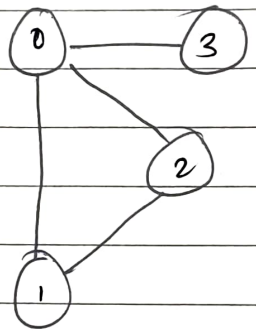
Graphs

* A Graph is a data structure that has collection of nodes containing data and these are connected to other nodes

Precisely, a graph 'G' is a D.S. (V, E) that consist of -

- set of Vertices 'V' / nodes
- set of Edges 'E', represented as ordered pairs of vertices (u, v).

Ex..



Vertices and Edges.

- Adjacency - A vertex is said to be adjacent to another vertex if there is an edge connecting them. In above graph, 2 & 3 are not adjacent.

- Path : A sequence of edges that allows to go from vertex 'A' to vertex 'B' is called a Path. Ex: 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2

- Directed Graph A graph in which edge (u, v) doesn't necessarily mean that there is ~~a~~ an edge (v, u) as well. The edges in such a graph are represented by arrows.

Graph Representation in Memory

① Adjacency Matrix

An adjacency matrix is a 2D-Array of V x V vertices. Each row and column represent a vertex.

a[i][j] = 1 represents that there is an edge connecting vertex 'i' and 'j'.

Ex:

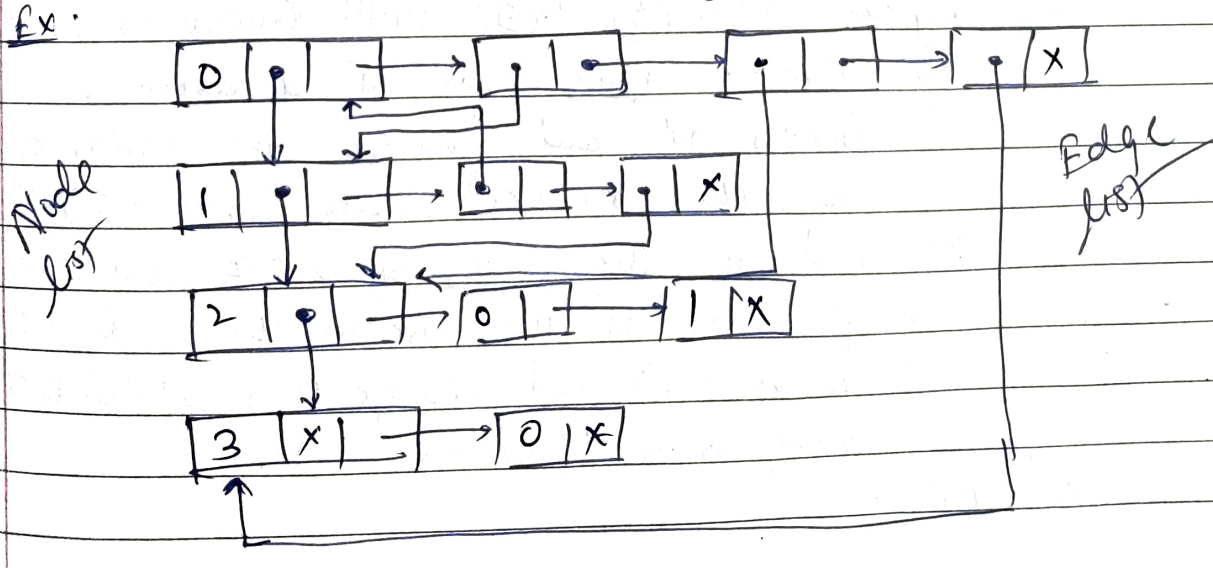
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

② Adjacency list

An adjacency list represents a graph as an array of linked list.

Index of array represents a vertex and each element in its linked list represent the other vertices that form an edge with the vertex.

Ex:



* Traversing a Graph

For traversal algorithms each node 'N' in 'G' is considered to be in one of the three states.

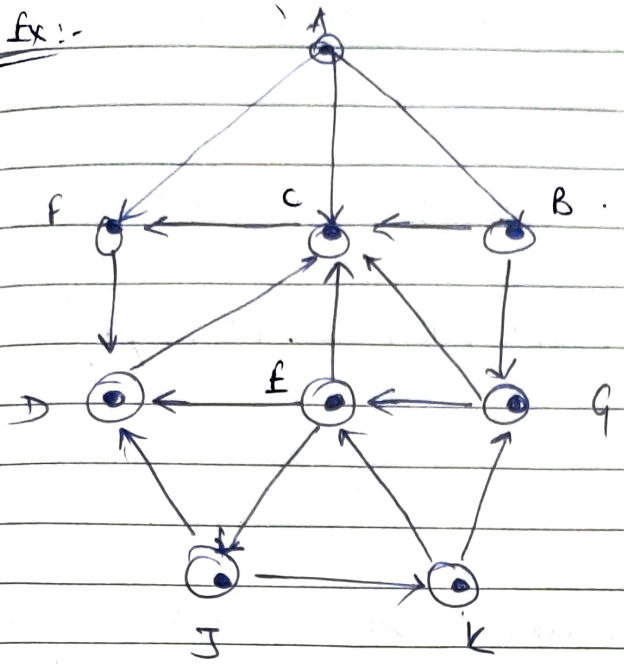
STATUS = 1 ⇒ (Ready State) Initial state of Node 'N'.
STATUS = 2 ⇒ (Waiting State) Node 'N' is on queue or stack, waiting to be processed.
STATUS = 3 ⇒ (Processed State) Node 'N' has been processed.

* Breadth First Search (BFS).

This algorithm executes BFS on 'G' beginning at a starting node 'A'.

1. Initialize all nodes to the ready state (STATUS = 1)
2. Put the starting node 'A' in QUEUE and change its state to the waiting state (STATUS = 2)
3. Repeat steps 4 & 5 until QUEUE is empty.
   4. Remove front node 'N' of QUEUE. Process 'N' and change the status of N to the processed state (STATUS = 3)
   5. Add to the rear of QUEUE all the neighbors of 'N' that are in the ready state. (STATUS = 1) and change their status to the waiting state (STATUS = 2).
6. Exit.

Ex :-



Adjacency List

A : F, C, B.
B : G, C
C : F
D : C
E : D, C, J.
F : D
G : C, E
J : D, K.
K : E, G.

(1) Add 'A' to QUEUE and SEQ = NULL

QUEUE = A
SEQ = NULL

States A B C D E F G J K
2 1 1 1 1 1 1 1

(2) QUEUE = ~~A~~, F, C, B
SEQ = A

States A B C D E F G J K
3 2 2 1 1 2 1 1 1

(3) QUEUE = ~~A~~, F, C, B, D
SEQ = A, F
F removed and D added to QUEUE.

States A B C D E F G J K
3 2 2 2 1 3 1 1 1

(4) QUEUE = ~~A~~, ~~F~~, C, B, D
SEQ = A, F, C

States A B C D E F G J K
3 2 3 2 1 3 1 1 1

(5) Q : ~~A~~, F, ~~C~~, B, D, G
SEQ : A, F, C, B

States A B C D E F G J K
3 3 3 2 1 3 2 1 1

(6) Q : ~~A~~, ~~F~~, ~~C~~, B, ~~D~~, G,
S : A, F, C, B, D

States A B C D E F G J K
3 3 3 3 1 3 2 1 1

(7) Q : ~~A~~, ~~F~~, ~~G~~, ~~B~~, ~~D~~, ~~G~~, E
S : A, F, C, B, D, G

States - A B C D E F G J K
3 3 3 3 2 3 3 1 1

(8) Q : ~~A~~, ~~F~~, ~~C~~, ~~B~~, ~~D~~, ~~G~~, ~~E~~, J
S : A, F, C, B, D, G, E

S :- A B C D E F G J K
3 3 3 3 3 3 3 2 1

(9) Q: A, F, C, B, D, G, E, J, K Status A B C D E F G J K
S: A, F, C, B, D, G, E, J 3 3 3 3 3 3 3 3 2

(10) Q: Empty
S: A, F, C, B, D, G, E, J, K Status 3 3 3 3 3 3 3 3 3

BFS - A, F, C, B, D, G, E, J, K

* **DFS:-**

(1) Initialize all nodes to ready state [STATUS = 1]
(2) Push starting node 'A' onto STACK and change its state to waiting state [STATUS = 2]
(3) Repeat (4) & (5) while STACK is not empty
(4) Pop top element 'N' from STACK, change its state to "Processed state" [STATUS = 3]
(5) Push on STACK all neighbors of 'N' that are in ready state [STATUS = 1] and change their state to waiting state [STATUS = 2].
(6) Exit

For above example

DFS:- A, B, G, E, J, K, D, C, F

* Spanning tree

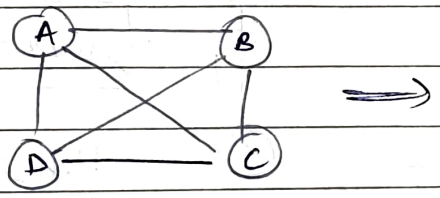
A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with minimum possible no. of edges.

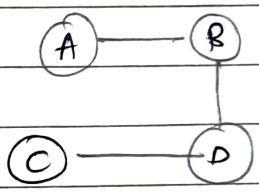
If a vertex is missed then it is not a spanning tree

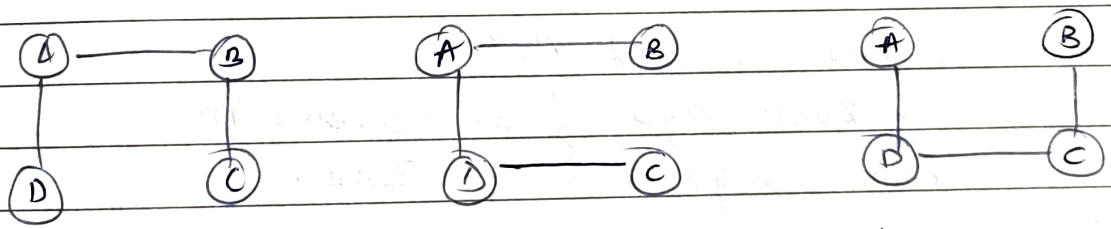
Complete Graph containing 'n' vertices may have n^{n-2} no. of spanning trees.

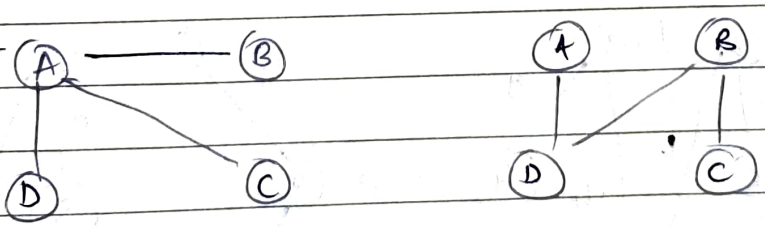
Ex :- $n=4$ max. no. of spanning trees possible is equal to $4^{4-2} = 16$. from a complete graph

Ex. of Spanning tree





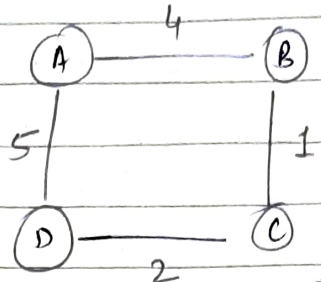




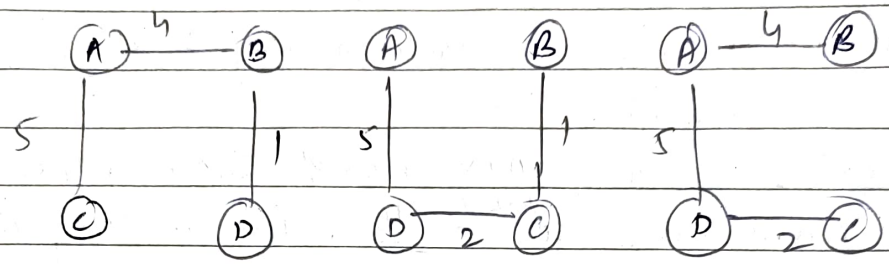
Minimum Spanning tree (MST)

A MST is a spanning tree in which the sum of the weight of edges is as minimum as possible

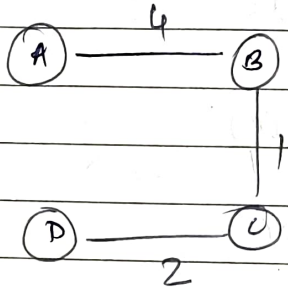
Example :-



Possible spanning trees from above graphs are -



sum = 10 sum = 8 sum = 11



sum = 7 - minimum

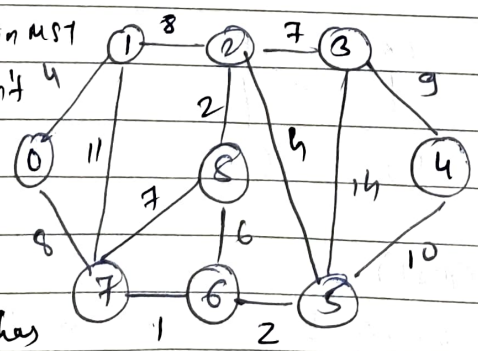
Hence, MST.

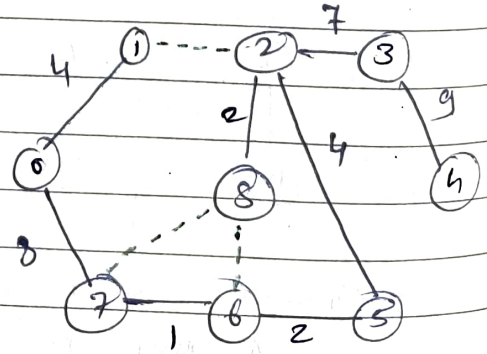
Algorithms to find MST

1. Kruskal's algo
2. Prim's algo

⇒ covered in DAA.

1) Sort edges.
2) Include in MST if it doesn't form a cycle.
3) Repeat till MST has "n-1" edges.





MST.

Given Edge ⇒ Not included as they form cycle