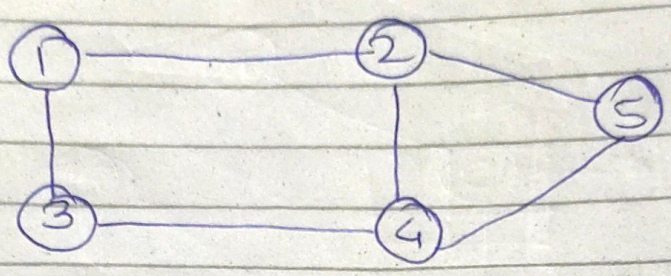
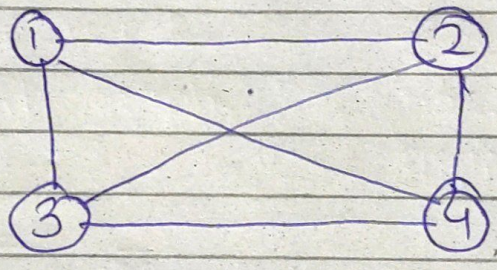
Unit 03

Page No.
Date

* Graph
- Graph is a non-linear data structure which consists of vertices (V) connected by edges(E) where edges may be directed or undirected



① complete Graph :
- each vertex all other vertex is connected ahe.



imp

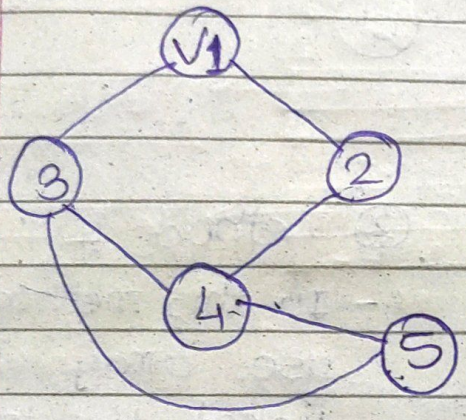
* Storage representation.

1] Adjacency matrix :

- In this representation matrix or 2 dimensional array is used to represent the graph.
- If there is connection between two vertex write 1 otherwise write 0 in matrix.

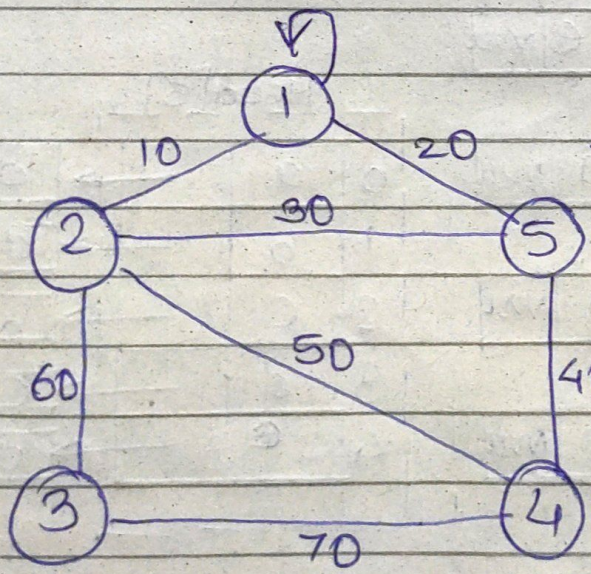
Example :

(1)



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	1	1
4	0	1	1	0	1
5	0	0	1	1	0

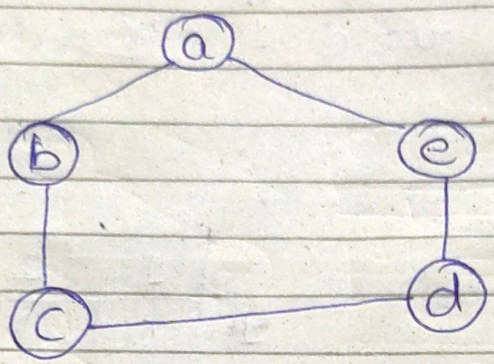
(2)



	1	2	3	4	5
1	0	10	0	0	20
2	10	0	60	50	30
3	0	60	0	70	0
4	0	50	70	0	40
5	20	30	0	40	0

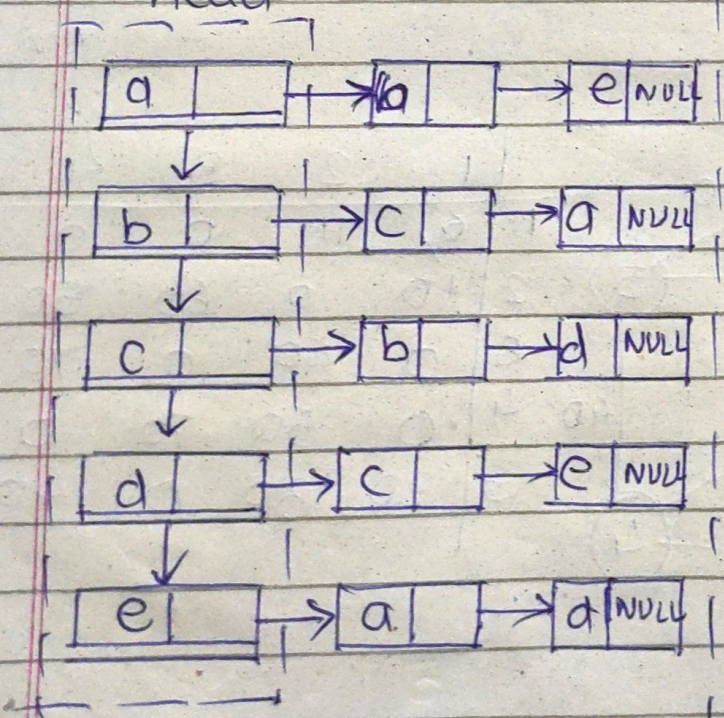
② Adjencency List :-

- In this represensentation a linked list is used to reprent a graph
- Here are 2 methods which are used to display graph



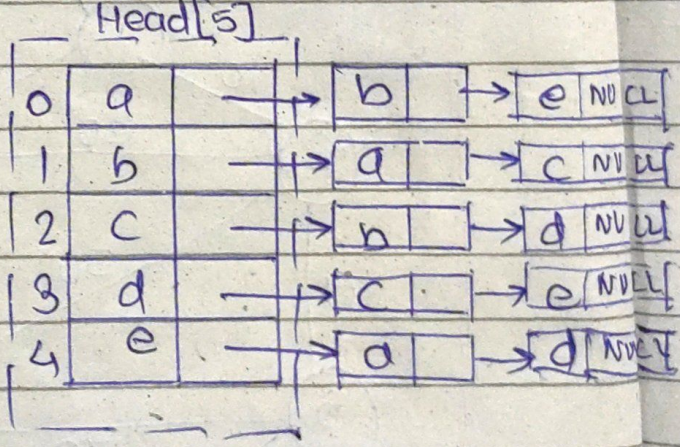
① method 1

- In this method linked list is only used to represent graph.

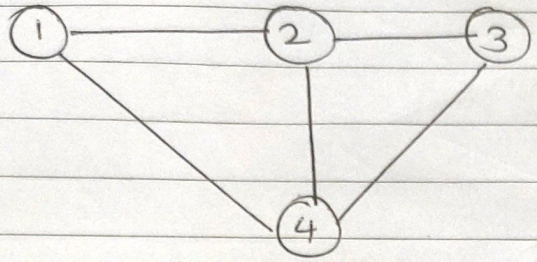


② method 2

- In this method we use array with linked list to represent graph.

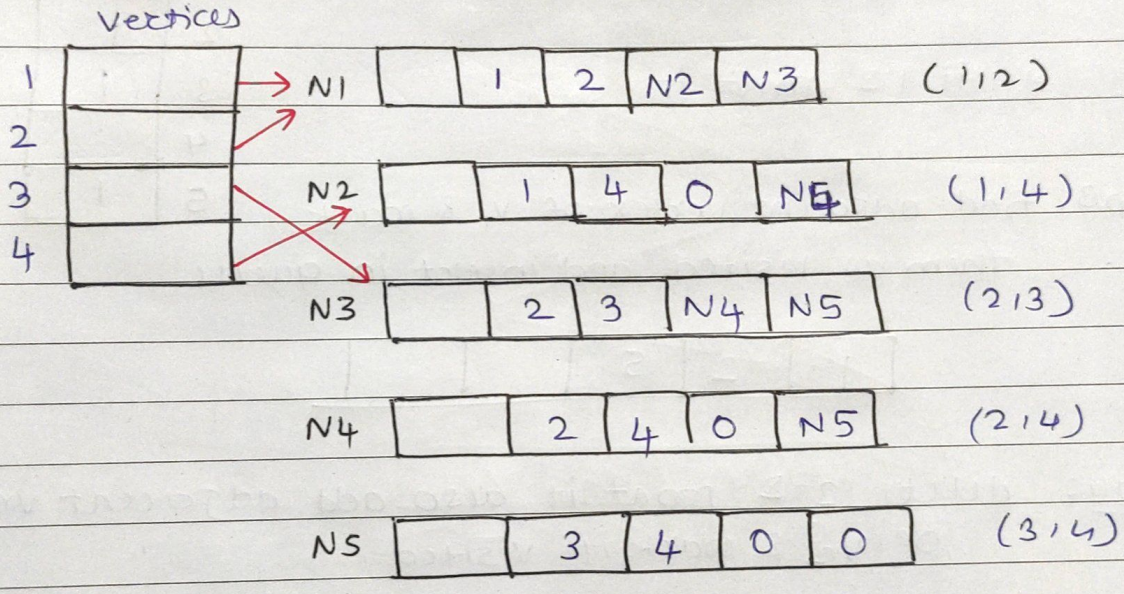


ADJACENCY MULTILIST



vertices :- 1, 2, 3, 4

edges :- (1,2) (1,4) (2,3) (2,4) (3,4)



Adjacency Multilist :-

Vertex 1 : N1 → N2 — doubt {In book its given N2}

Vertex 2 : N1 → N3 → N4

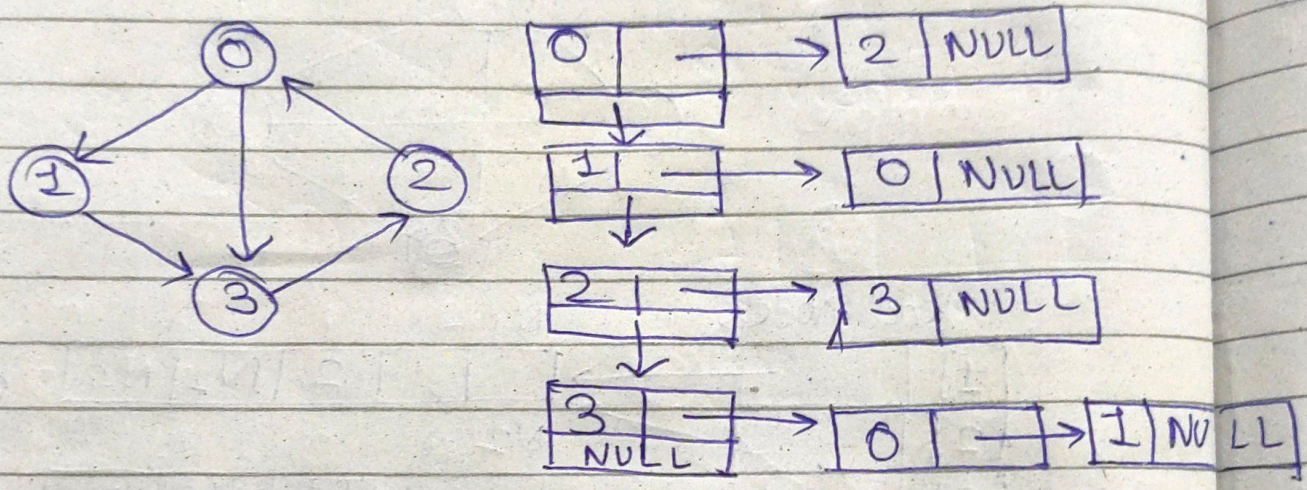
Vertex 3 : N3 → N5

Vertex 4 : N2 → N4 → N5

④ Inverse Adjecency List:

- Inverse Adjecency list is an adjencency list representation in which only incoming edge to a vertex is reprent For example:-

- Example :-



* Traversal :-

1. Breth First Search (BFS) :-
   - In BFS we strart from some vertex and find all the adjacent vertices of it This process will be repeated for all the vertices so that the vertices laying on same level (breath) get printed.
   - for avoiding repetitions of vertices we maintain array of visiting nodes
   - A queue data structure is used to store adjacent vertices.
   - Time complexity :→ O(V+E)
     V = no of vertex , E = NO of edges.

LL

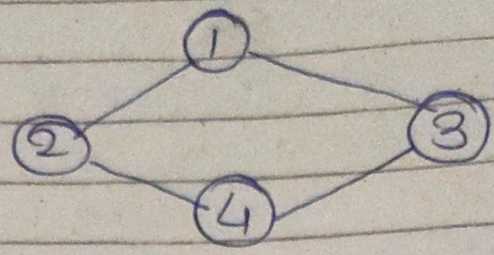
* Algorithm :-

Step Input : Graph
Output : BFS Travesal.

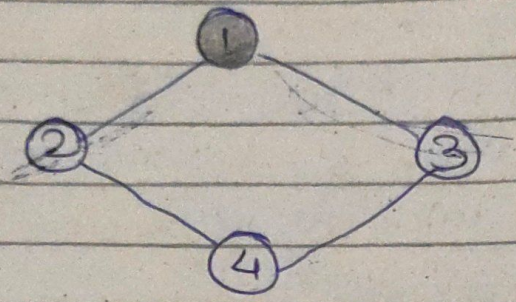
Step 1 : Start
Step 2 : Create a graph Depending on the type of graph (directed or undirected)

Step 3 : Read the vertex from which you want to travel a graph as V_i
Step 4 : Initialize the visited Array to 1 at index of V_i
Step 5 : Inserted visited vertex V_i in queue.
Step 6 : visit the vertex which is at the front of the queue store in Result Delete it from queue and place its adjacent nodes in the queue.
Step 7 : Repeat the step 6 till the queue is not empty.
Stop 8 : stop.

* Example :- BFS



(1)



0	1	0	0	0
0	1	2	3	4

visited array

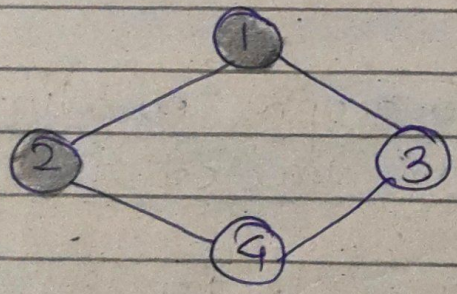
0	1	2	3	4
~~1~~				

queue.

front
Rear

(2) - insert vertex 1 in queue and marked the index 1 of visited array by 1, 1 get printed.
- Delete 1 and print it so 1 get printed.
-

(2)



Find Adjecent vertex of 1 and mark 1 in visited Array. insert those in queue.

0	1	1	1	
0	1	2	3	4

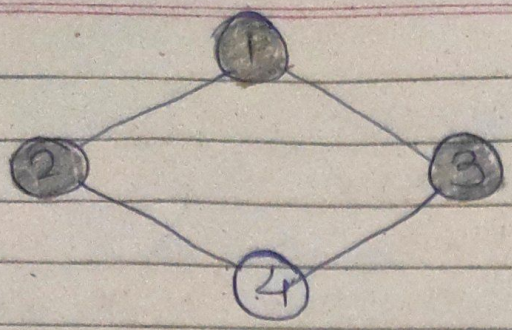
visited Array.

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

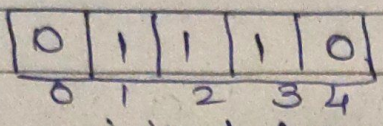
front

- 2 get printed
- 2 get deleted.

③

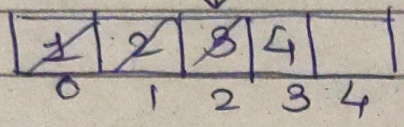


Find ajecent of 2 insert in queue.



visited Array

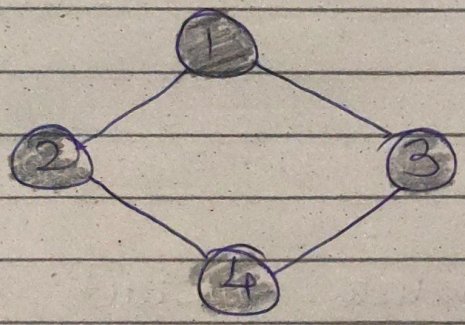
front.

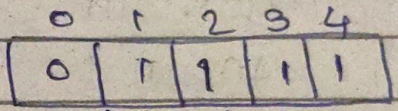


Queue.

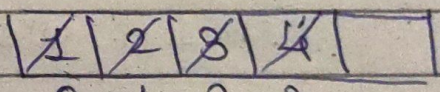
- front element 3 get printed
- 3 is deleted.
- 3's adjecent node get inserted in queue.

④





visited Array



front

queue.

- front get printed
- front element get deleted

Result : 1, 2, 3, 4.

② Depth first search (DFS):-

- In depth first search we travel from one vertex and travels the path as deeply as we can go when there is no vertex further we traverse back and search for unvisited vertex. ↑start
- An Array is maintained of visited vertex
- In DFS we use stack data structure

* Algorithm :-

Input :

output :

Step 1 : start

step 2: ~~push~~ Initialize a stack and visited Array push the vertex from which the serching should be started into the Array.

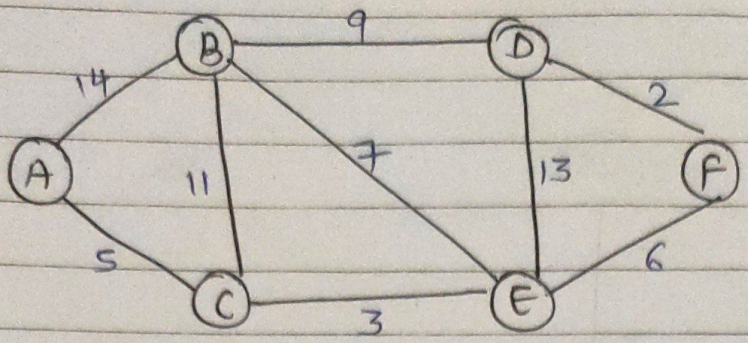
step 3: pop the stack, print popped vertex if it is not visited then mark it as visited.

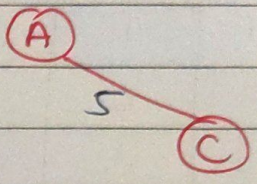
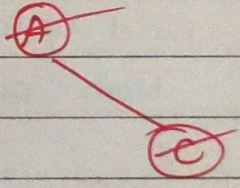
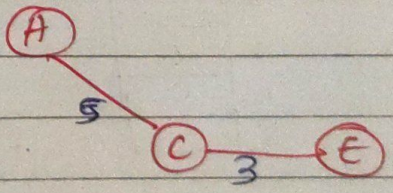
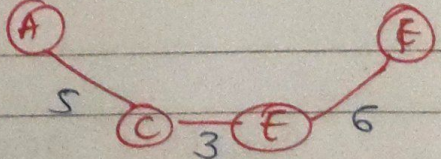
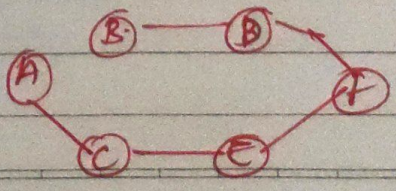
step 4: then find adjacent vertices to the current vertex and push them onto the stack.

step 5: Repeat step (ii) and (iii) if stack is not empty

step 6: stop.

DIJKSTRA'S ALGORITHM



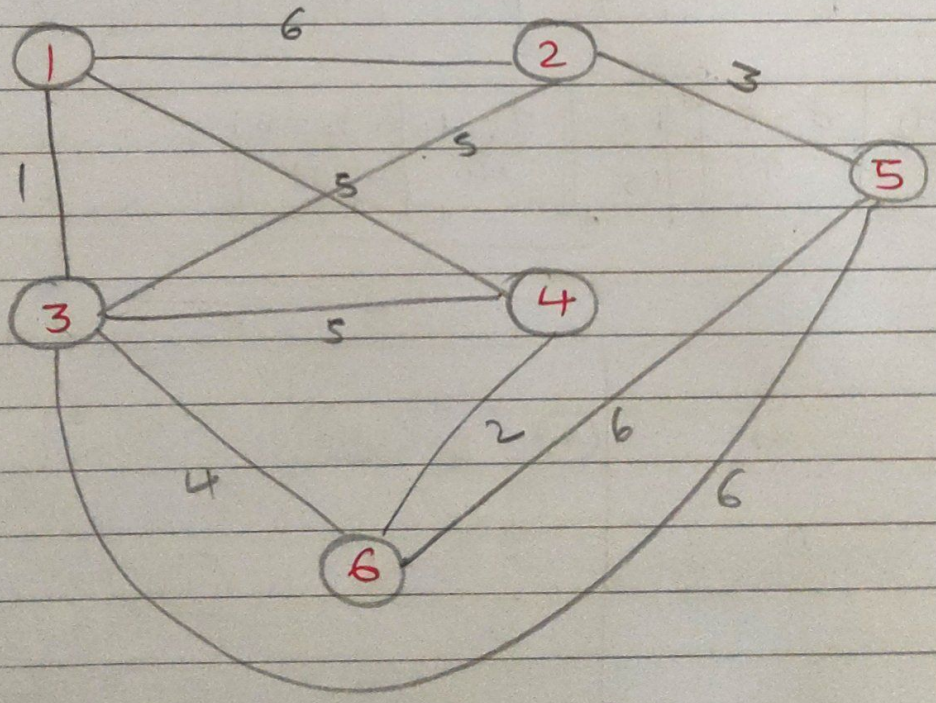
Source Vertex	Distance with other vertex	Path.
A	A - B = 14	
	A - C = 5	
	A - D = ∞	
	A - E = ∞	
	A - F = ∞	
~~B~~	~~B - C = 11 + 5 = 16~~	
	~~B - D = 5 + 9 = 14~~	
	~~B - E = 5 + 7 = 12~~	
	B - F = 5 + ∞ = ∞	
C	C - B = 5 + 11 = 16	
	C - D = 5 + ∞ = ∞	
	C - E = 5 + 3 = 8	
	C - F = 5 + ∞ = ∞	
E	E - B = 8 + 7 = 15	
	E - D = 8 + 13 = 21	
	E - F = 8 + 6 = 14	
F	F - B = 14 + ∞ = ∞	
	F - D = 14 + 2 = 16	
D	D - B = 16 + 9 = 25	

3. GRAPHS

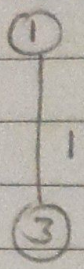
① Consider the graph represented by the following adjacency algorithm :

	1	2	3	4	5	6
1	0	6	1	5	0	0
2	6	0	5	0	3	0
3	1	5	0	5	6	4
4	5	0	5	0	0	2
5	0	3	6	0	0	6
6	0	0	4	2	6	0

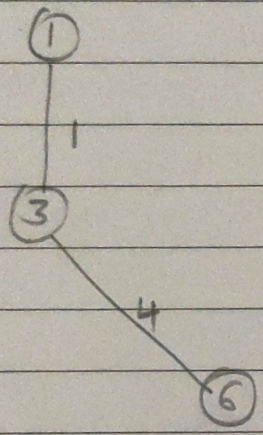
→ find minimum spanning tree of this graph using prim's Algorithms



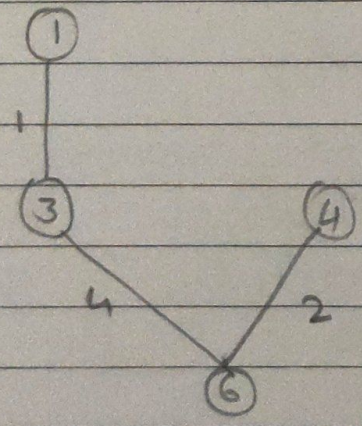
step 1 :



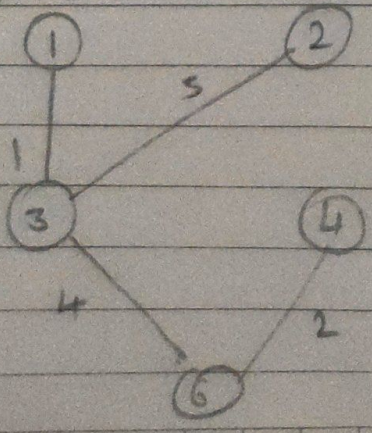
step 2 :



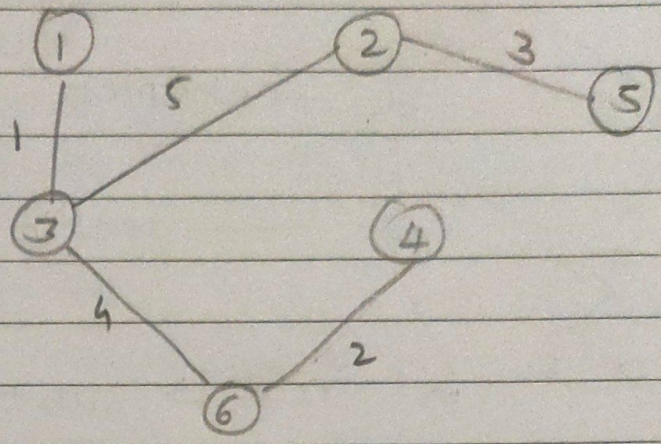
step 3 :



step 4 :



step 5 :

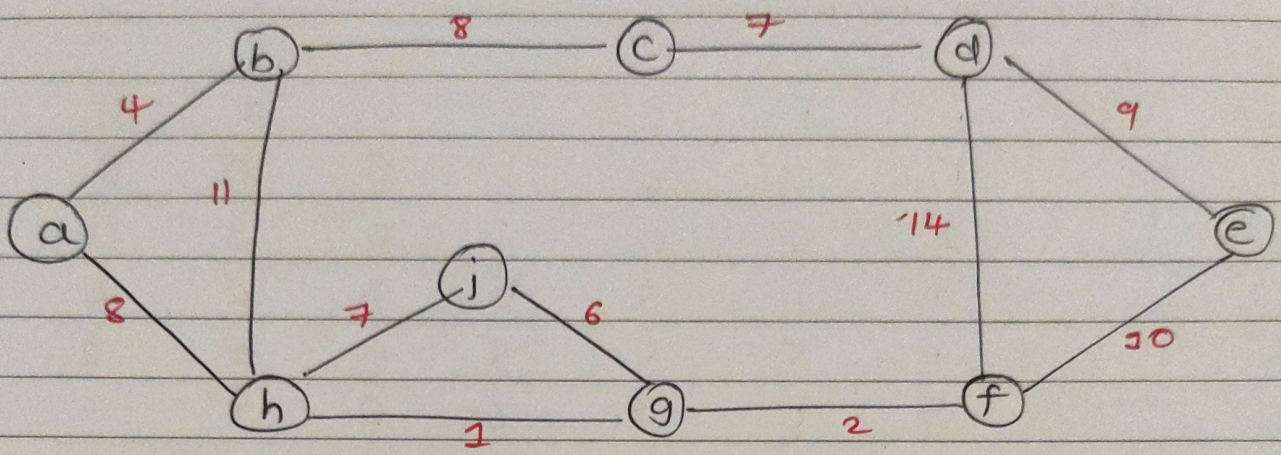


MST = 1 + 4 + 2 + 5 + 3

= 15

acyclic always

(2) minimum spanning tree of following graph using Kruskal's algorithm.



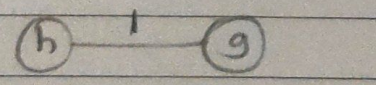
→ step 1: Ascending order arrangement of edges }

h to g	g to f	a-b	i-g	c-d	h-i	a-h	b-c
1	2	4	6	7	7	8	8

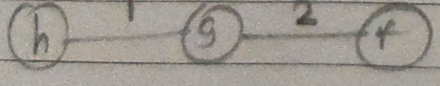
~~a-h~~	d-e	~~b-h~~	f-e	d-f	b-h
~~8~~	9		10	14	~~11~~

step 2;

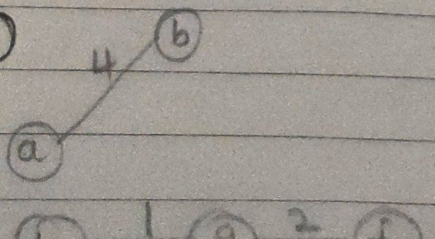
(1)



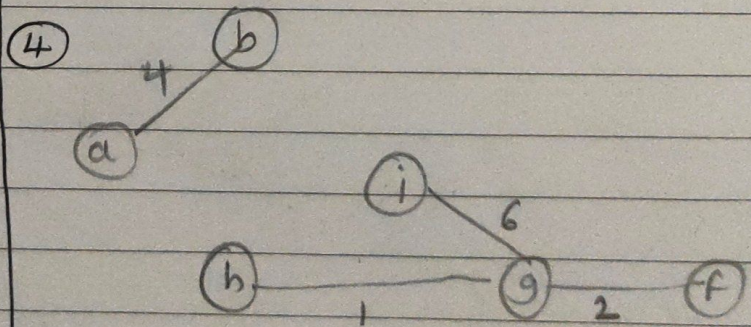
(2)



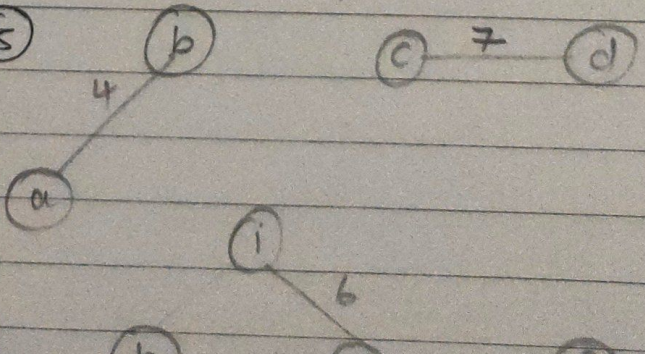
(3)

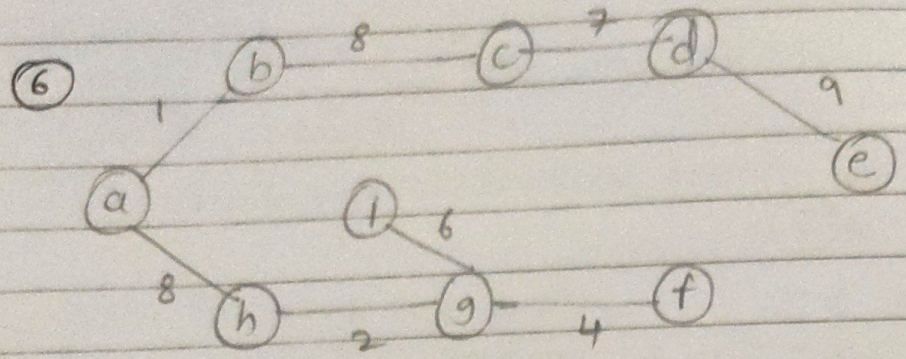


(4)



(5)





cost of minimum spaning tree value is:

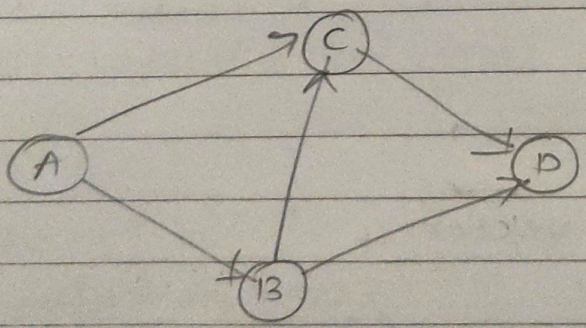
$= 1+8+8+2+6+4+7+9$

$= 45$

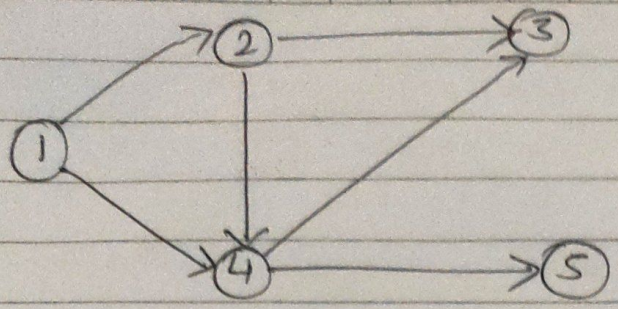
(3) Write a short Note on Topological sorting.

→

A topological sorting is an ordering of vertices in a directed a cyclic graph, such that if there is a path from Vi to Vj then Vj appears after Vi

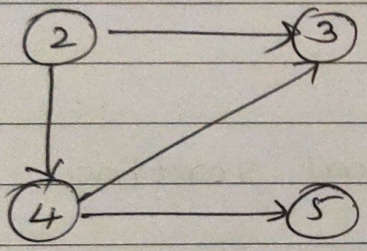


A simple algorithm to find a topological ordering is first to find any vertex, indigree = 0. Print this vertex, and remove it along with edges from graph. Then apply the same strategy to rest of the graph.



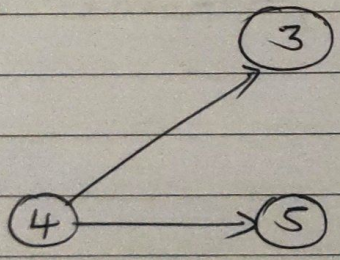
	1	2	3	4	5
Indegree	0	1	2	2	1

Step 1: after deleting vertex 1



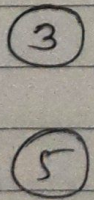
	2	3	4	5
id	0	2	1	1

Step 2: After deleting vertex 2



	3	4	5
id	1	0	1

Step 3: After deleting vertex 4



	3	5
id	0	0

Topological ordering = 1, 2, 4, 3, 5 or
1, 2 4, 5, 3