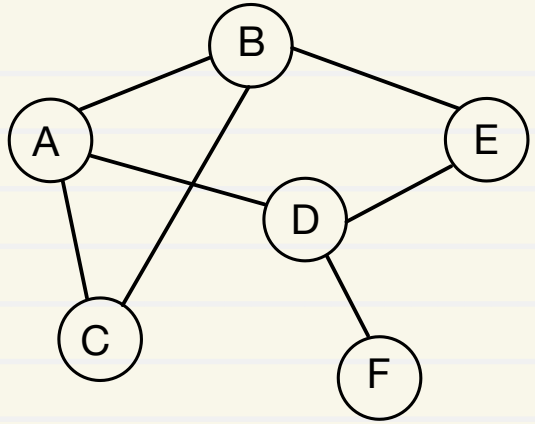
2	0	1	3	4	5	

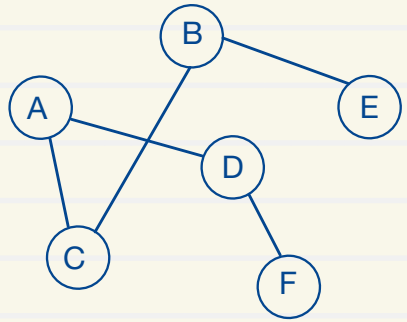
length

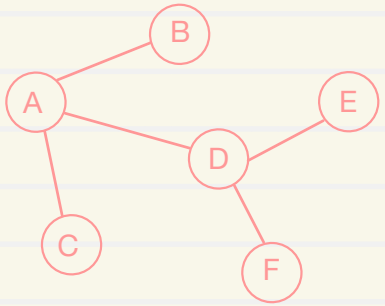
1	1	0	2	2	3
0	1	2	3	4	5

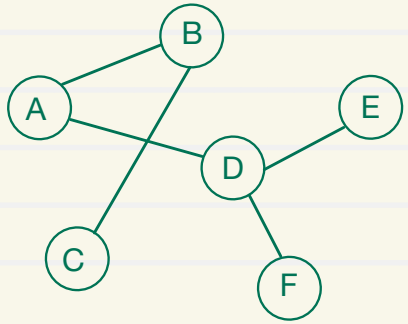
Spanning tree

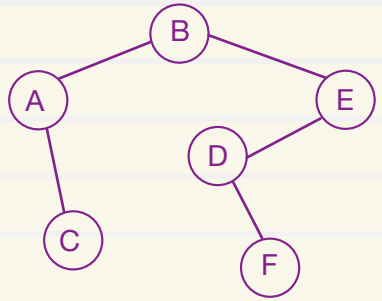
- Tree is a graph without cycles.
- Spanning tree is connected sub graph of the given graph that contains all the vertices and sub set of edges (V-1).
- Spanning tree can be created by removing few edges from the graph which are causing cycles to form.
- One graph can have multiple different spanning 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.
    1. BFS spanning tree
    2. DFS spanning tree
    3. Prim's MST
    4. Kruskal's MST



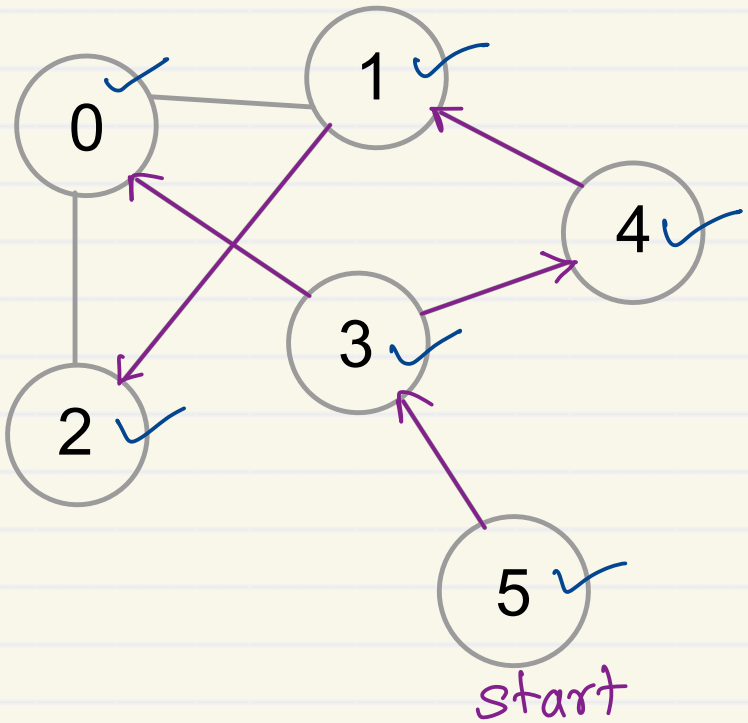


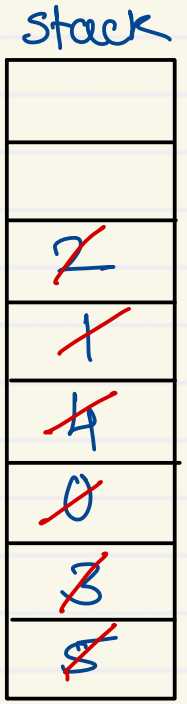






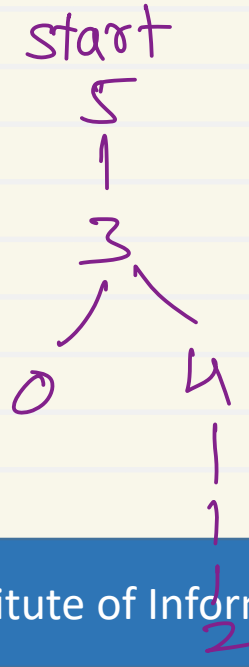
DFS Spanning tree



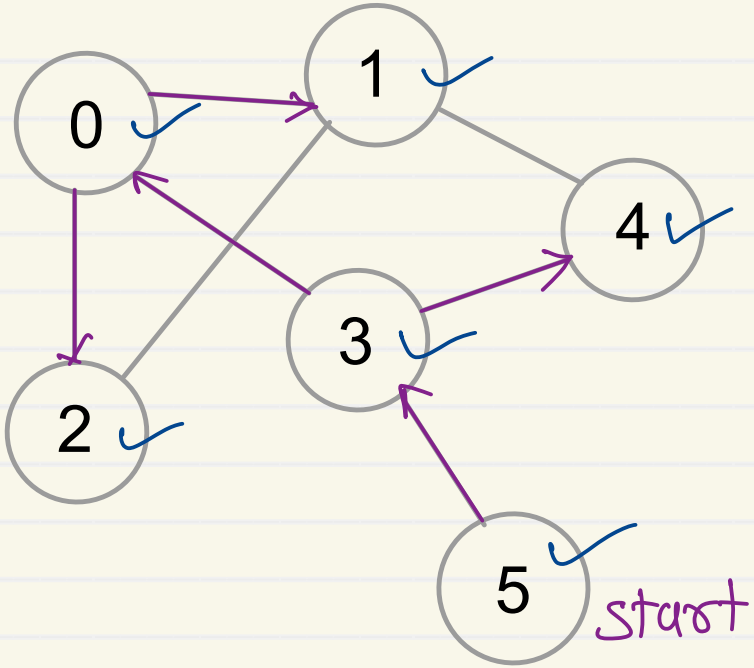


1. Choose a vertex as start vertex.
2. Push start vertex on stack and mark it.
3. Pop vertex from stack.
4. Push all non marked neighbours of the vertex on the stack and mark them. Also print the vertex to neighbouring vertex edge.
5. Repeat step 3 - 4 until stack is empty.

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



BFS Spanning tree

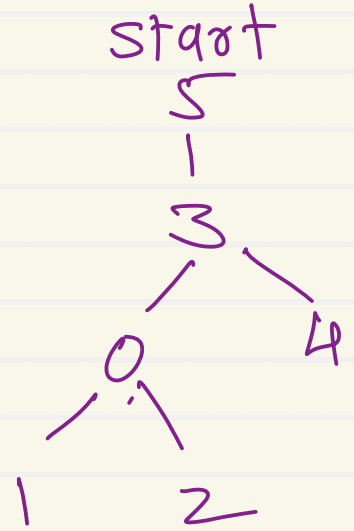


1. Choose a vertex as start vertex.
2. Push start vertex on queue and mark it.
3. Pop vertex from queue.
4. Push all non marked neighbours of the vertex on the queue and mark them. Also print vertex to neighbouring vertex edge.
5. Repeat step 3 - 4 until queue is empty.

Queue

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

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

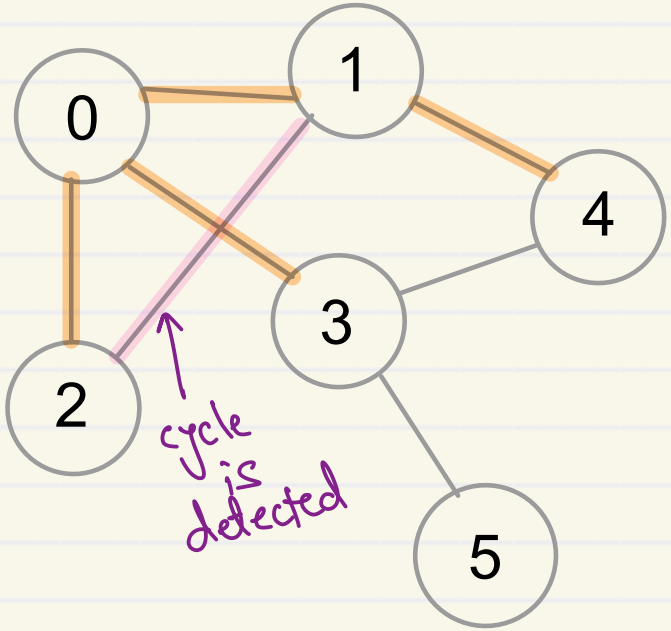


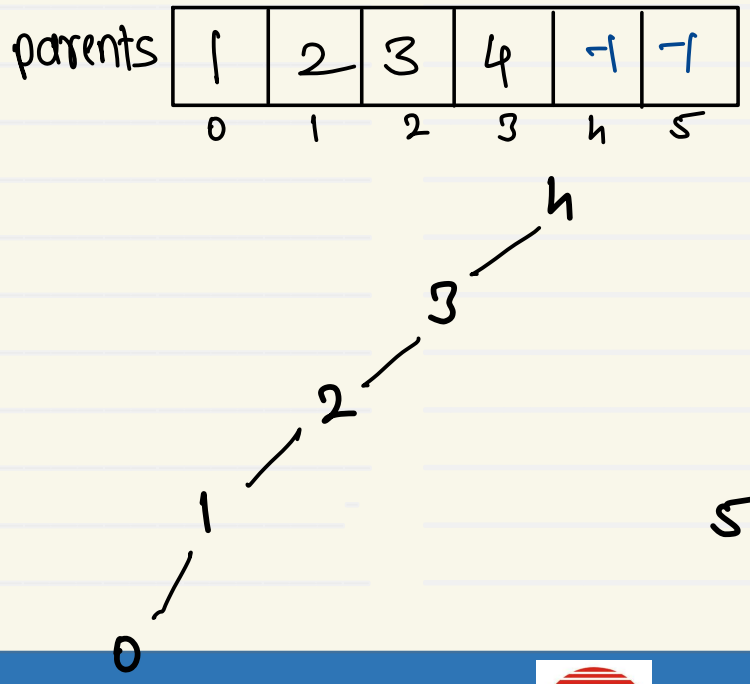
Union Find Algorithm

(used to detect cycle)

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

- skewed tree implementation
  $T(V) = O(V)$
- rank based tree implementation
  $T(V) = O(\log V)$

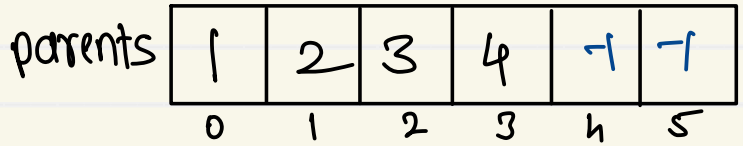




src root	dest root	src	dest
0	1	0	1
1	2	0	2
2	3	0	3
3	4	1	4
4	4	1	2
		3	4
		3	5

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

```
void union(parents, srcRoot, destRoot){
    parents[srcRoot] = destRoot;
}
```



find(parents, 2)

v = ~~2~~, ~~3~~, 4

return 4

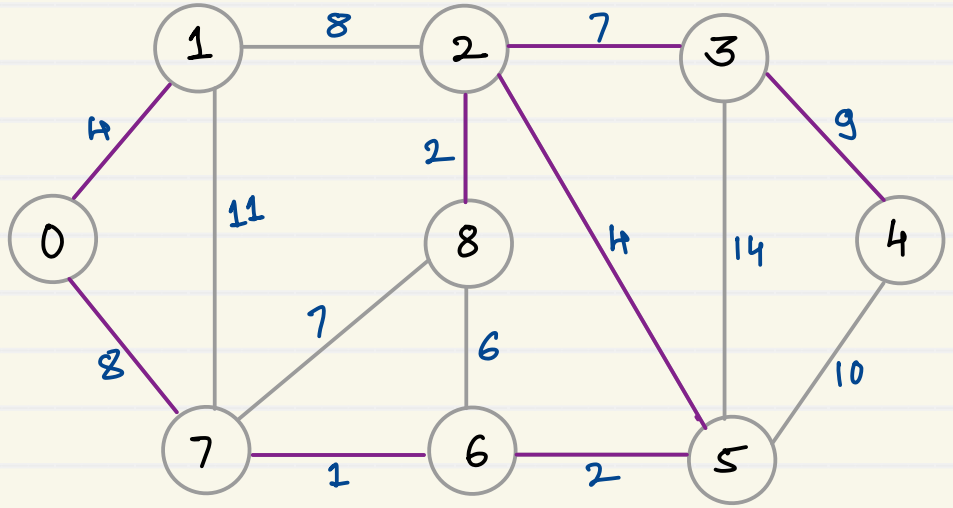
find(parents, 5)

v = 5

return 5

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, otherwise discard it.
3. Repeat step 2 until there are V-1 edges in the spanning tree.



S	d	W	
7	6	1	✓
8	2	2	✓
6	5	2	✓
0	1	4	✓
2	5	4	✓
8	6	6	✗
2	3	7	✓
7	8	7	✗
0	7	8	✓
1	2	8	✗
3	4	9	✓
5	4	10	
1	7	11	
3	5	14	

wt = 1+2+2+4+4+7+8+9

= 37