Graphs :—

definition of graphs in semester ① in DMS.

Representation of graphs :—

Graphs can be represented in many ways. some of the representations are

i) set representation
ii) sequential representation
iii) linked representation

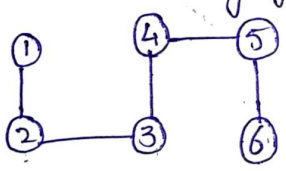
i) set representation :—

In this representation two sets are maintained. They are

① set of vertices represented as V(G)

② set of Edges which is a subset of the given vertex (V×V) which is represented as E(G).

Example :— Consider the following graph.



$V(G) = \{1,2,3,4,5,6\}$

$E(G) = \{(1,2)\ (2,3)\ (3,4), (4,5)\ (5,6)\}$

Advantage :— * from the memory point of view the set representation is more (~~efficient~~) efficient.

dis-advantage :— * The set representation does not allow storing the parallel edges in a multigraph.

Sequential representation :—

A graph can be represented in a sequential manner by using matrices.

1) adjacency matrix
2) Incidence matrix

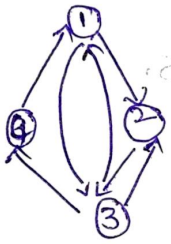
1) Adjacency matrix :-

This matrix provides the information (about) whether the vertex is adjacent to another vertex or not.

$a_{ij} = 0$ ⟶ No ~~(adjacent)~~ edge b/w vertices

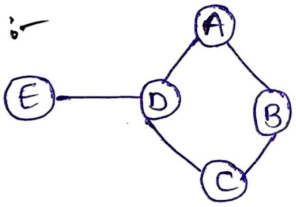
$a_{ij} = 1$ ⟶ If there is an edge from v_i to v_j

⟶ The representation of matrix is called as adjacency matrix. because the Entries in the matrix are represented with 0 or 1.

Ex-:① 

corresponding adjacency matrix

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

Ex-② :- 

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

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

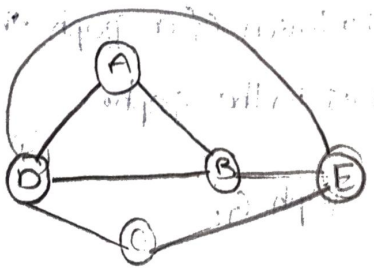
Hw Draw a graph for the correspondent adjacency matrix.

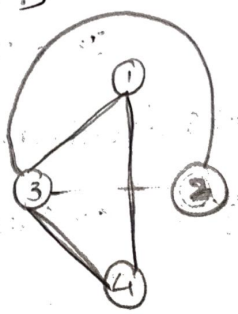
i)

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

ii)

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

i) 

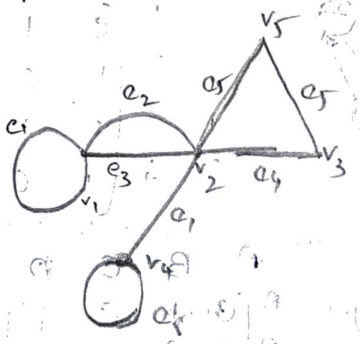
ii) 

Incident matrix :-

Let 'G' be a graph with 'n' vertices and 'e' edges then the incident matrix is a matrix of order nxe where 'n' is represented as corresponding rows and 'e' is represented as corresponding column in the matrix.

$$a_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ edge of } e_j \text{ is no incident to } i^{th} \text{ vertex } v_i \\ 0, & \text{otherwise} \end{cases}$$

Ex-: Suppose the graph G is as follows.



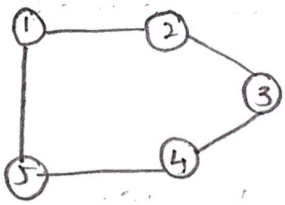
The incident matrix for the graph G is

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1	1	1	0	0	0	0	0
v_2	0	1	1	1	0	1	1	0
v_3	0	0	0	1	1	0	0	0
v_4	0	0	0	0	0	0	1	1
v_5	0	0	0	0	1	1	0	0

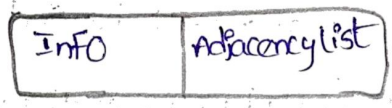
Linked list representation :-

In a linked list representation of a graph the number of links depends on number of vertices in the graph.

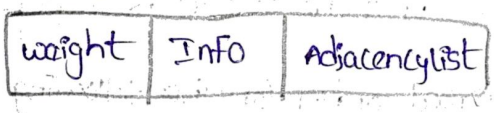
Ex-: consider the following graph G.



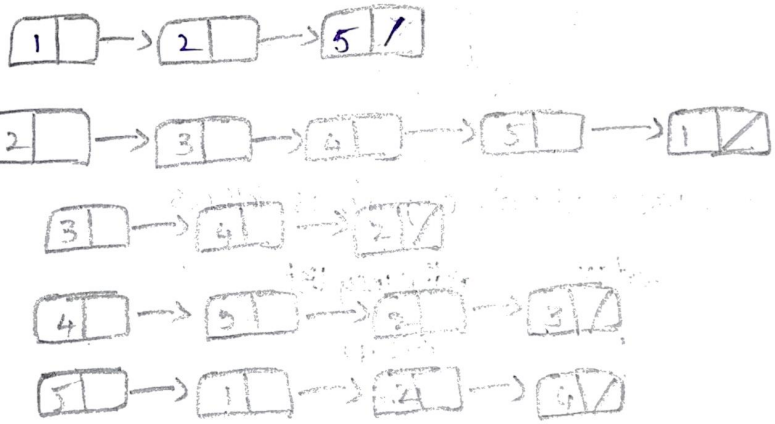
note:- i) for a non-weighted graph the node is



ii) for a weighted graph the node is



The linked list representation of the above graph is



Graph Traversal :-

Traversal is nothing but visiting each node in a systematic order. There are 2 methods for Graph Traversal.

1) Breadth first search (BFS)
2) Depth first search (DFS)

i) Breadth first search :-

→ BFS stands for breadth first search.

→ This search is also known as level order traversal.

→ Queues data structure is use for this traversal method.

iv) In this method we can choose any node as a starting node.

procadure:-

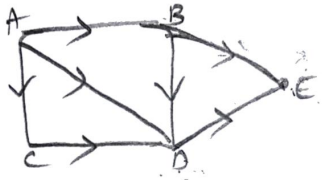
i) Insert the ~~stopping~~ starting node into the queue.

ii) Delete the front Element from the que and insert all unvisited visited vertices into the queue, at the End and traverse the vertices.

note:- whenever a node is visited ~~to~~ mark the value in the array has true.

iii) Repeat the step no (2) until the queue is Empty.

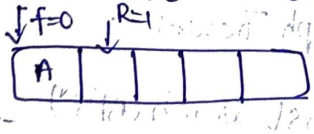
Ex:- construct the traversal order for the following Graph using BFS method?



The adjacency list in the above graph as follows.

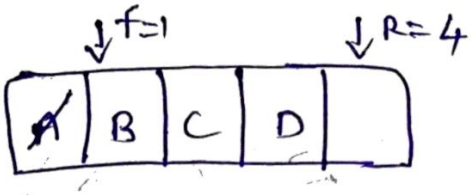
vertex	Adjacency list
A	B, C, D
B	D, E
C	D
D	E
E	—

i) initially insert 'A' into the queue.

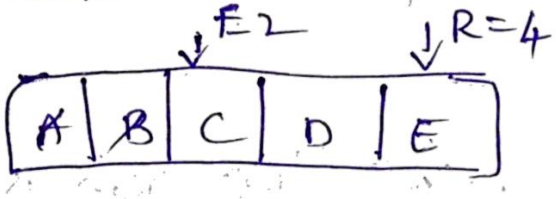


ii) Remove the front Element 'A' from the queue increment the front pointer by 1.

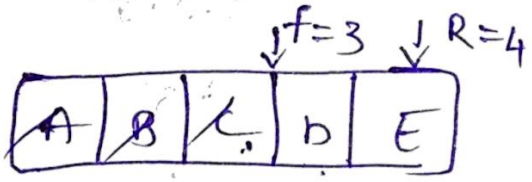
Now insert all the corresponding adjacency Element of the deleted vertex 'A' into the queue.



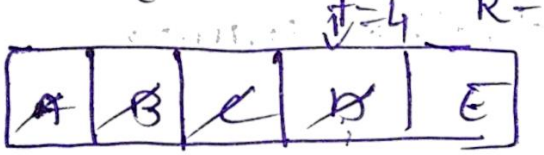
iii) Delete the front element from the queue and add the adjacent elements of the deleted element into the queue.



iv) Delete 'C' element from the queue and add the adjacent elements of the deleted element into the queue.



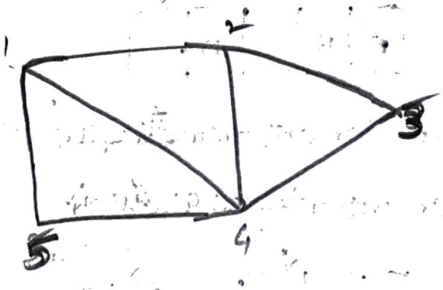
v) Delete 'D' element from the queue and add the adjacent elements of the deleted element into the queue.



vi) Delete 'E' element from the queue. since there is no adjacent Elements to be visited the queue is empty we can stop the process.

The traversal nodes are A, B, C, D, E.

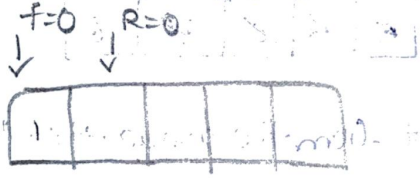
HW



The adjacency list in the above graph as follows.

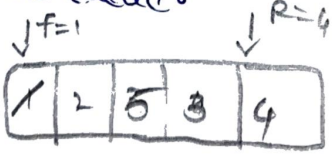
vertex	adjacency list
1	2,3,4
2	4,5
3	4
4	5
5	—

i) initially insert 'i' into the queue.

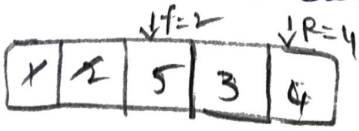


ii) Remove the front element 'i' from the queue increment the front pointer by '1'.

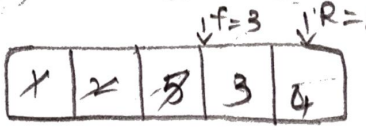
now insert all the corresponding element of the deleted vertex 'i' into the queue.



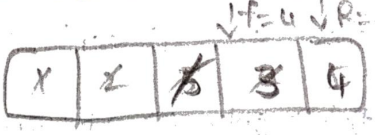
iii) delete the front element from the queue and add the adjacent elements of the deleted element into the queue.



iv) delete 5 element from the queue and add the adjacent elements of the deleted element into the queue.



v) delete 4 element from queue and add the adjacent elements of the deleted element into the queue.



vi) Delete 5 element from the queue. since there is no adjacent elements to be visited the queue is Empty.

we can stop the process.

The traversal nodes are 1, 2, 5, 3, 4.

DFS :-

i) DFS stands for depth first search

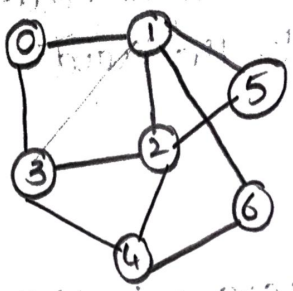
ii) This is also one of the traversal techniques, which works on the principle Last-in-first out

iii) This technique uses stack data structure for traversal.

iv) we can take any node while traversing and consider it as root node or starting node of the graph.

* In case of BFS the element which is deleted from the queue, then the adjacent nodes of that deleted element is inserted into queue. But in dfs when a node is ~~deleted~~ removed from the stack then only one adjacent node is added into stack.

consider the following graph.



i)

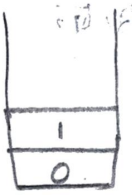
step-1 - first we insert the Element '0' in the stack.

This node contains two adjacent nodes as 1 and 3.

now we can take only one adjacent node and push into stack for traversal.

step2 - suppose we take ~~or~~ 1 as traversing vertex then '1' is pushed into stack and the stack is printed.

ii)



step-3 - now we will identify the adjacent vertices of the node '1'

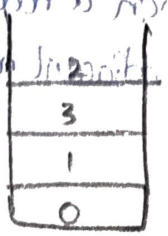
[3,2,5,6] are the vertices.

step-4 - now we consider node '3' as the traversal vertex and push into the stack and print the stack.

iii)

we identify unvisited adjacent vertex of 3.

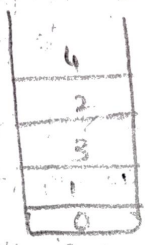
iv)



The adjacent vertices for node `3` are (0, 1, 2, and 4). among these 0, 1 are already visited. Among 2 and 4 `2` has been unvisited more than `1`. so we push `2` into the stack and print.

Step-5 - now the unvisited adjacent vertices of `2` are (1, 3, 4, 5) we consider `4` as the visiting vertex and push on to the stack and print the stack.

v)



step-6 - now we will consider the un visited adjacent vertices of node `4` (that is 6). Therefore `6` is pushed into the stack and printed.

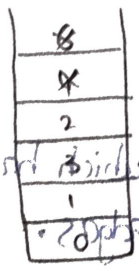
vi)

Back tracking :-

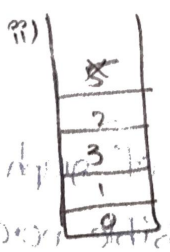
* Now we delete each node available in the stack and verify whether any unvisited vertex is available or not.

* If we find any unvisited adjacent vertex then push that node into stack and again repeat the deletion process of the nodes in the stack.

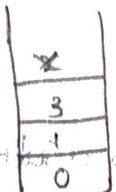
i)



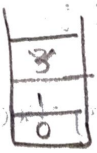
ii)



iii)



iv)



v)

vi)

stack is empty.

so we stop the process and print the final traversal nodes.

0 → 1 → 3 → 2 → 4 → 6 → 5

Differences between BFS and DFS :—

Breadth first search	Deapth first search
→ BFS stands for breadth first search.	→ (Deapth) DFS stands for deapth first search.
→ It is a vertex based technique to find the shortest path in a graph.	→ It is an edge based technique to find the shortest path.
→ Queue datastructure is used for BFS traversal.	→ stack datastructure is used for DFS traversal.
→ BFS does not allow backtracking concept.	→ DFS allows backtracking concept.
→ BFS is ~~fast~~ slower than DFS technique.	→ DFS is faster than BFS techniques.
→ BFS is not memory efficient when compared with DFS.	→ DFS is more efficient than BFS. since it occupies less memory space.
→ BFS technique is not suitable for decission trees.	→ DFS technique is more suitable for exploring the decission trees.

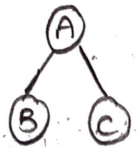
V.V.V IMP topic

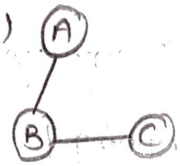
Spanning trees :—

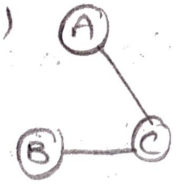
The spanning tree is a subset of graph G which has all the vertices covered with minimum possible no. of edges.

Ex→ consider the following graph.

The spanning tree that can be constructed from the above graph are

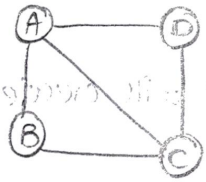
i)


ii)


iii)


note :- no. of spanning trees that can be constructed from an undirected graph can be calculate by using the formula n^{n-2} where 'n' is the no. of vertices or nodes

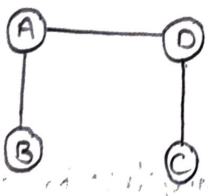
Ex-2 : construct different spanning trees for the following graph.

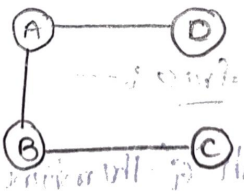


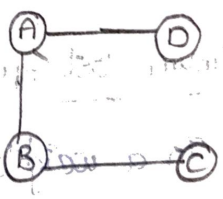
In the given graph we have 4 nodes. so the no. of spanning trees we can get = n^{n-2}

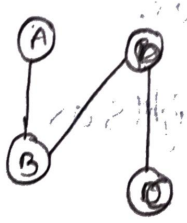
$= 4^{4-2}$

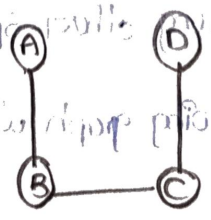
$= 4^2 = 16$

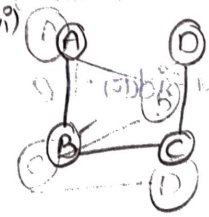
i)


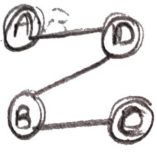
ii)


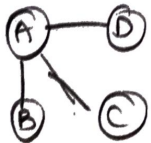
iii)


iv)


v)


vi)


vii)


viii)


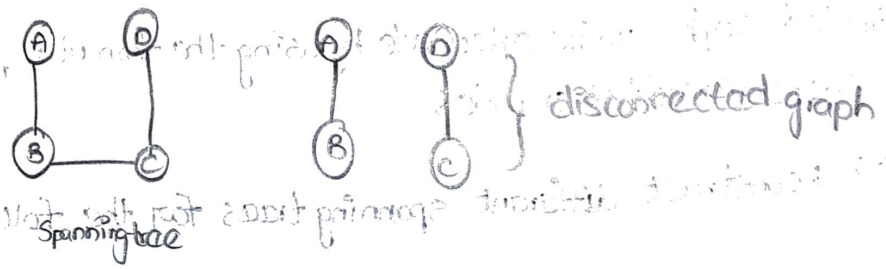
properties of spanning tree :-

1) for a connected graph 'G' we can have more than one spanning tree.

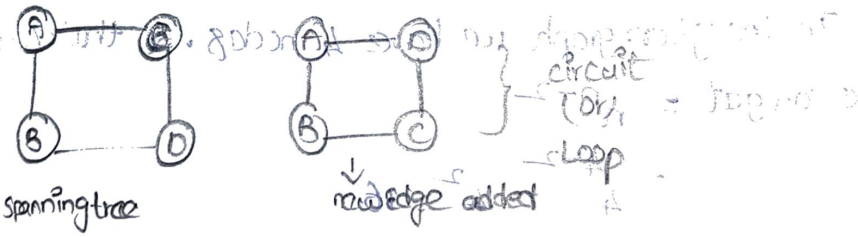
2) all the possible spanning trees are of a graph 'G' have the same no. of edges and vertices.

3) The spanning tree does not have any cycle (loop).

4) Removing one edge from the spanning tree will make the graph dis-connected.



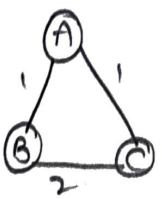
5) Adding one edge to the spanning tree will create a circuit or loop



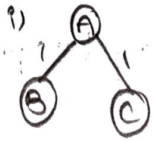
Minimum cost spanning tree :-

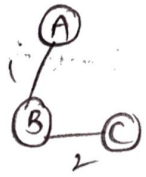
In a weighted graph 'G' the minimum spanning tree is a tree which has minimum weight among other spanning trees.

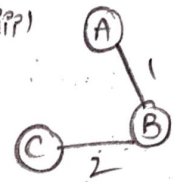
Ex:- consider the following graph with the weights as



The spanning trees for the above graph

i) 

ii) 

iii) 

from the above spanning trees the first spanning tree has least total cost (weight) when compared with other two spanning trees. so the first spanning tree is minimal cost spanning tree.

There are two most popular algorithms to find minimal spanning tree for a weighted graph.

Imp 1) prim's Algorithm

Imp 2) kruskal's Algorithm

1) Kruskal's Algorithm :—

1) This algorithm is used to find the minimum cost spanning tree for the given weighted graph.

2) This algorithm uses a greedy approach.

3) This algorithm ~~(tree)~~ the grey treates the graph as a forest. (Every node is considered as a tree)

4) A node is connected to another node if and only if it has least cost and should not violate all the properties of minimum spanning tree.

Steps :-

The main steps involved in this algorithm :—

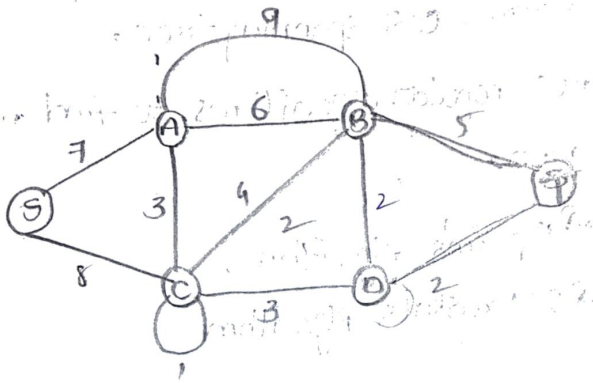
1) Remove all the loops, parallel edges (with maximum weight) from the given graph.

2) Arrange all the edges of the graph obtained from step no(1) in the increasing order of the weights.

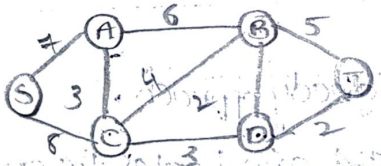
3) now add the edges which has least (cost) weight into the resultant spanning tree.

4) Throughout this process we have to check the spanning tree properties are applied properly.

5) After completion of the spanning tree add all the weights to obtain minimum cost spanning tree for the given graph.



1) Remove all the loops, parallel edges (with maximum weights) from the given graph.

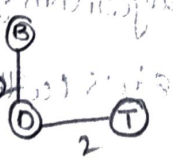


2) Arrange all the edges of the graph obtained from step no (1) in the increasing order of the weight.

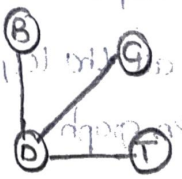
B,D	T,D	D,C	C,A	C,B	B,T	A,B	A,S	S,C
2	2	3	3	4	5	6	7	8

i)

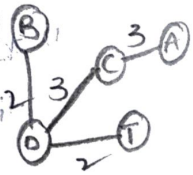
ii)



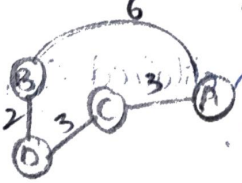
iii)

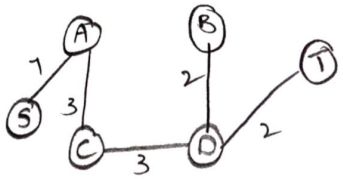


iv)



v)

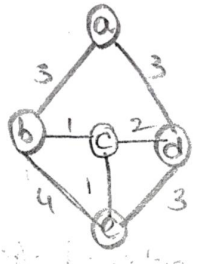




final add all the weights of the edges which gives minimal cost.

$2+2+3+3+7 \Rightarrow 17$

Eg2) construct minimal cost spanning tree for the following graph.

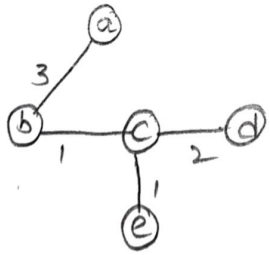


1) In the given graph there is no loops and parallel edges.

2) Arrange all the edges of the graph obtained from step no (1) in the increasing order of the graph.

b,c	c,e	c,d	a,b	a,d	d,c	b,e
1	1	2	3	3	3	4

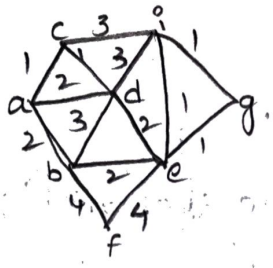
3) add the edges which has least weight into the resultant spanning tree.



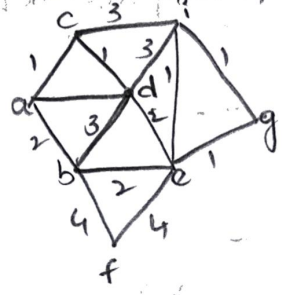
4) After completion of the spanning tree add all the weights to obtain minimal cost spanning tree for the given graph.

$3+1+1+2=7$

Ex-3 constauct minimal cost spanning tree for the following graph.



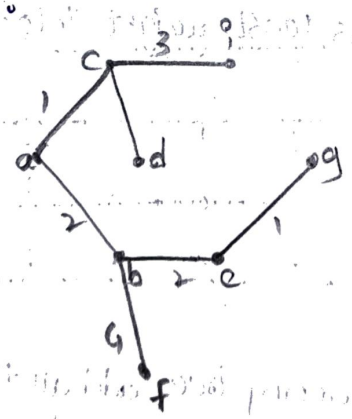
In the given graph there is no loop and parallel edges.



Arrange the all edges of the graph obtained from step no ① in the increasing order of the graph.

a,c	c,d	e,g	g,i	e,i	a,b	b,e	d,e	b,d	c,i	d,i	b,f	e,f
1	1	1	1	1	2	2	2	3	3	3	4	4

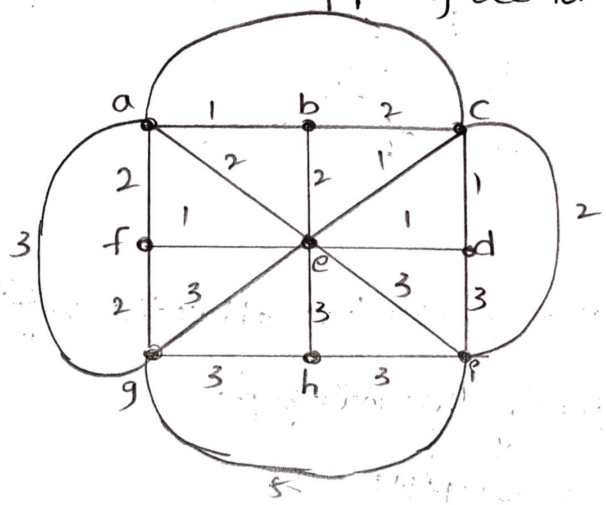
now add the edges which has least weight into the resultant spanning tree.



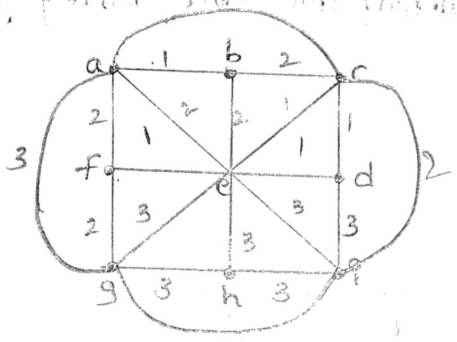
After completion of the spanning tree add all the weights to obtain minimum cost spanning tree in the graph.

$4+2+2+1+1+1+3 = 14$

Eg-4 :- construct minimal cost spanning tree for the following graph.



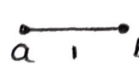
1) Remove all the loops & parallel edges (with maximum weights) from the given graph.

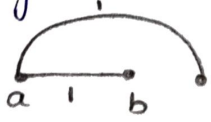


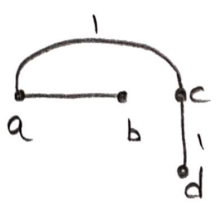
2) Arrange the all edges of the graph obtained from step ① in the increasing order of the graph.

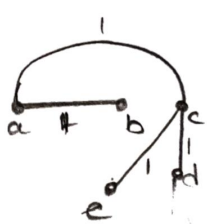
(a,b)	(c,d)	(d,e)	(e,f)	(a,c)	(c,e)	(a,e)	(b,c)	(b,e)	(a,f)	(f,g)	(i,c)	a,g
1	1	1	1	1	1	2	2	2	2	2	2	3
(d,i)	(e,g)	(e,h)	(g,h)	(h,i)	(e,i)	(g,i)						
3	3	3	3	3	3	~~3~~						

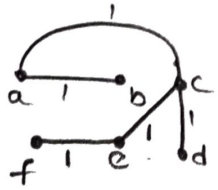
now consider the least weight among the edges and include it to the resultant spanning tree.

i) 

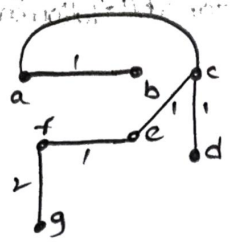
ii) 

iii) 

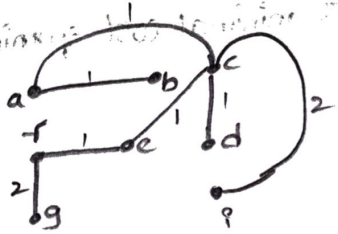
iv) 

v) 

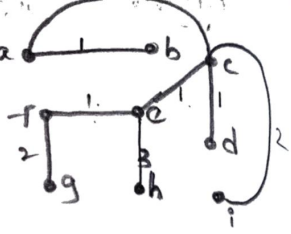
vi)



vii)



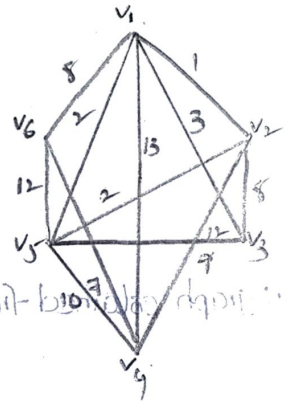
viii)



After completion of spanning tree add all the weights to obtain minimum cost spanning tree in the graph.

$$2+3+2+1+1+1+1+1 = 12$$

Eg-5 – construct a spanning tree for a following graph and calculate the minimal cost.



A (21)

1) In the graph there is no parallel edges and loops.

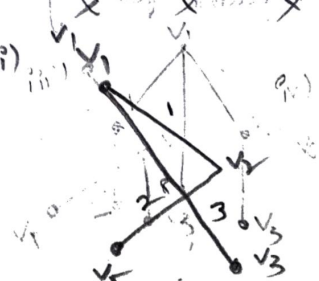
2) Arrange all edges of the graph obtained from step no (1) in the increasing order of the graph.

(V_1,V_2)	(V_2,V_5) V_1,V_5	(V_1,V_5) V_1,V_3	(V_1,V_3) V_4,V_6	(V_6,V_4) V_1,V_6	(V_1,V_6)	(V_2,V_4)	(V_5,V_4)	(V_5,V_3)	(V_5,V_6)	(V_1,V_4)
1	2	2	3	7	8	9	10	12	12	13
✓	✓	X	✓	X	✓	X	X	X	X	X

i)

ii)

iii)



iv)

prims algorithm:-

1) The prims algorithm is also use to find the minimum cost spanning tree, which is similar to krushkal's algorithm.

2) This algorithm also uses greedy approach.

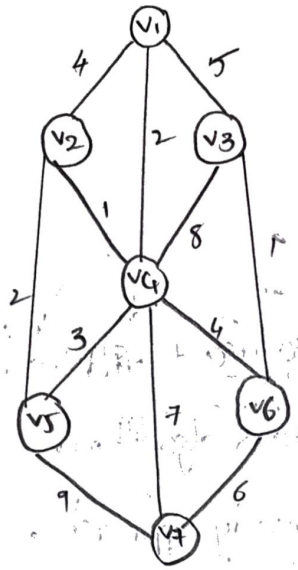
3) This algorithm share the similarities of shortest path first algori-thms.

4) The following are the steps implemented in prims algorithm to construct a spanning tree.

* observe the given graph and remove the loops, parallel edges (edge having highest cost is removed) from the graph).
* choose any node (arbitary node) as a root node.
* check the outgoing edges from the arbitary node, and select the edge with least cost and add it to the resultant spanning tree.
* Repeat step no-3 until all the vertices are Explore.

note:- while Exploring the outgoing edges if we find any edge which leads to cyclic we will not include that edge into the resultant spanning tree.

Ex-1 :- construct minimal cost spanning tree for the following graph.



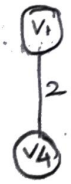
i) consider (V1) as arbitary node (root).

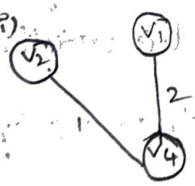
(V1)

ii) Now explore all the outgoing edges of (V1) and choose the least cost as the resultant edge and it to spanning tree.

i) $V_1 - V_2 \longrightarrow$ weight = 4
ii) $V_1 - V_4 \longrightarrow$ weight = 2 ✓
iii) $V_1 - V_3 \longrightarrow$ weight = 5

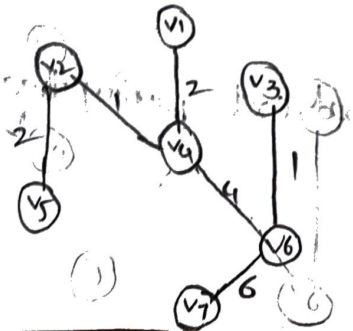
} Arrange these outgoing edges the edge $V_1 - V_4$ has least cost so it is taken as resultant edge and added to the spanning tree.

i)


ii)


$V_4 - V_1 \longrightarrow$ weight of Edge = 1
$V_4 - V_5 \longrightarrow$ weight of Edge = 3
$V_4 - V_6 \longrightarrow$ weight of Edge = 4
$V_4 - V_3 \longrightarrow$ weight of Edge = 8

consider the least cost edge and it to spanning tree.

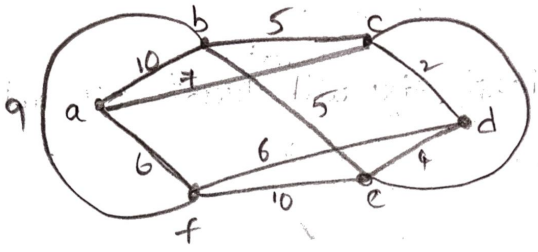
iii)


Total minimal cost 1+1+2+2+4+6

=16

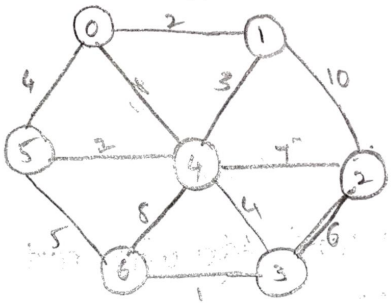
Exa-: construct minimal spanning tree for the following graph for using prims algorithm.

i)



->23

i) (In the given graph) ii)

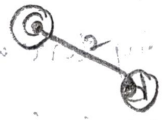


->16

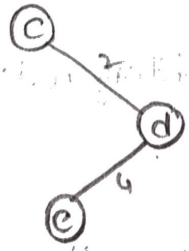
i) remove the loops and parallel edges in the graph.

consider 'c' as the arbitary node.

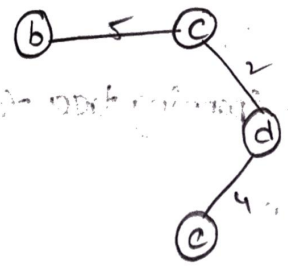
ii) The least weight edge is consider first and add it to spanning tree.



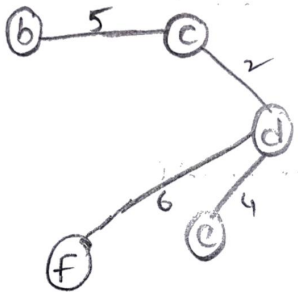
iii) now consider next least Edge and add it to the spanning tree.



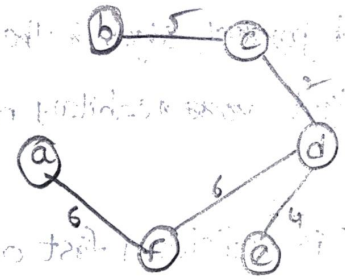
iii) next consider the least Edge with least cost and add it to the spanning tree.



iv) next consider least edge with next least cost and add it to the spanning tree.



v) next consider least edge with next least cost and add it to the spanning tree.

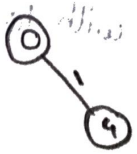


It is the final spanning tree.

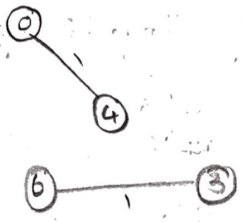
Total minimal cost = 2+4+5+6+6 = 23

iii) i) consider 'o' is the arbitary node.

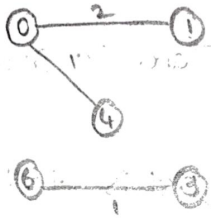
ii) The least weight edge is consider first and add it to the spanning tree.



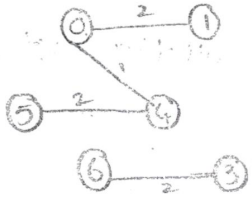
iii) now consider the least Edge with next least cost and add it to the spanning tree.



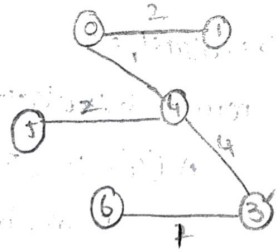
iv) now consider the least Edge with next least cost and add it to the spanning tree.



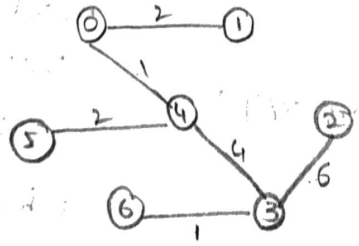
v) now consider the least Edge with next least cost and add it to the spanning tree.



vi) now consider the least Edge with next least cost and add it to the spanning tree.



vii) now consider the least Edge with next least cost and add it to the spanning tree.



It is the resultant spanning tree.

Total minimal cost = 1+1+2+2+4+6

= 16