**Graphs**

Tree represents hierarchical relation and Graph represents Friendship relations. There can be cycles and any node can be connected to any node.

Graph is represented by two sets, -vertices set and Edge set

V={v1,v2,v3}

E={(v1,v2),(,v2,v3),(v1,v3)}

Directed Graph: For an edge (v1,v2) , we can move in only one direction v1->v2. Example: WWW , used for crawlers as well.

Undirected Graph: Edge(v1,v2)==Edge(v2,v1). We can move in both directions. Example social networking if A is friend of B, B is also Friend of A.

Degree: No of edges going through a vertex.

For undirected graph Sum of degree of all vertices =2\* Edges

Max no of edges: If not self-cycle is there

V\*(V-1)/2. Because every edge contributes to two vertices.

For directed graph we have indegree and outdegree. And sum of each is equal to no of edges.

Max no of edges: V\*(V-1)

Path (Walk): A sequence traversal of vertices, where all of them are connected by edges.

Simple path: A path with no repetitions of vertices.

Cyclic Graph: If there exists a walk in the graph which starts and ends on same vertices

Weighted Graphs: Weight is assigned to each edge. Example: Road network where weight is based on distance between locations. Computer networks.

Strongly connected component: Each pair of vertices are reachable from each other.

Graph Representations:

1. Adjacency Matrix: Better for dense graphs

Size= V\*V

A[i][j] =1 if there is an edge from vertex i to j.

For undirected graph, it is a symmetric matrix, i.e., lower triangle and upper triangle are mirror images because if A[i][j]=1 then A[j][i] will also be 1.

For vertices name other than numbers, we will need additional DS map.

Space complexity: Thetha(V\*V)

Check edge for two vertices: Thetha (1)

Find all adjacent of vertex: Thetha (n)

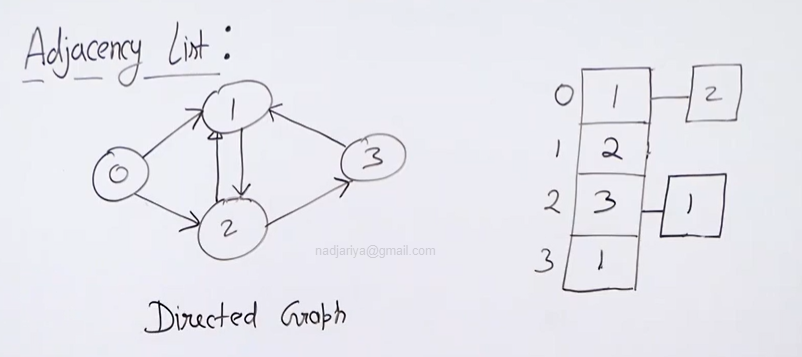
Degree of vertex: Thetha(n);

Add/Remove an edge: Thetha(1)

Add/Remove vertices: Thetha(V\*V) Because we need to copy to new matrix

Disadvantages: If there are less edges lots of redundant information is stored.

1. Adjacency List: Represented by list of lists. Only stores vertices connected to a vertex. Better for sparse graphs



Space: Thetha(V+2E) Undirected

Thetha(V+E) directed

Check edge for two vertices: O(V)

Find all adjacent of vertex: Thetha (Degree of vertex)

Add an edge: Thetha(1)

Remove an edge: O(V)

BFS: Use queue and visited array/set

Time complexity:

O(V+E) if graph is completely disconnected then O(V)

<https://stackoverflow.com/questions/26549140/breadth-first-search-time-complexity-analysis>

Applications of BFS:

* Shortest Path in undirected graph.
* Crawlers in search engine
* Peer to peer network
* Broadcasting
* Social networking search
* Garbage Collection Cheney’s
* Cycle detection
* Ford Fulkerson Algo

DFS: Traverse adjacent and then traverse all vertices reachable through those. Using recursion or stack.

Application of DFS:

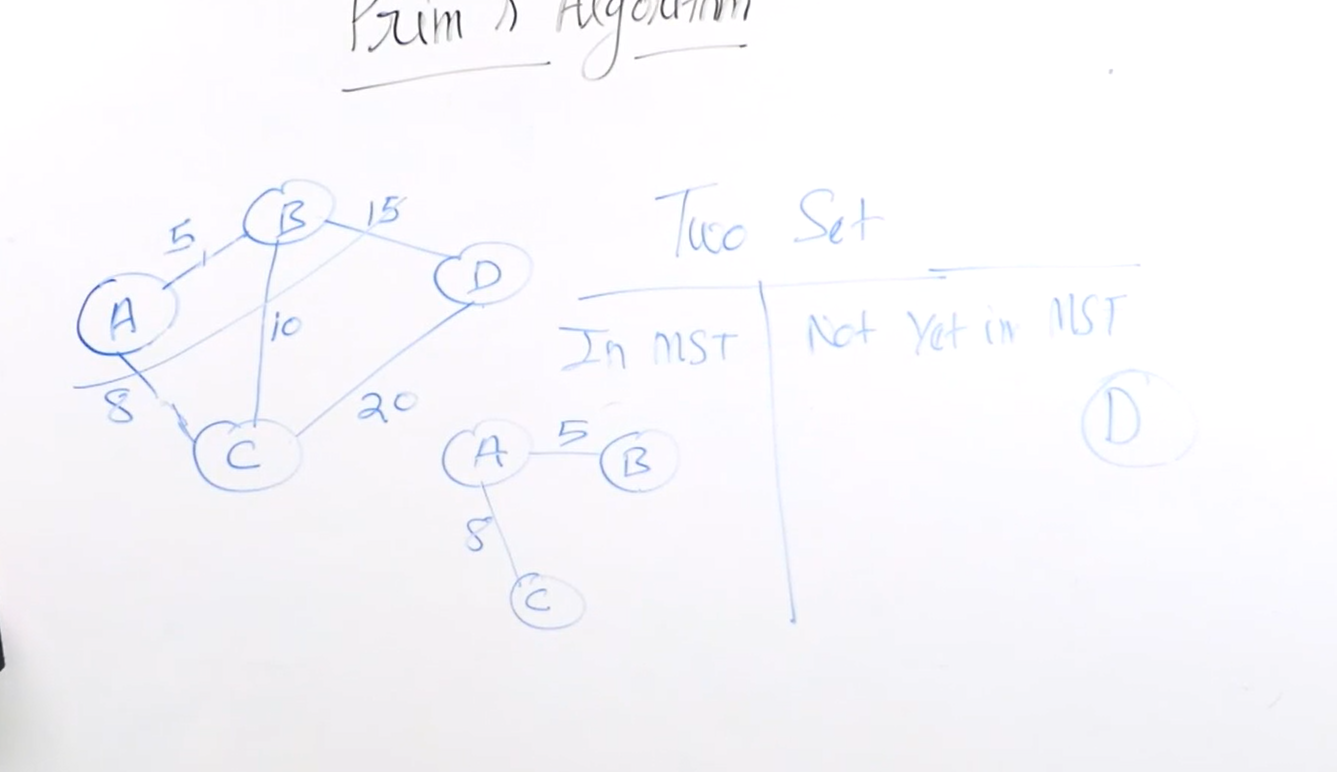
* Cycle detection
* Topological sorting (Job scheduling, a job should execute before jobs that are dependent on it) Works for acyclic graphs
* Strongly connected components
* Solving maze and similar puzzles
* Path finding

Spanning Tree: No Cycles. And all nodes are reachable from each other.

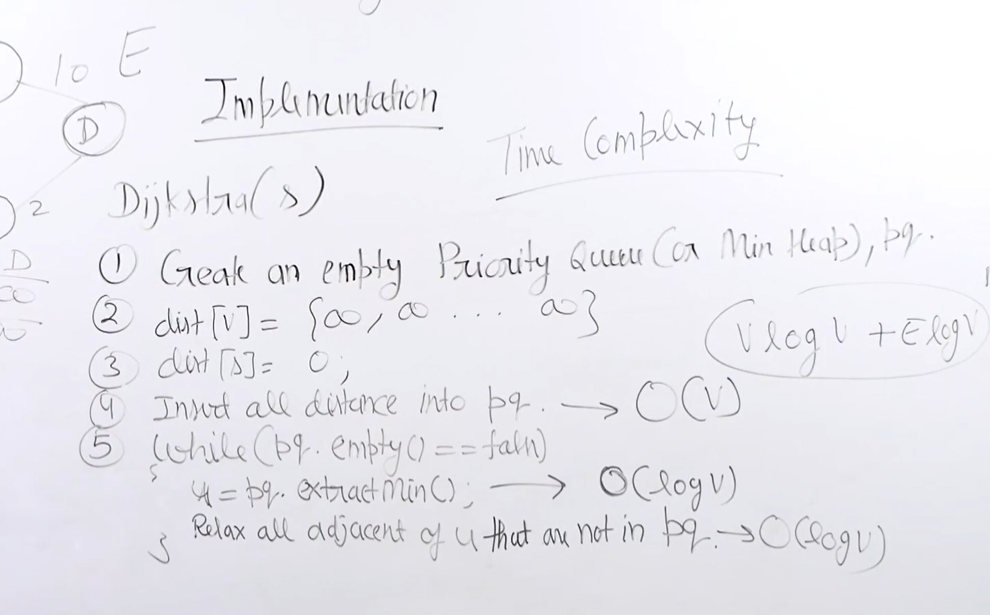
V-1 edges for unweighted graph

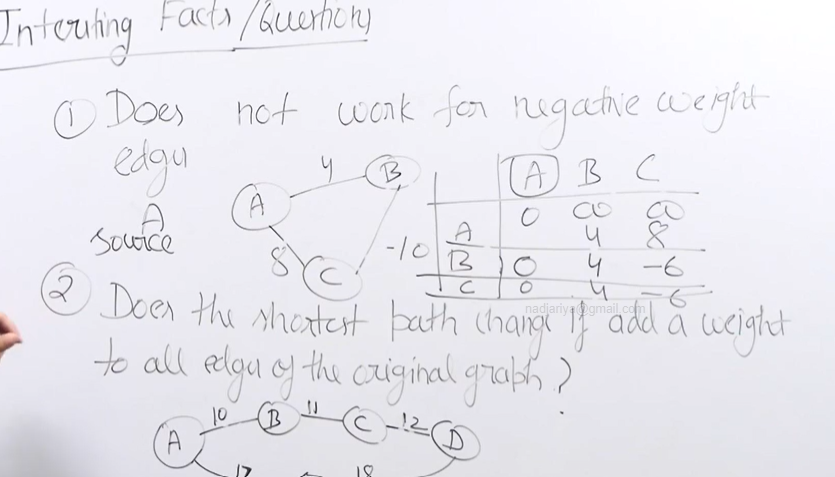
Min Spanning tree: Given a weighted and connected graph find a min weight spanning tree.

Prim’s Algo: Greedy algo to find MST. Maintain two set one which are already included in MST set and rest in other set. Find min cut to connect one of the remaining vertices to MST set.



Dijkstra Algorithm: Finds shortest path between vertices.

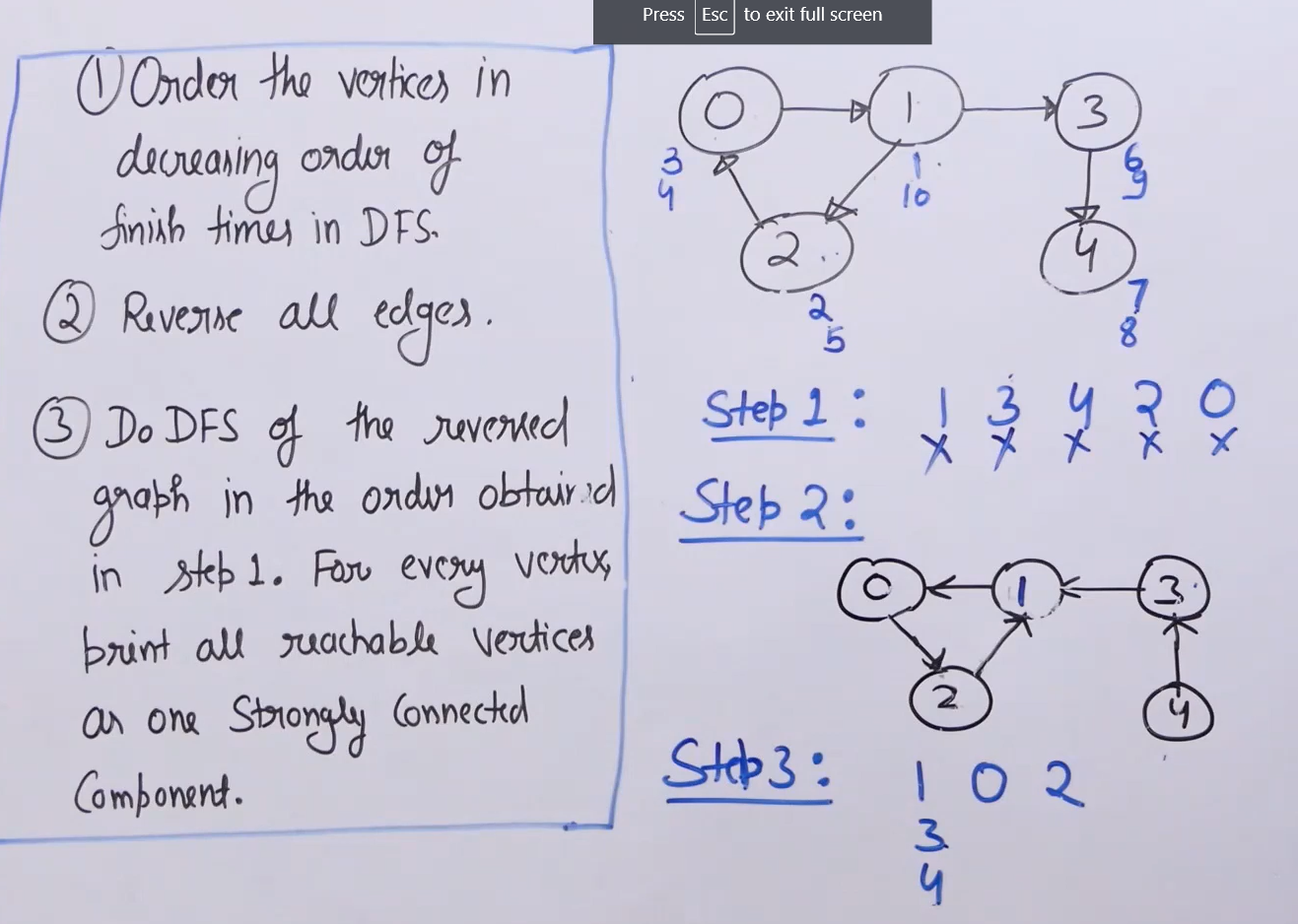




Shortest path changes on addition of number to each weight as some path can have less edges and it might become less.

Multiplying each weight with a number does not change a shortest path

Kosaraju’s Algo for strongly connected component directed graph:



Articulation point: A vertex in a graph, removal of which and its edges increases the number of connected components.

Revisit: Bellmanford, Articulation point, Bridges, Tarjan’s Algo,Krushkal’s Algo