Unit - V

Graph

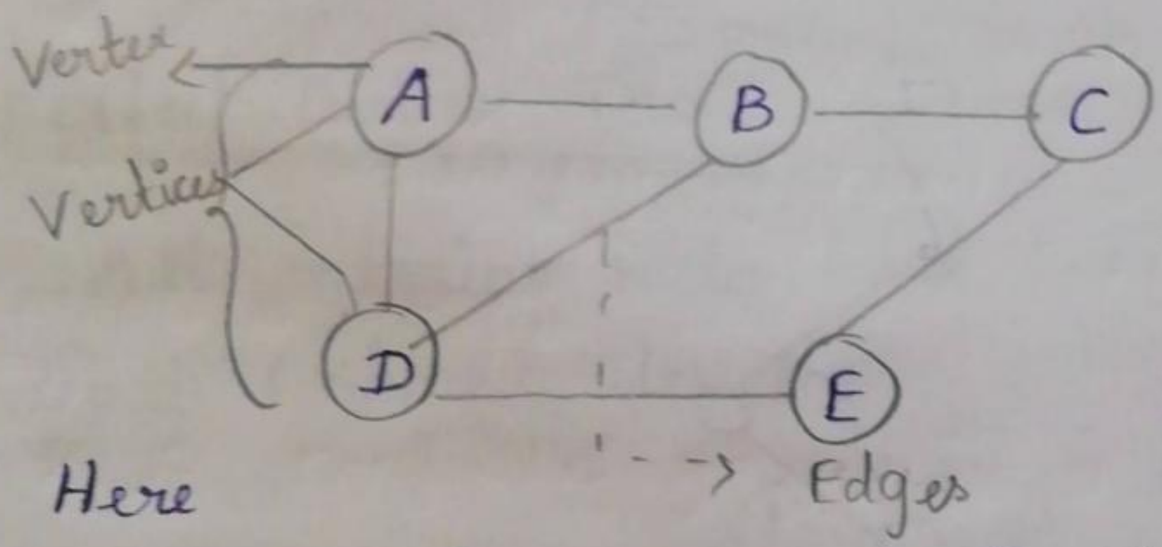
* graph is a non linear data structure which is a collection of vertices and edges.
* Graphs are used to model any situation where entities or things are related to each other in pairs

Example : Family tree
Transportation networks

Definition:

A graph G is defined as an ordered Set (V, E), where V represents the set of vertices and E represents the edges that connect these Verties

Example



Here

V(G) = { A, B, C, D, & E } Vertex

E(G) = { (A, B), (B, C), (A, D), (B, D), (D, E), (C, E) } Edges

* A Graph can be directed or undirected

* In an Undirected graph, edges do not have any direction associated with them

Representation of Graph

matrix / list

1) Adjaceny Matrix Representation :

* An Adjaceny matrix is used to represent which nodes are adjacent to one another two nodes are said to be adjacent if there is an edge connecting them.

* In a directed graph G, if node v is adjacent to node u then there is definitely an edge from u to v. That is, if v is adjacent to u, we can get from u to v by.

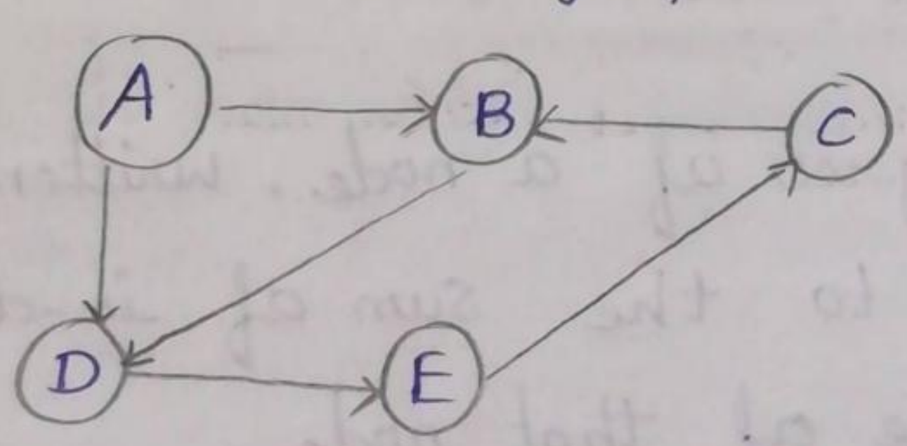
* For any graph G having n nodes the adjacendy matrix will have the dimensions of n * n.

* In an adjacendy matrix, the rows & column are labelled by graph vertices

a_{ij} / 1[if v_i is adjacent to v_i]
\ 0[otherwise]

Example :

Directed graph.



Self loop not accepted.

	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	0
C	0	1	0	0	0
D	0	0	0	0	1
E	0	0	1	0	0

2) Adjacency List Representation

List is a best

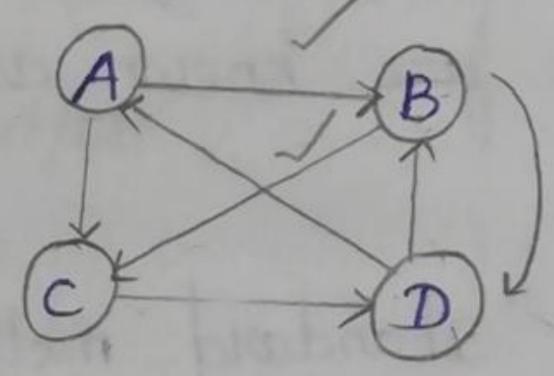
In a adjacency list representation of graph for each Vertex, keep a list of all adjacent Vertices.

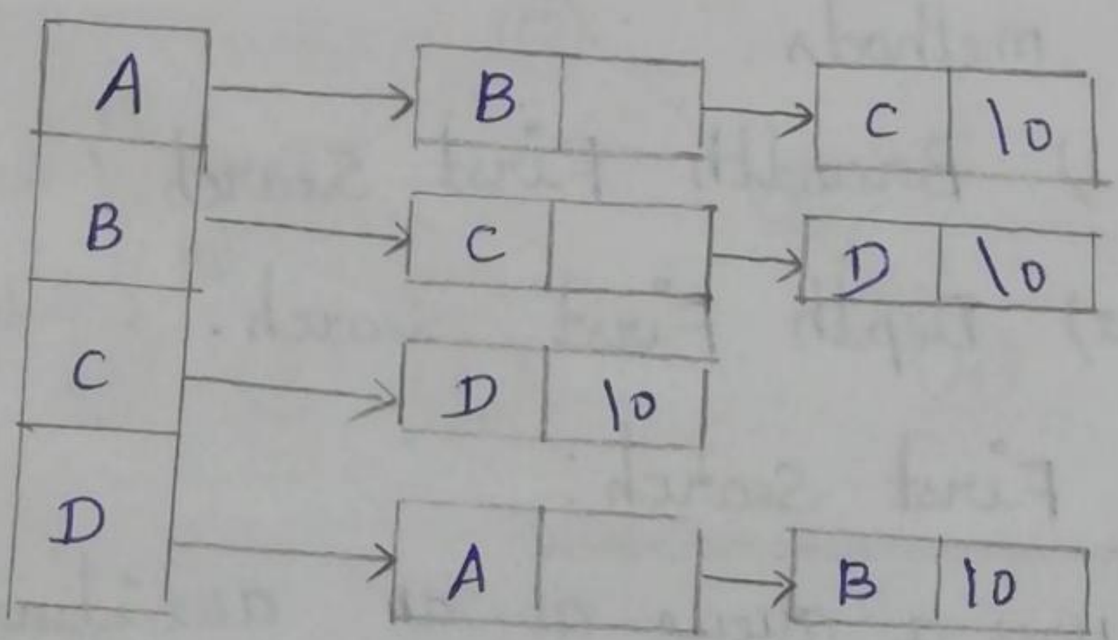
Always list is the best

No memory Wastage.

Easy to Implement.

Example:





Advantages:

* It is easy to follow and clearly shows the adjacent nodes of a particular node

* It is used for storing graph that have a small moderate no. of. edges.

* Adding new nodes in G is easy and straight forward when a graph is represented using an adjacency list.

* Adding new nodes in adjacency matrix is a different task, as the size of the matrix needs to be changed and existing nodes may have to be recorded.

Graph Traversal

DFS

Algorithms :

* The method of examining the nodes and edges of the graph are known as traversing a graph.

* There are two standard methods of graph traversal methods.

1) Breadth First Search. (BFS) (Queue - FIFO)

2) Depth First Search. (DFS) (Stack - LIFO)

Breadth First Search :

It uses a queue as an auxiliary data structure to store nodes for futher processing Depth.

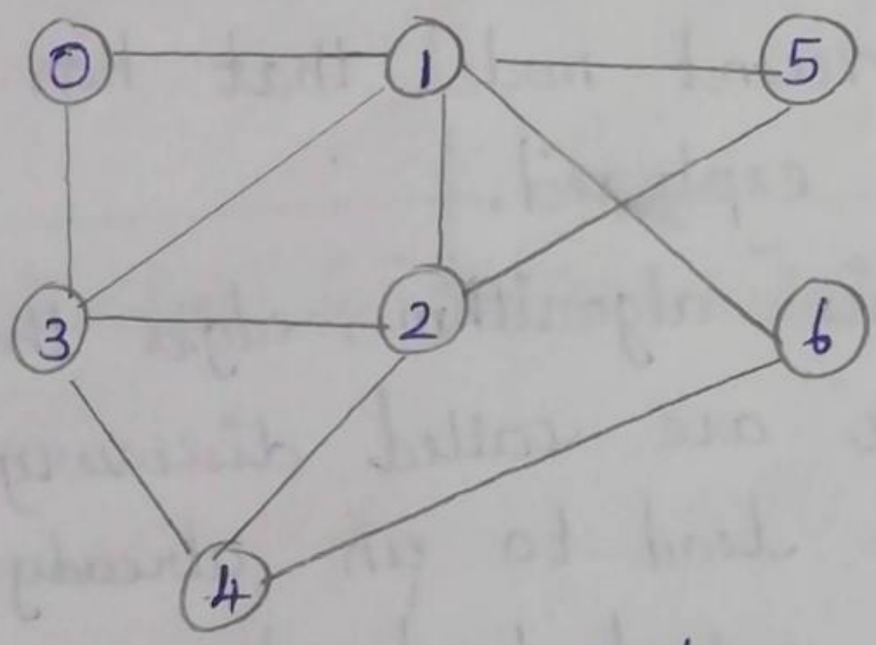
Depth First Search DFS

* It uses a Stack But Both these algorithms uses a variable status

Breadth First Search Algorithm [BFS]

BFS is a graph search algorithm that begins at the not node & explore all the neighbouring node.

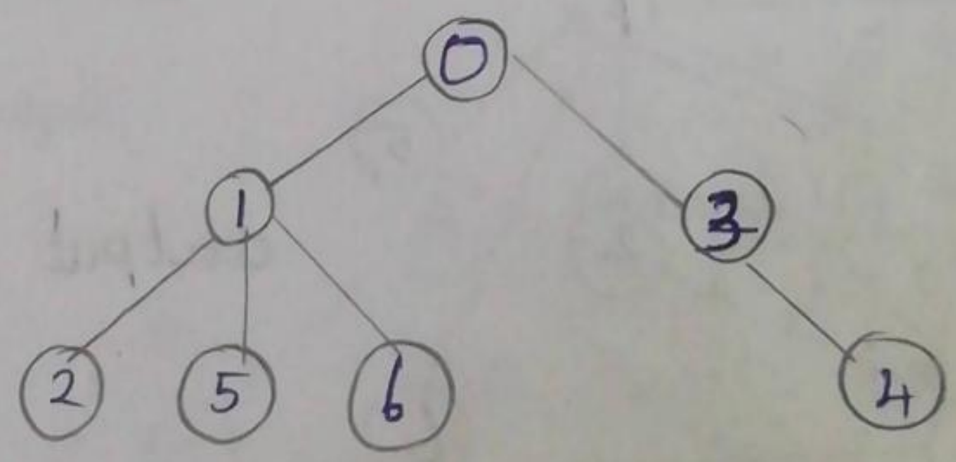
Example: 3, 2, 5, 6



Output : 0, 1, 3, 2, 5, 6, 4

Example :

First 0
How many node going outside



Queue

0	1	3	2	5	6	4

Application of BFS

1) Finding all connected components in a graph G

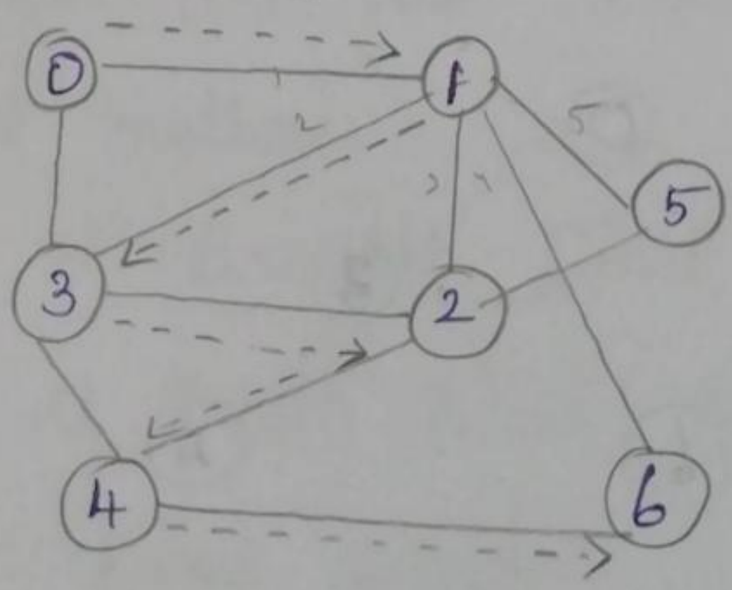
2) Finding the shortest two node u, v of unweighted graph or weighted graph.

Depth First Search: Algorithm:

* The DFS algorithm progresses by expanding the starting node of G & then going deeper & deeper until the goal node is found or Until a node that has no children in encountered When a dead end is reached the algorithm back tracks, returning to the most recent node that has not been Completely explored.
* In this algorithm, edges that lead to a new vertex are called discovery edges & edges that lead to an already visited Vertex are called back edges.

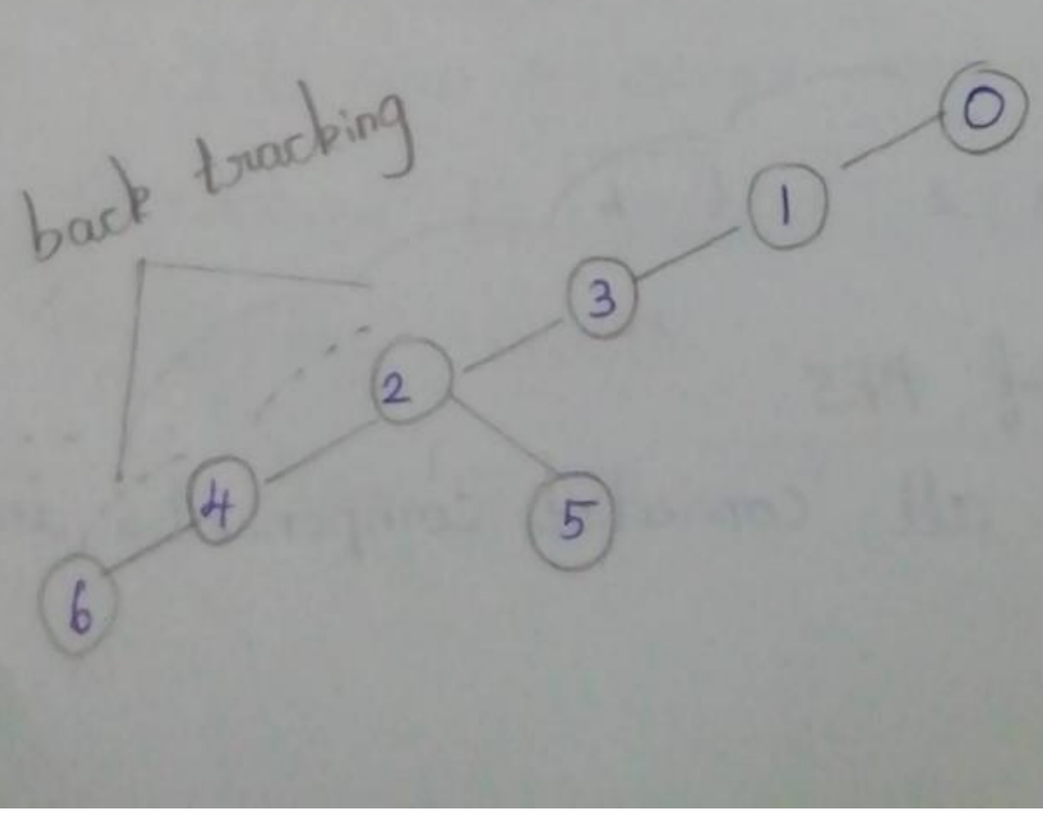
Example

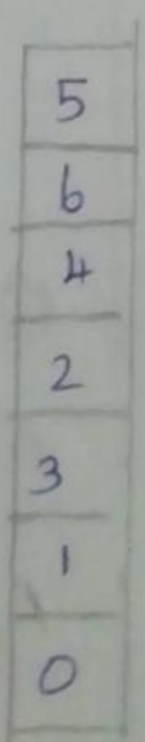
Unweighted using stack order



back tracking is possible.

output 0, 1, 3, 2, 4, 6, 5





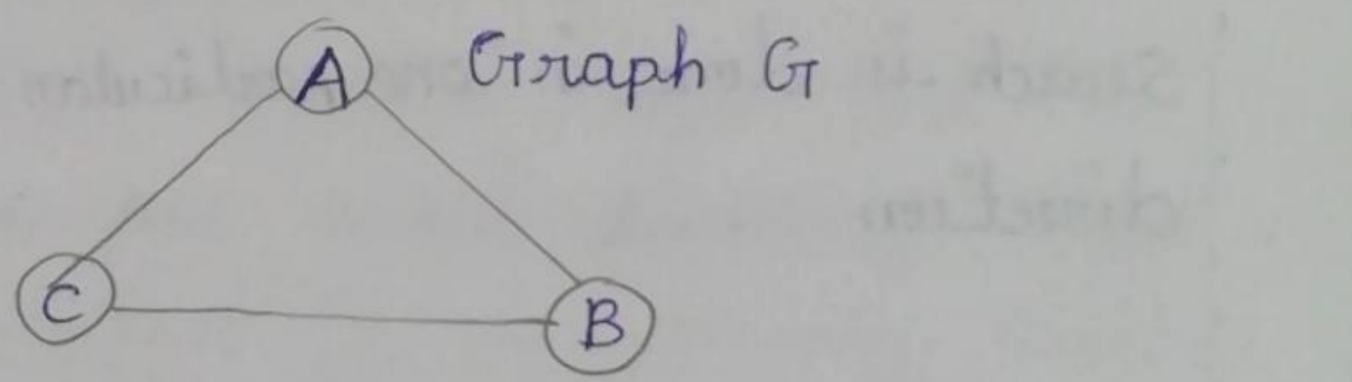
Stack format (LIFO)

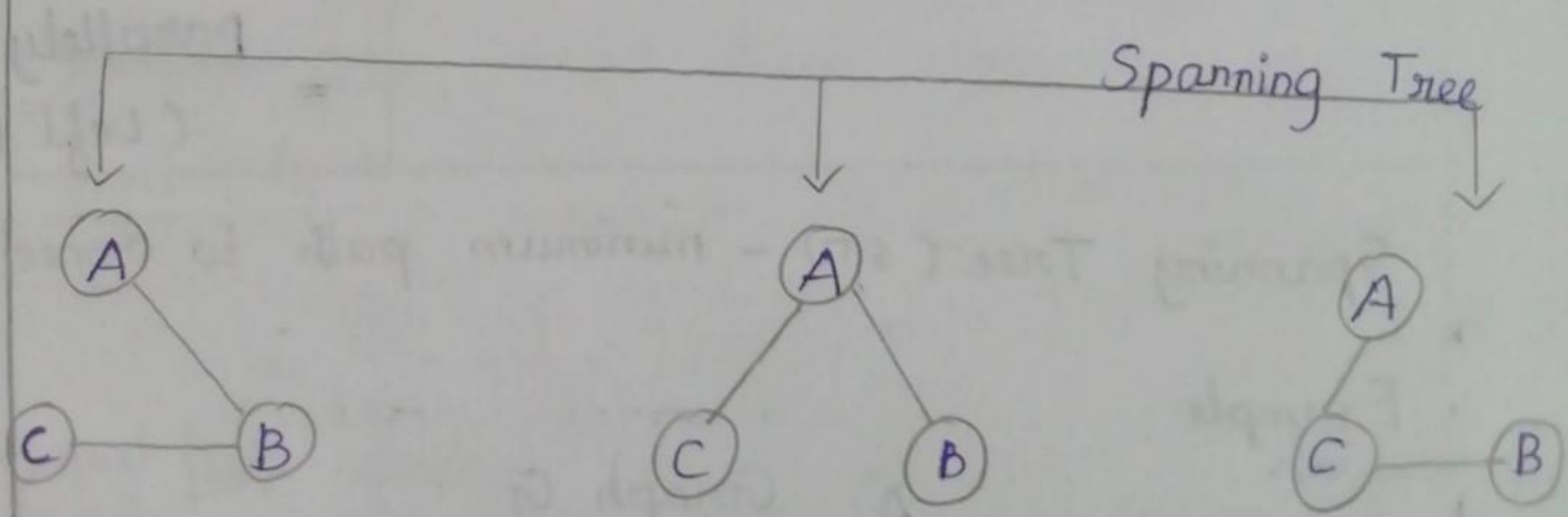
DFS Vs BFS

DFS (Depth First Search) stack order back tracking	BFS (Breadth First Search) FIFO not back tracking
Back tracking is possible from a dead end	Back tracking is not possible
Vertices from which exploration is incomplete & processed in a LIFO order	The vertices to be explored are organized an FIFO order.
Search is done in one particular direction	The vertices in the same level are maintained parallely

Spanning Tree :

* It is a subset of graph G, which has all the vertices covered with minimum possible no. of. edges. Hence, a spanning tree does not have cycle & it can't be disconnected
* By this definition, we can draw a conclusion that every connected & undirected Graph G has atleast one Spanning tree. A disconnected graph does not have any spanning tree, as it can't be spanned to all its vertices

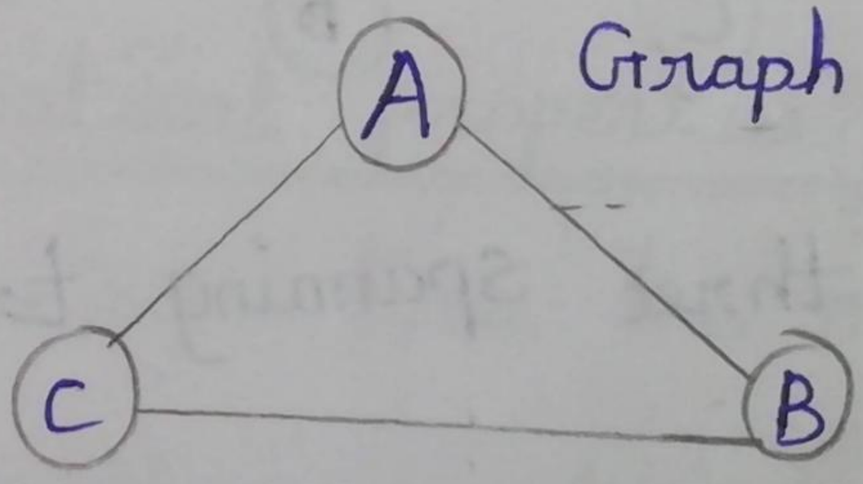




* we Found three spanning trees off one complete graph.
* A complete undirected graph can have maximum n^{n-2} no. of. spanning trees, where n is the no. of. nodes
* In the above addressed example $3^{3-2} = 3$ spanning trees are possible.

Spanning Tree (ST) - minimum path to connect

Example

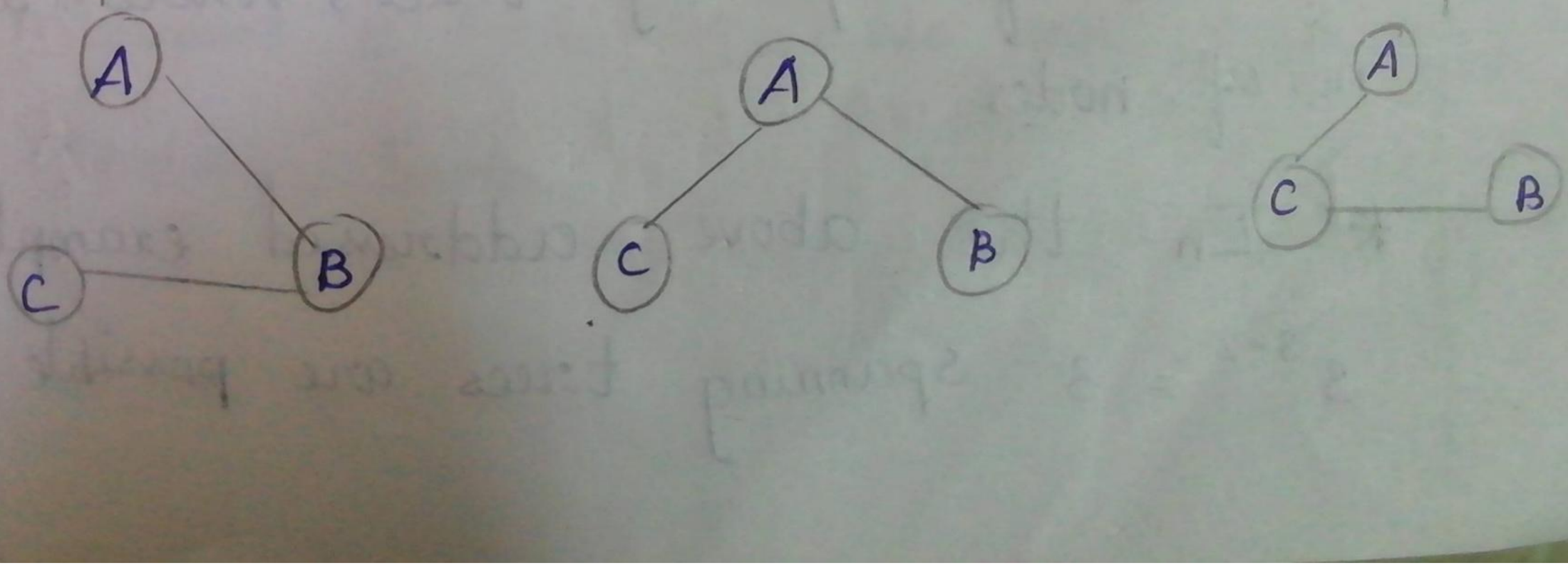


Graph G

self loop - (not allow cycling loop)

S ⟶ D

Spanning Tree



prim's Algorithm :

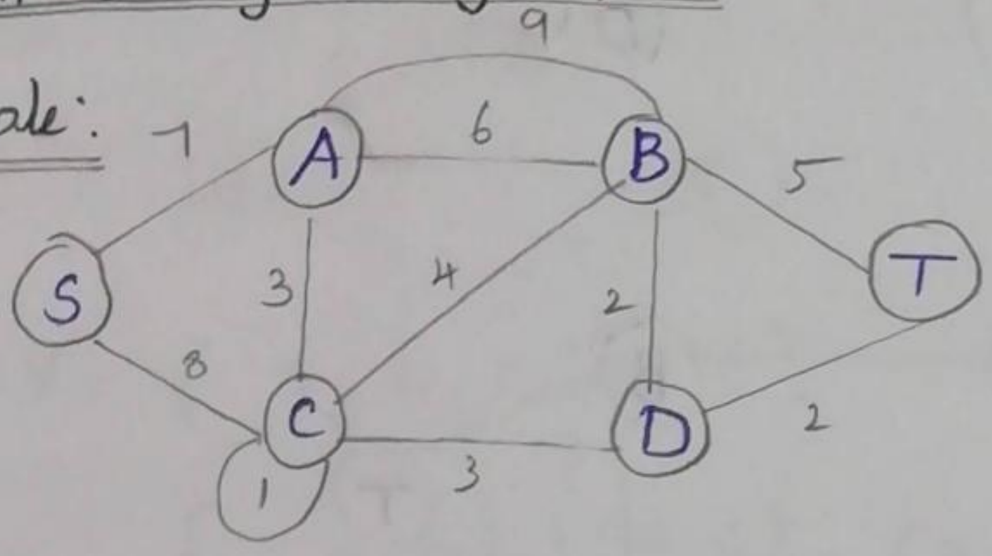
* It to Find minimum cost spanning tree uses the greedy approach.

* It shares a similarity with the shortest path first algorithms.

* It in contrast with kruskal's algorithm, treats the nodes as a single tree & keepson adding new node to the spanning tree.

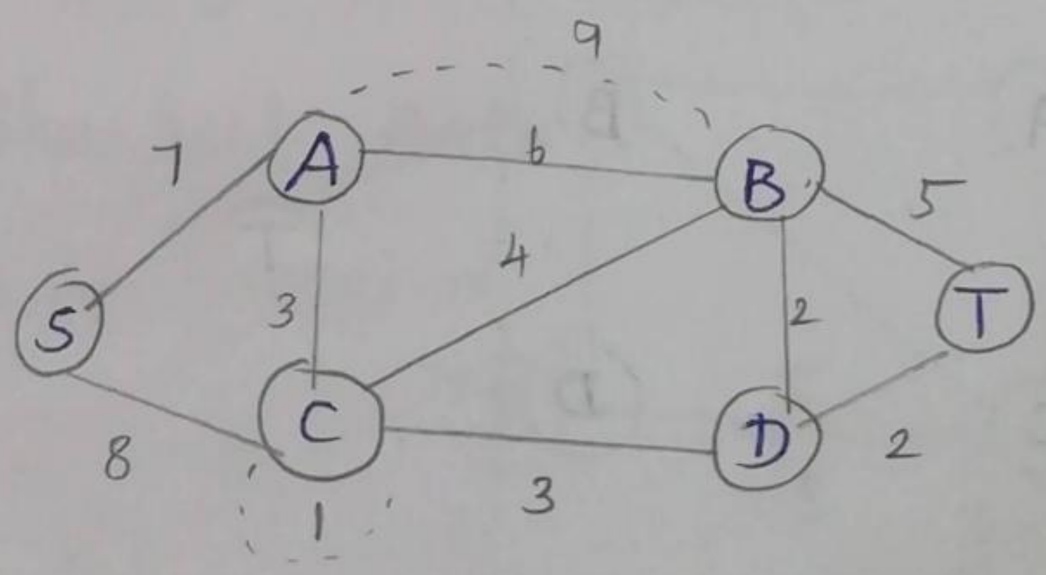
From the given graph:

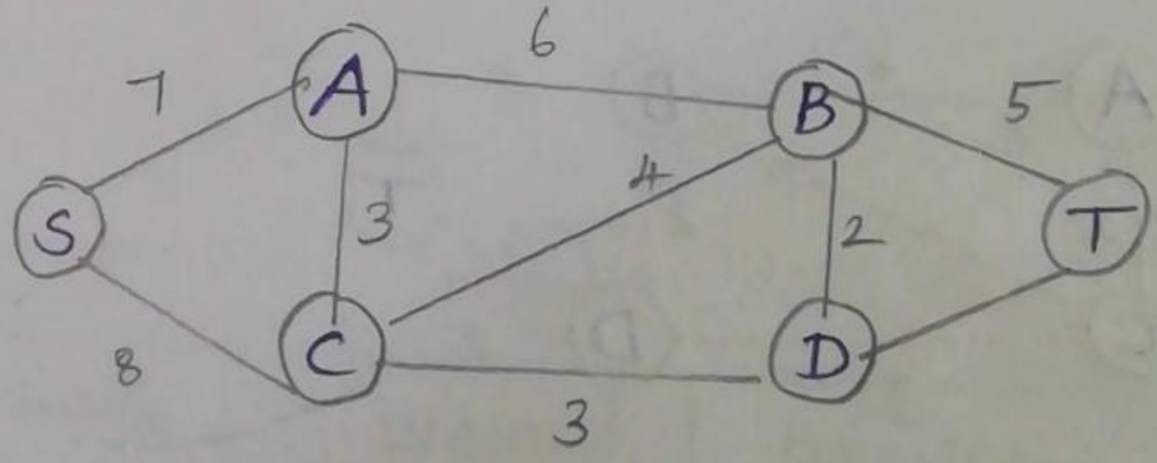
Example:



Step 1:

Remove all loops & parallel edges





Step 2:

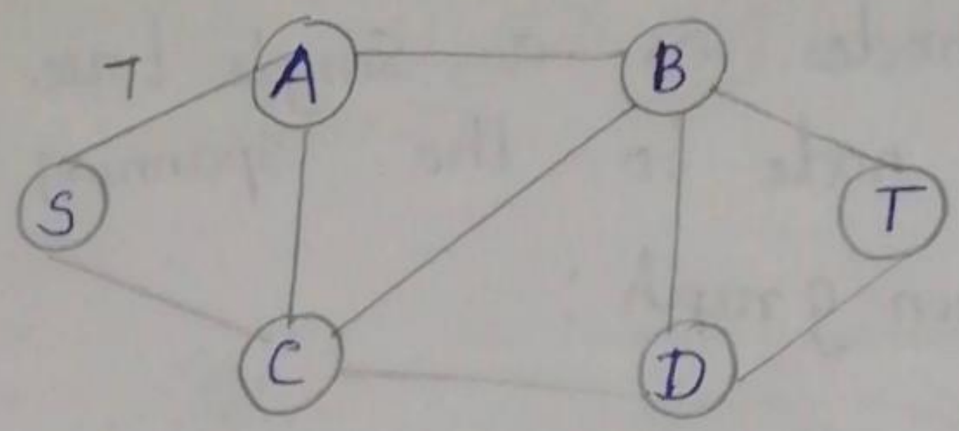
Choose any arbitrary node as not node

* Assumphon S - T

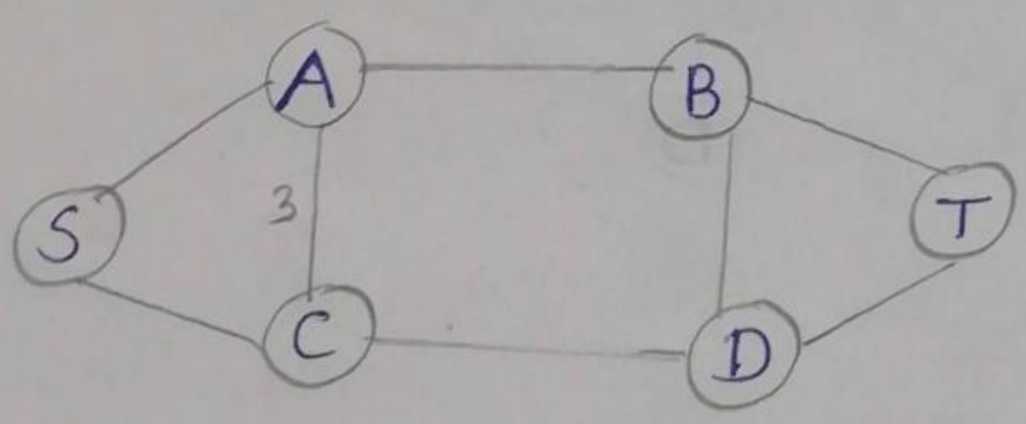
Step 8:

Outgoing edges & select the one with less cost

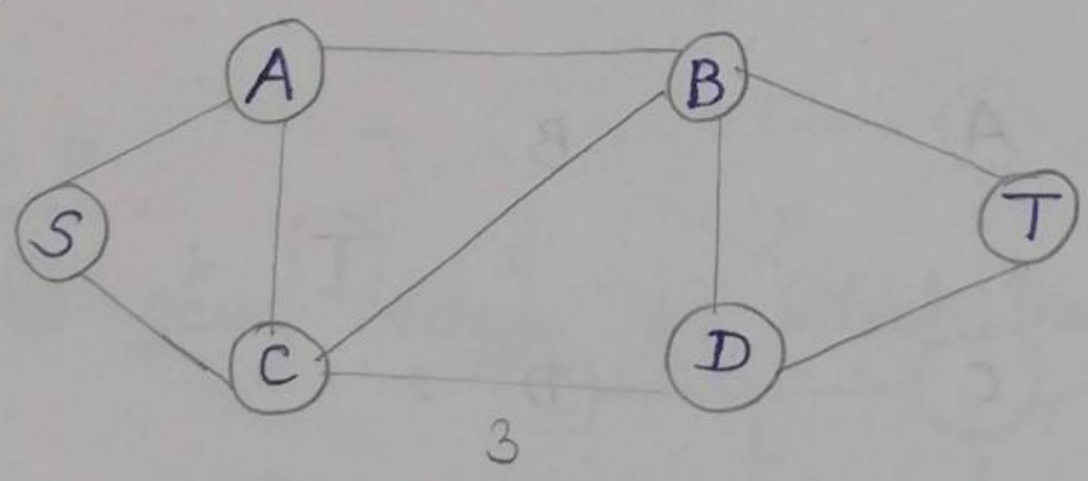
i)



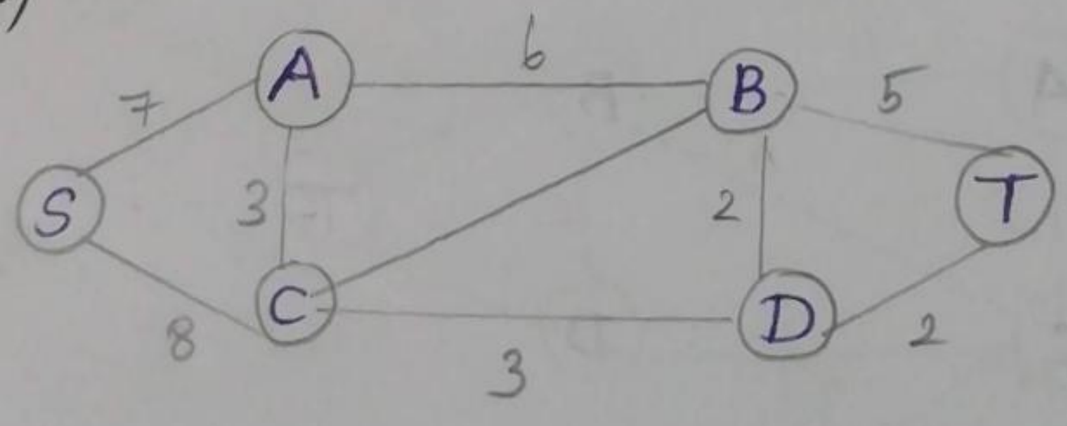
ii)



iii)



iv)



Note: 1) all should visit the all path.

2) No cyclic graph / loop.

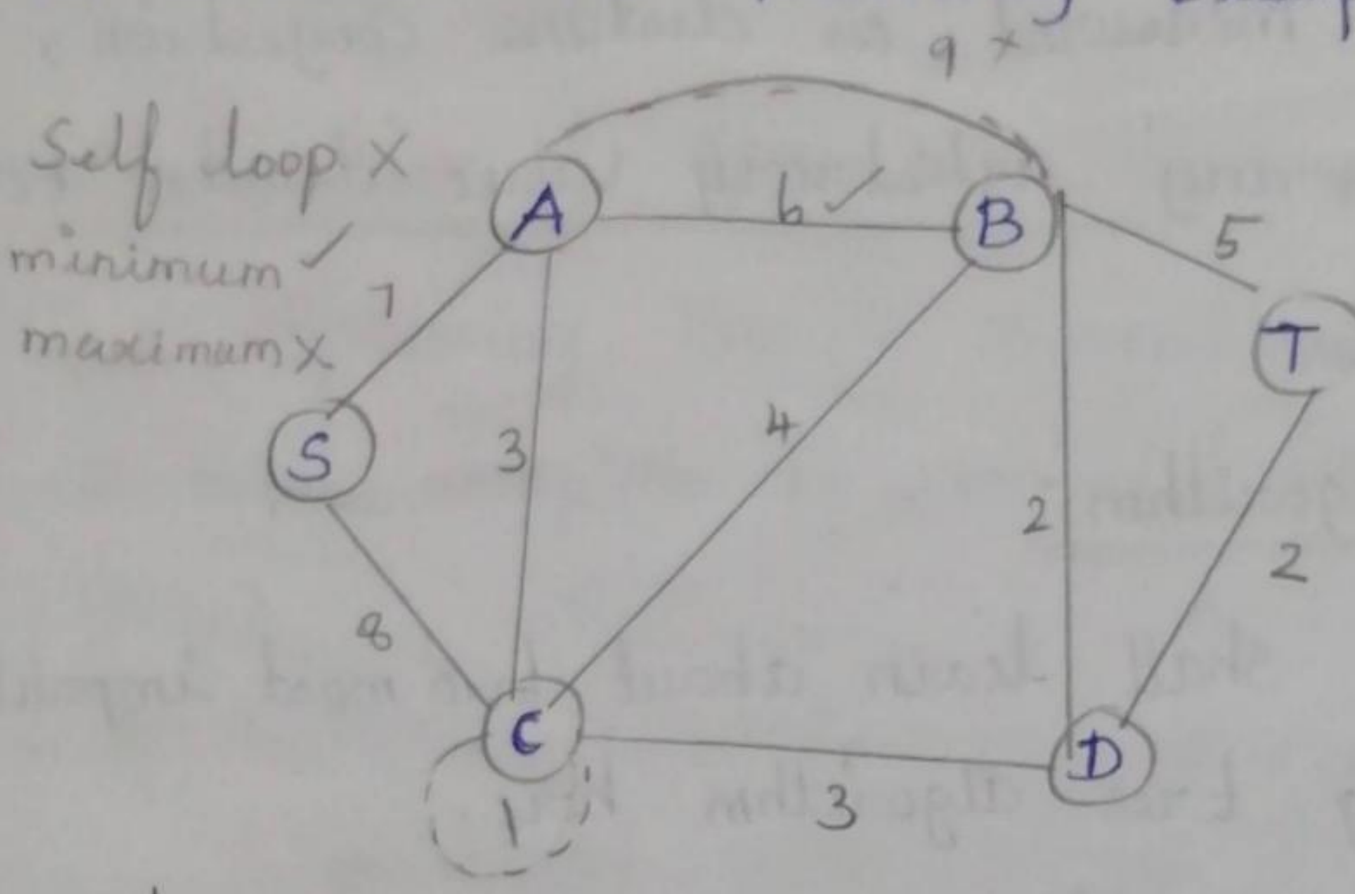
Kruskal's Spanning tree algorithm.

* It is to find the minimum cost spanning trees uses the greedy approach.

* This algorithm treates the graph as a forest & every node it has as an individual trees.

* A tree connects to another only if only if it has the least cost among all available options & does not violate MST properties.

* To understand Kruskal's algorithm let us consides the Following Example



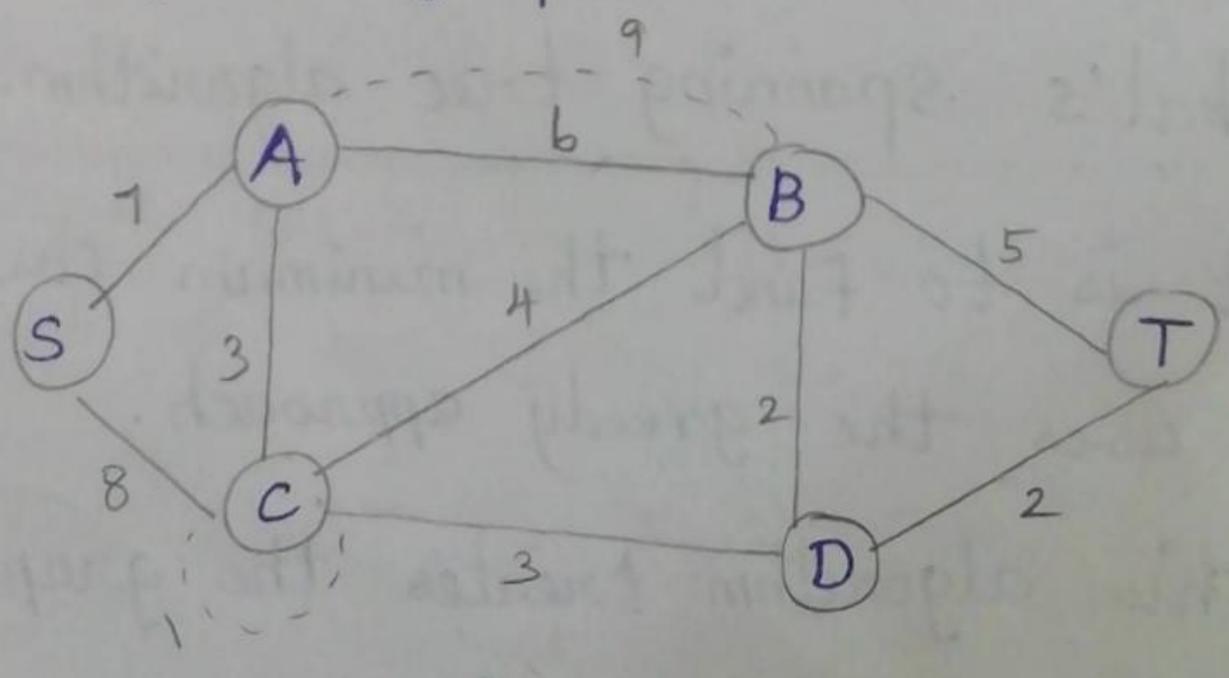
1. Self loop ✗
2. A to B, B to A minimum value

S-A-C-D-T
7 3 3 2

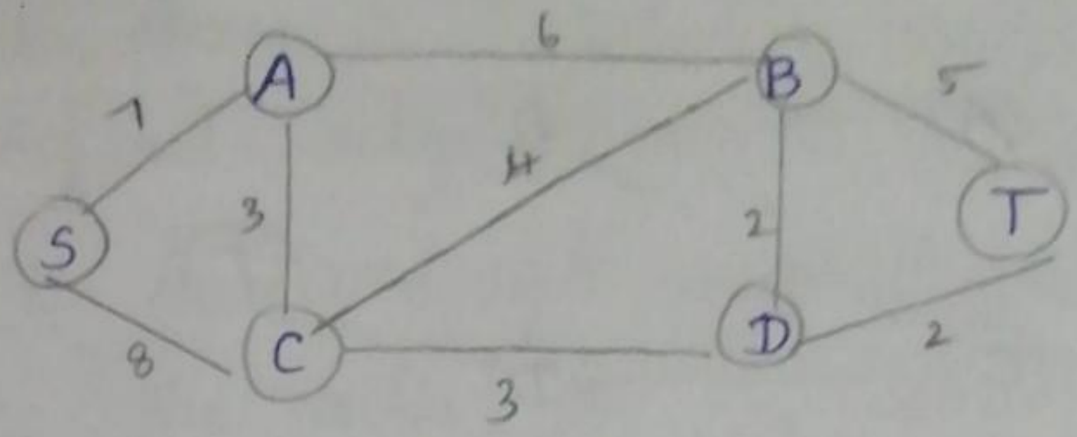
Step 1:

Remove all loops and parallel Edges.

* Remove all loops & parallel edges from the given graph.



* In case of parallel edges, keep the one which has the least cost associated & remove all others.



step 2:

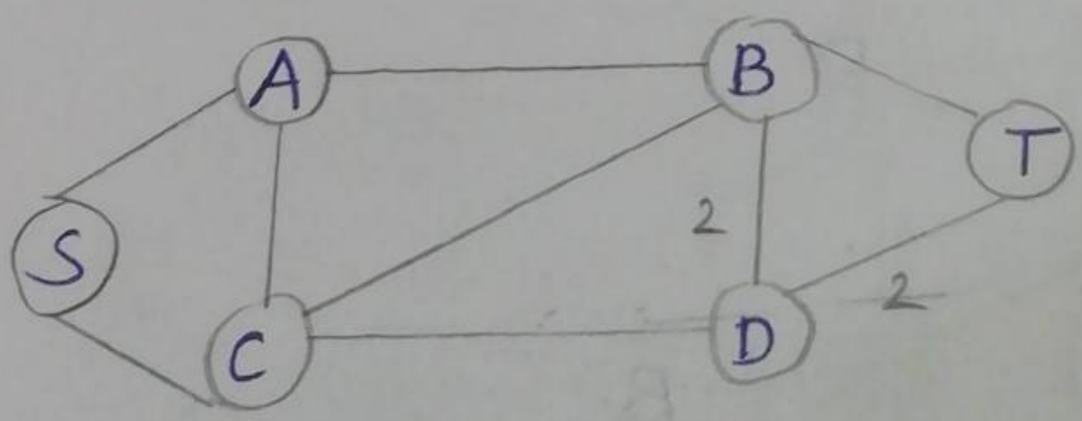
✓Arrage All edges in their increasing order of weight.

* The next step is to create a set of edges & weight, & arrange them in an ascending order of weight are (cost)

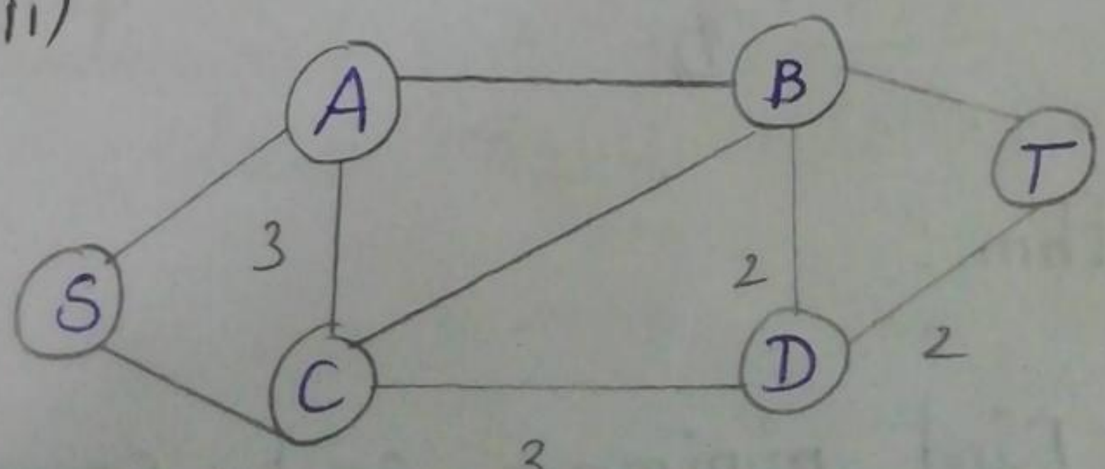
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Step 3: Add the edge which has the least weightase.

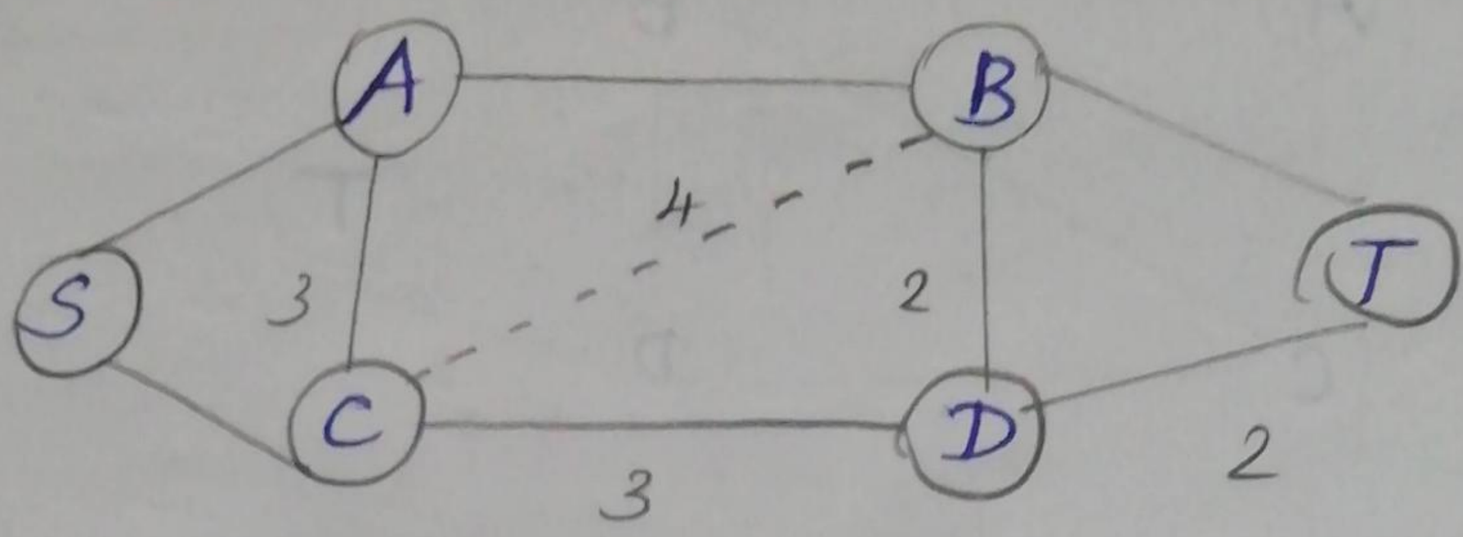
i)



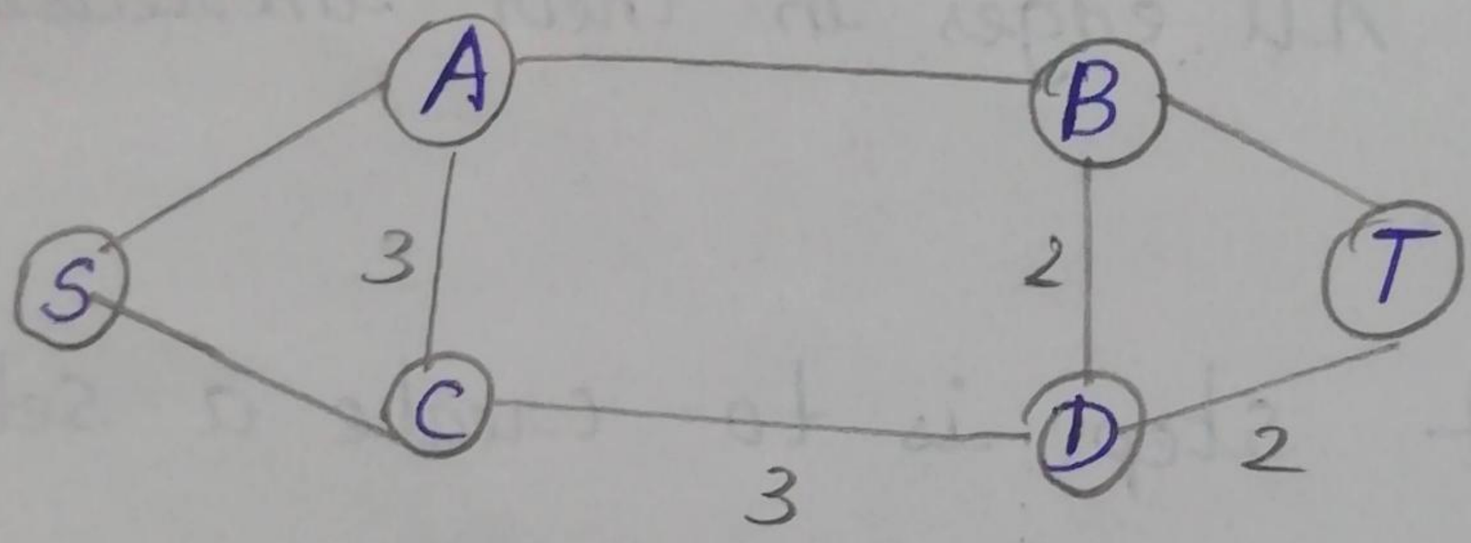
ii)



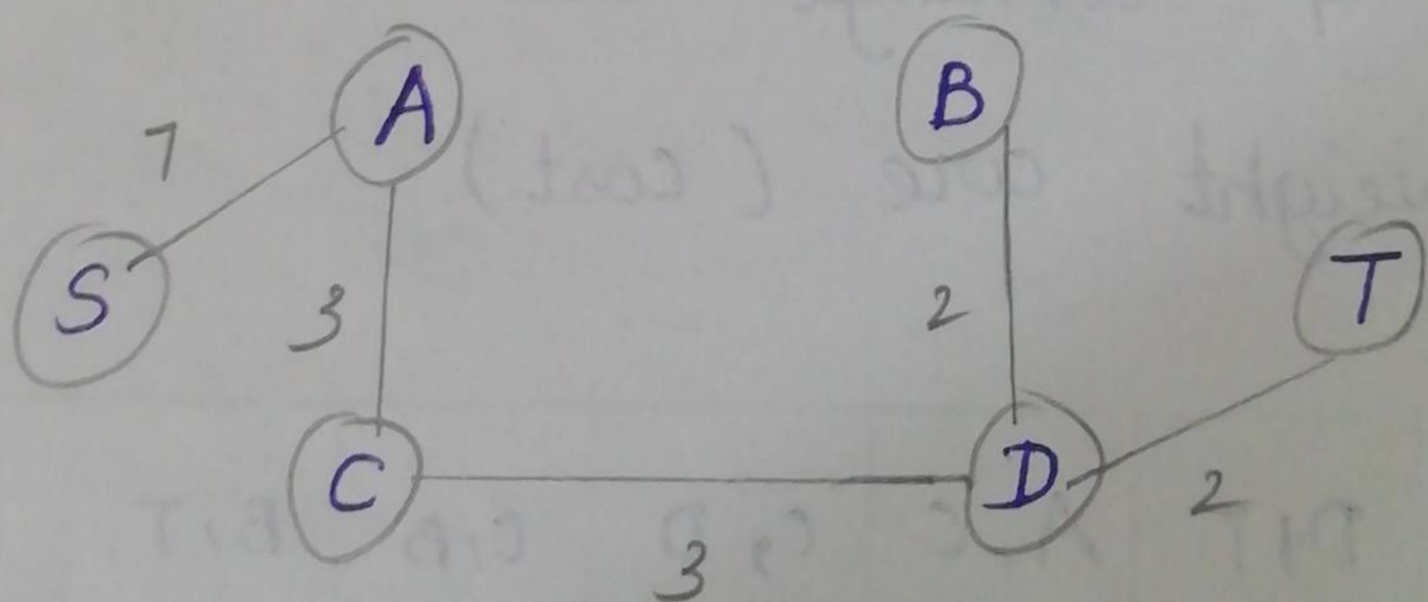
iii)



iv)



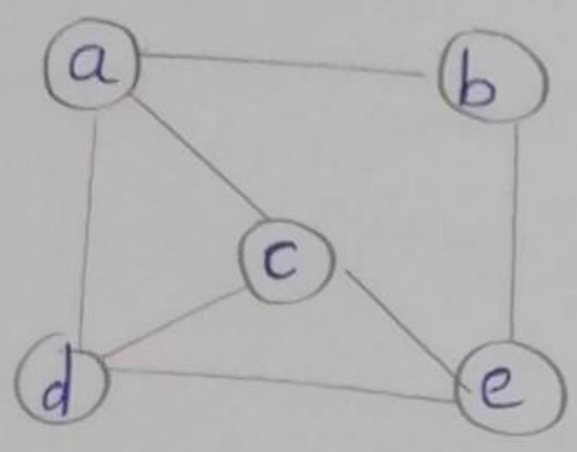
v)



Dijkstra's Algorithms

* Shortest path

Example :



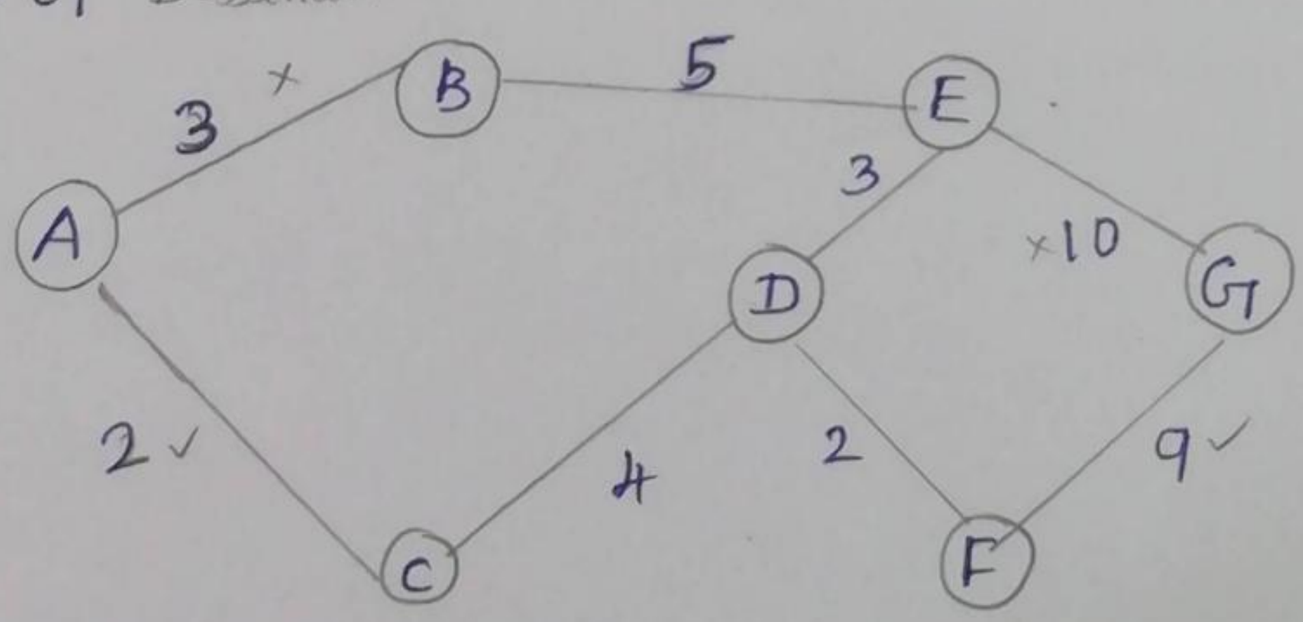
Vertices V = { a, b, c, d, e }

E = { (a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e) }

MST weighted Graph

Shortest Bath

A → G Desination.



Node	weight	previous
A	0	NULL
B	3 ×	A
C	2 ✓ 1st	A
D	6	C
E	8	B
F	8	D
G	9 10	F

A - C - D
2 + 4 = 6

A - B - E = 3 + 5 = 8 ✓

A - B - E = 3 + 5 = 8 ×

A - C - D - E = 2 + 4 + 3 = 9 ×

A - C - D - F = 2 + 4 + 2 = 8

A - D
2 + 4 = 6

A - B - E - G
3 + 5 + 2 = 10 ×

A - E
3 + 5 = 8

A - C - D - F - G
2 + 4 + 2 + 1 = 9 ✓

A → C → D → F → G

1. Table Completed