* Graph:

Graph is a non-linear data structure in which Graph G is composed of a finite set of vertices and edges.

Represent as: G = (V, E)

⇒ Graph Representation:

There are two type of Graph Representation.

1) Adjancency Matrix Representation.
2) Adjacency Linked List Representation.

1 Adjacency Matrix Representation:

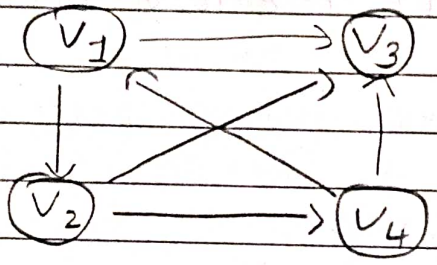
In this matrix representation, we have to use 2 dimensional array.

This representation is also called Sequential representation.

If edge from
V_i to V_j then $A[i, j] = 1$

If edges are not
from V_i to V_j then $A[i, j] = 0$

Ex.



	V_1	V_2	V_3	V_4
V_1	0	1	1	0
V_2	0	0	1	1
V_3	1	0	0	0
V_4	1	0	1	0

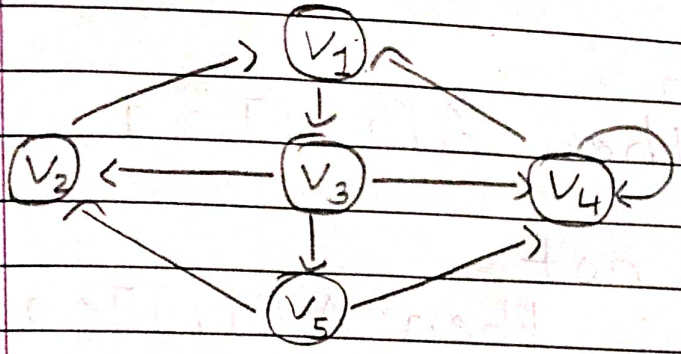
2 Adjacency Linked list Representation

In this representation, vertices are represent by ~~three~~ Two part,

1) Data
2) 2 link part.

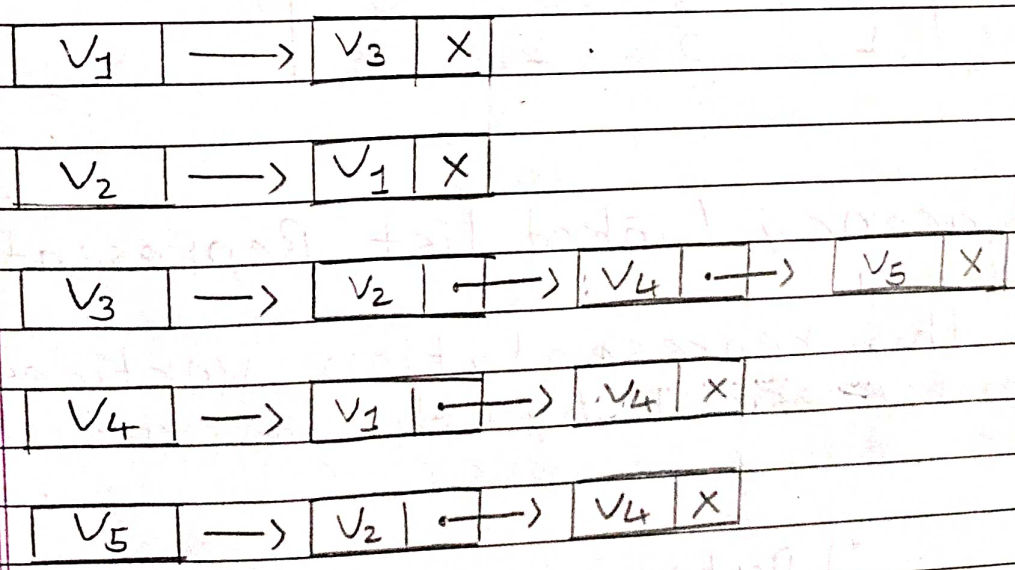
Page No.
Date : / /

Ex.



Vertex with adjacency list,

$V_1 \rightarrow V_3$
$V_2 \rightarrow V_1$
$V_3 \rightarrow V_2, V_4, V_5$
$V_4 \rightarrow V_1, V_4$
$V_5 \rightarrow V_2, V_4$



SMVS
Page No.
Date : / /

=> Basic Terminology of Graph:

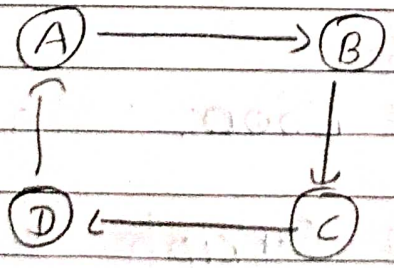
1 Directed Graph: A Graph is called directed Graph, if every ~~ve~~ edges of graph has direction.

2 Undirected Graph: A Graph is called undirected Graph, if edges does not have any direction.

3 Null Graph: A Graph is called null graph, if graph does not contain any edges.

4 Path: A path in a graph is a finite sequence of edges which joins a sequence of vertices.

Ex.

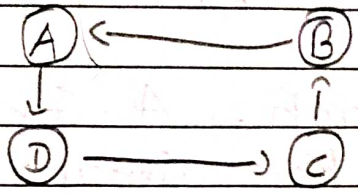


Path → A - B - C - D

SMVS
Page No.
Date : / /

5 Cycle: A cycle is a path in which first and last vertices are same.

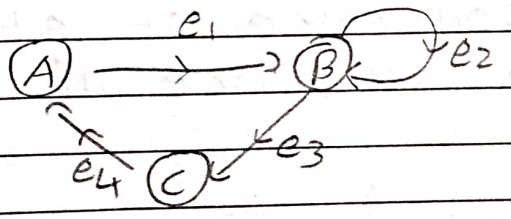
Ex.



Cycle → A - D - C - B - A

6 Loop: An edge will called Loop, if edges starting and ending point are same.

Ex.



e_2 is called Loop.

7 Cyclic Graph: Graph is called Cyclic Graph, if graph contain cycle.

8 Acyclic Graph: Graph does not contain cycle.

SMVS
Page No.
Date : / /

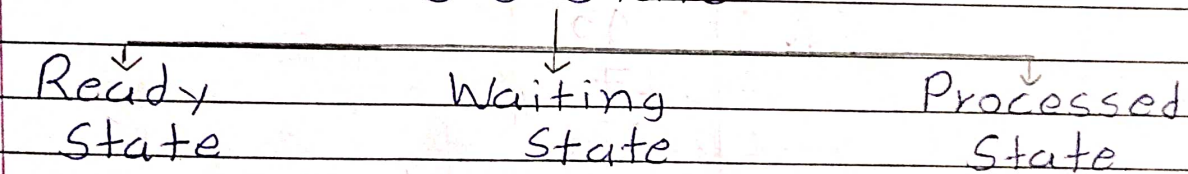
* BFS (Breath First Search):

Breath First Search method is use for searching in a graph.

For BFS searching, we have to use Queue for tracking graph.

In BFS, For tracking the graph we have to start from Root vertex.

In BFS there are three state.

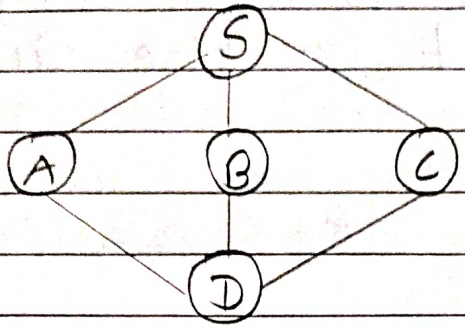


-> Ready State: Element is not in Queue and Processed not start.

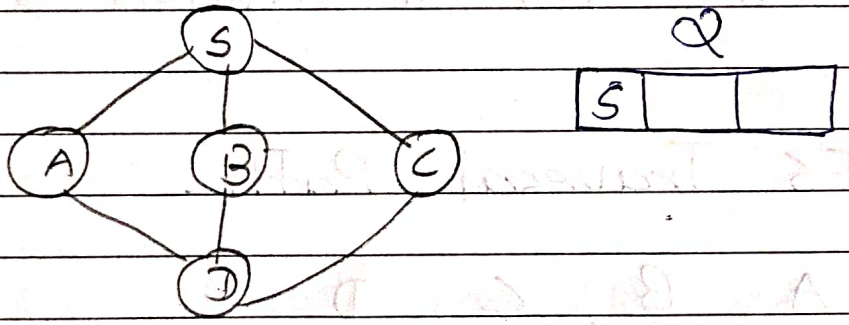
-> Waiting State: Element is in Queue, but Processed not start.

-> Processed State: Processed is start and element is out in the Queue.

Ex. Track the Graph:

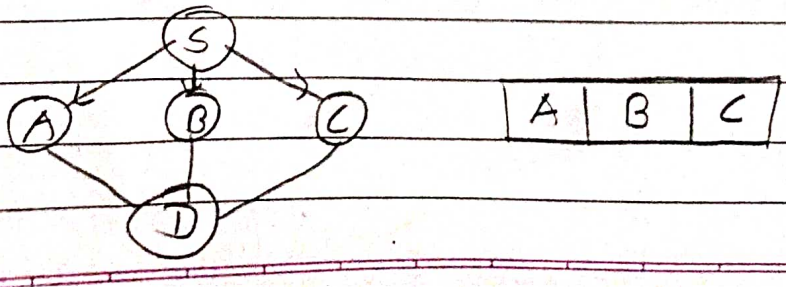


- Step-1 - In BFS, First we have to select Root vertex and insert root element in the Queue.

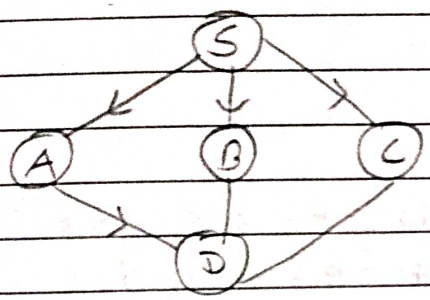


- Step-2 - After the root element we have to select adjcest nodes in the graph.

In this graph A, B and C are adjsent node and insert into Queue and delete root element



- Step - 3 - After that select First element and insert adjsent node element in the Queue



B	C	D

- Step - 4 - After the D element there are no a unwisited node.

So, BFS Travesal Path:

S, A, B, C, D

- Algorithm:

1 Initialize all nodes to ready state.

2 Insert the starting node in queue and change its state to waiting state.

3 ~~dete~~ Repeat the 4 and 5 step untill Queue is empty.

4 Delete the front node n of queue. Process and change to state to Processed state.

5 Insert n in the queue, all the adjesent node are in ready state and change this state to waiting state.

6 Exit.

* DFS:

Depth First Search method is use for searching in a graph.

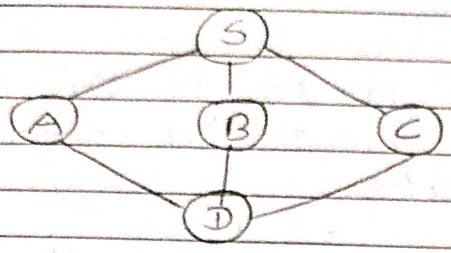
For DFS, we have to use stack for tracking the path.

In DFS we have to start from the root node.

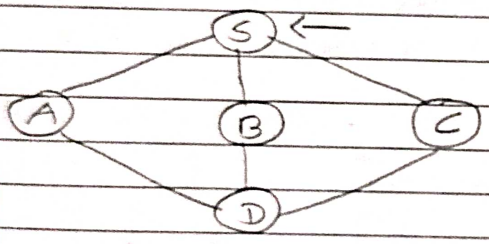
First we have to insert root element in root.

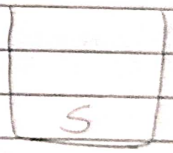
SMVS
Page No.
Date : / /

Ex. Track the Grap :



- Step-1 - In DFS, First we have to select root element and push root element in the stack.

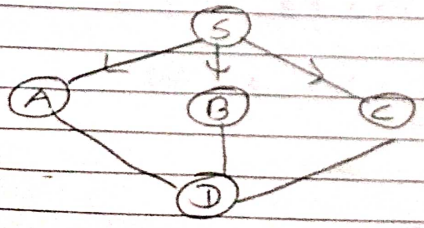




push S

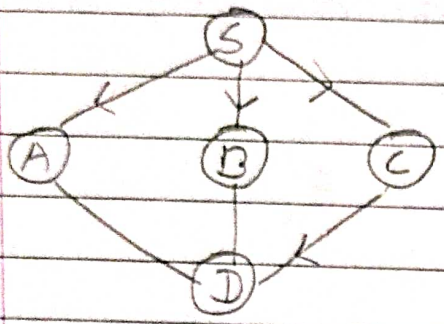
- Step-2 - After the root element, we have to select root element adjesent node.

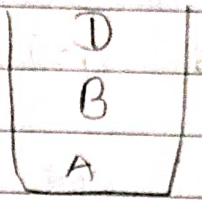
and pop the A and push A, B and C



SMVS
Page No.
Date : / /

- step - 3 - After that take C element and push adjesent element of the C. and pop C.





- step - 4 - After that pop D, pop B and pop A.

So, DFS Travesal Path.

S C D B A

Page No.
Date : / /

* Minimum Spanning Tree:

=> Spanning Tree:

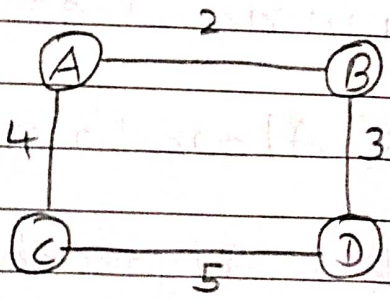
A spanning tree of a connected graph G contain all the nodes and has the edges which connects all the nodes.

In a spanning tree, there are no cycle in the graph.

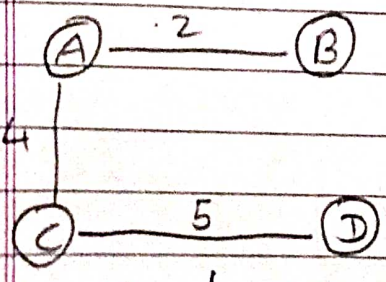
=> Minimum Spanning Tree:

A minimum spanning tree is founed when the edges are picked to minimize the total cost.

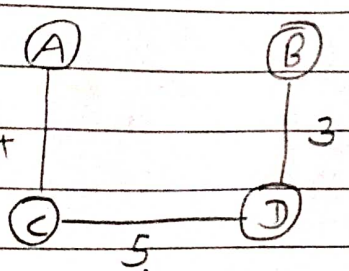
Ex.



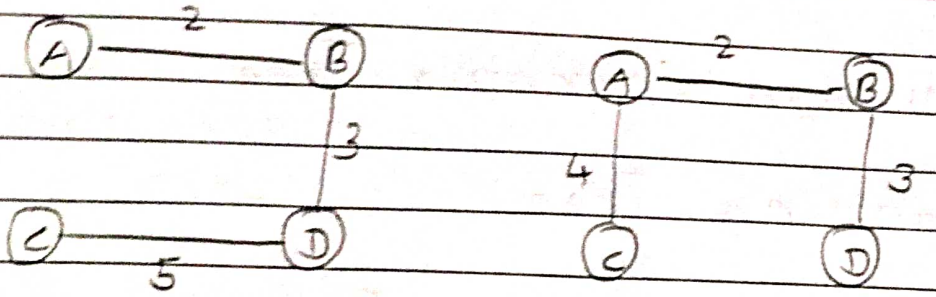
-> Spanning Tree:



Cost = 11



Cost = 12



Cost = 10 Cost = 9

- This four are possible spanning tree.
- Cost = 9 is a minimum spanning tree.

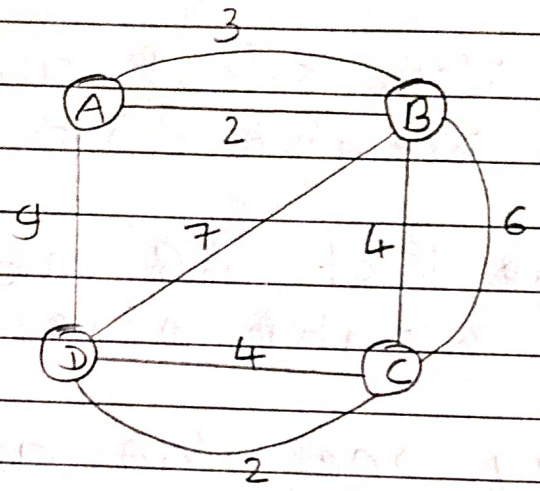
* 2 Method of Finding Minimum Spanning Tree.

There are two method of Finding Minimum Spanning Tree.

1) Prim's Algorithm
2) Kruskal's Algorithm.

1 Prim's Algorithm :

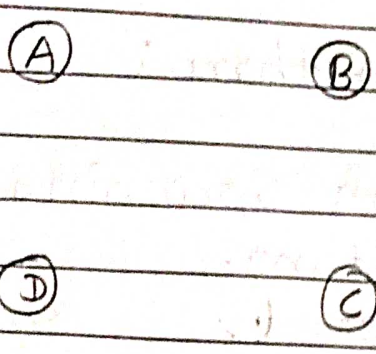
Ex. Find Minimum Spanning Tree using Prim's Algorithm:



- Step - 1 - Create edge list of this given graph.

Edge	Weight
AB	2
AB	3
AD	9
BC	4
BC	6
BD	7
CD	4
CD	2

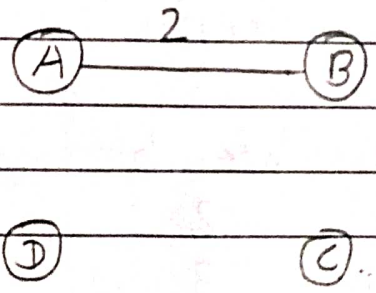
- Step - 2 - Create skeleton for Spanning tree.



- Step - 3 - Now select any edge with minimum weight from edge list.

In the edge list, AB and CD are minimum weight edge.

Select any one edge and delete from the list.

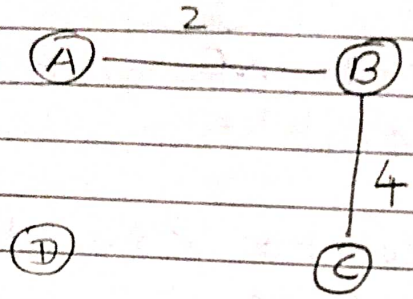


- Step-4 - After that select one end point from AB edge.

This are the edges: AB(3), AD(9), BC(6), BD(7), BC(4).

After that select minimum weight list.

In this list, BC edge is minimum weight edge.



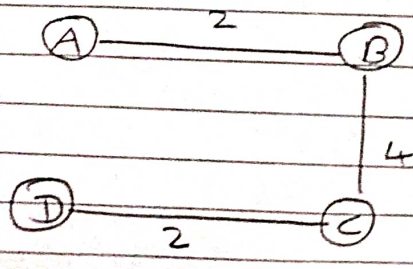
- step - 5 - After that select any one end from A, B and C.

This are the edges: AB(3), AD(9), BC(6), BD(7), CD(4), CD(2).

AB(3) and BC(6) are cycle. So, this edges are remove from list.

After that select minimum weight list.

In this list, CD edge is minimum weight edge in the list.



- Algorithm:

1 Start

2 Create edge list of given graph with their weights.

3 Draw all nodes to create skeleton for spanning tree.

4 Select edge with lowest weight and add it to skeleton and delete from edge list.

5 Select one end from edge list, while adding edge take the one end from skeleton tree and its cost should be minimum.

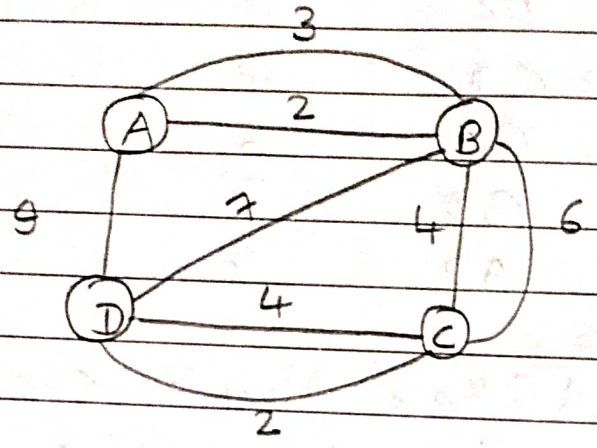
6 Repeat step-5 until n-1 edges are added.

7 End.

Date : / /

2 Kruskal's Algorithm:

Ex. Find the minimum spanning tree using Kruskal's Algorithm.



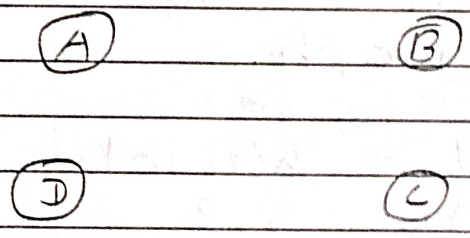
- Step-1- Create edge list of this given graph.

Edge	Weight
AB	2
AB	3
AD	9
BC	4
BC	6
BD	7
CD	4
CD	2

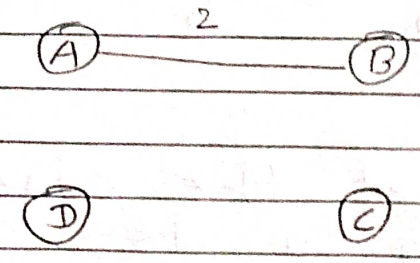
- Step-2 - Short the edges list according to its weight.

Edge	Weight
AB	2
CD	2
AB	3
BC	4
CD	4
BC	6
BD	7
AD	9

- step- 3 - After that, create skeleton for spanning list.

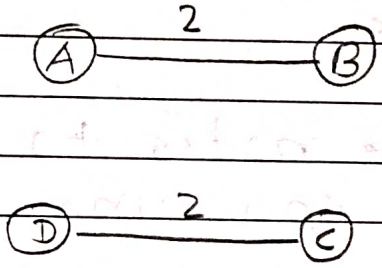


- step- 4 - After that, Now select first entry from the edge list and delete from the edge list.



- step-5 : Now select second entry from the list, but this entry does not create a cycle.

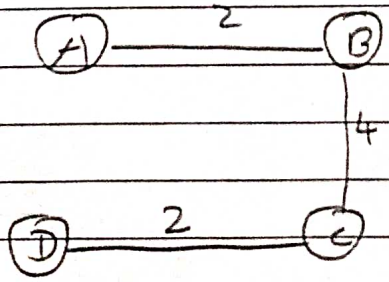
In this list, CD = 2 is a minimum edge in the list and remove from the list.



- step-6 - After that select next edge from the list. Here AB is create cycle.

So, we have to select next edge from the list.

In this list, BC = 4 is a minimum edge in the list.



- Algorithm:

1 Start

2 Create the edge list of given graph with its weight.

3 Sort the edge list according to their weights.

4 Draw all the nodes to create skeleton for spanning tree.

5 Pick up the edge at the top of the edge list and remove from the list.

6 Connect vertices in the skeleton until edges should be n-1.

7 Repeat this step 5 and 6.

8 End.