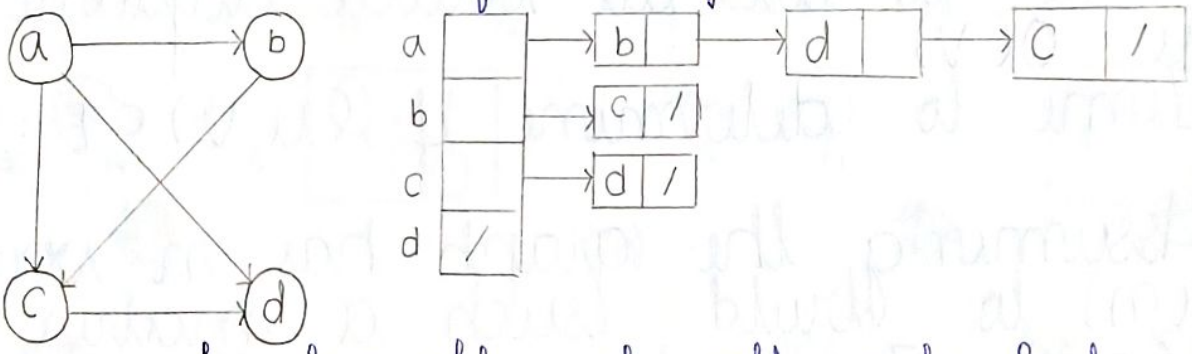
1. Given graph G in adjacency matrix representation. Write a pseudocode to find the adjacency list representation. Discuss its time complexity.

Pseudocode

1. Create an array A of size N.
2. Type of array must be list of vertices.
3. Initially each list is empty so each array element is initialised with empty list.
4. Iterate each given edge of the form (u, v).
5. Append v to the u^{th} list of array A.
6. If graph is undirected append u to the v^{th} list of array A.



Sum of lengths of all adj lists is

$$\sum_{v \in V} \text{out-degree}(v) = |E|$$

Total storage : $O(V+E)$

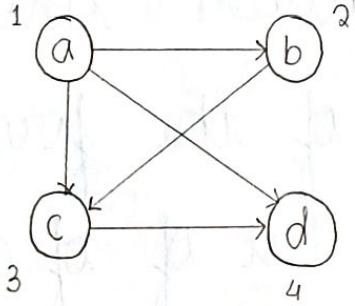
$T(n) = O(E)$

In worst case of complete graph time & space complexity becomes $O(v^2)$.

2. Given a graph G in adjacency list representation. Write a pseudocode to find adjacency matrix representation. Discuss its time complexity.

Pseudocode

1. Create a matrix A of size N×N and initialise it with zero.
2. Iterate over each given edge of form (u, v) and assign 1 to A[u][v]
3. If graph is undirected then assign 1 to A[v][u].

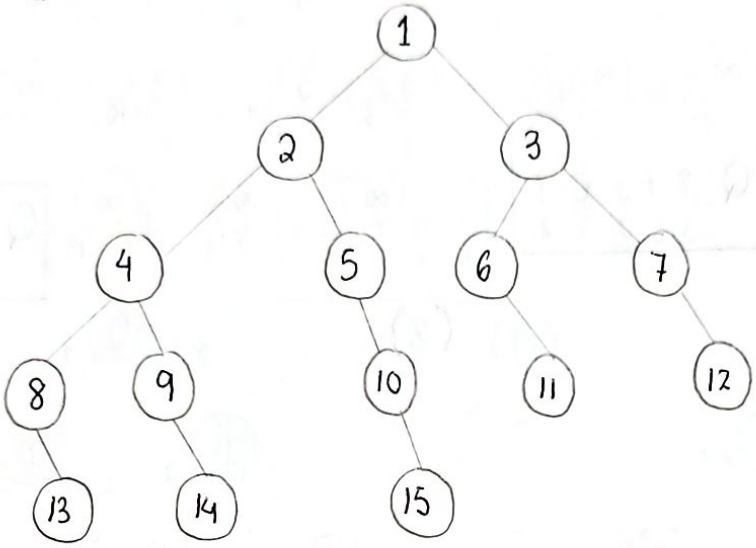


	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

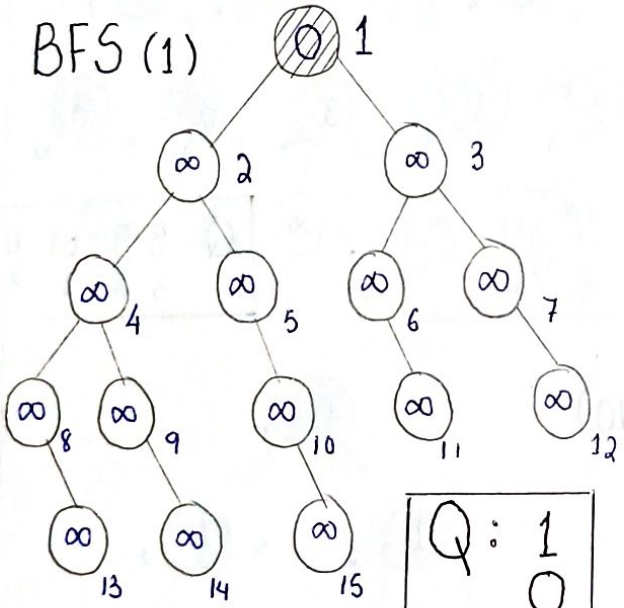
Space : $O(V^2)$
Time : to list all vertices adjacent to u : $O(V)$
Time to determine if $(u, v) \in E$: $O(1)$

Assuming the graph has n vertices, $T(n)$ to build such a matrix is $O(n^2)$. To build adjacency matrix we need to create a square $n \times n$ matrix and fill its values with 0 and 1. It costs $O(n^2)$ space.

3. Assume both BFS and DFS algorith will choose node with lesser index first. Apply BFS and DFS on G starting from node 1.

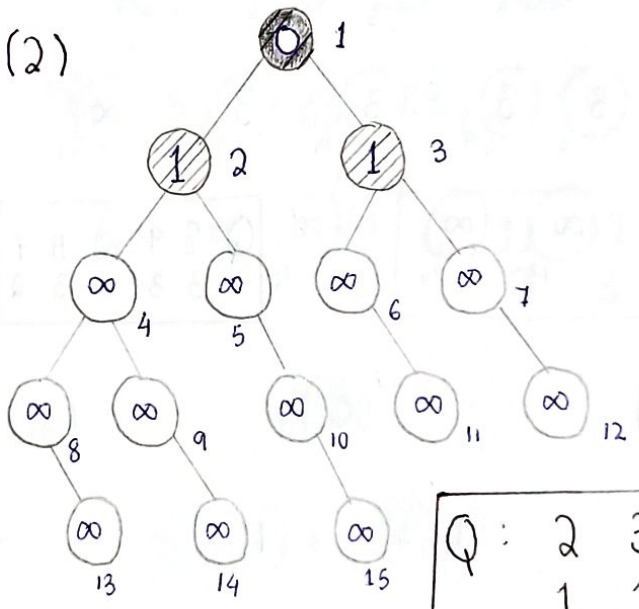


BFS (1)



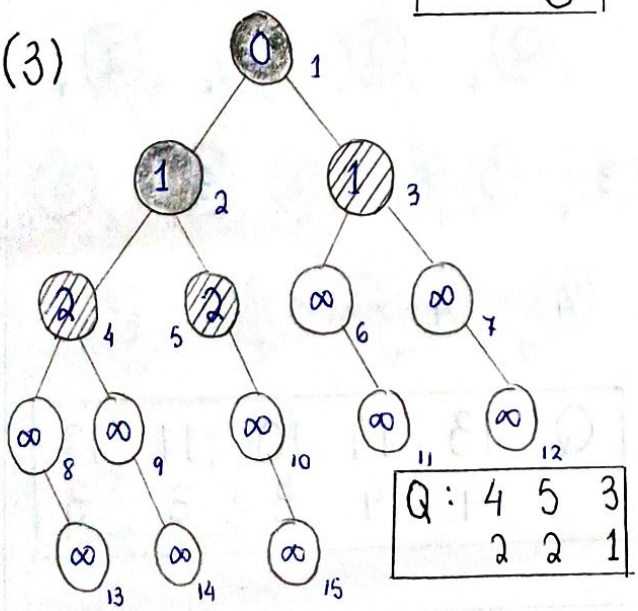
Q:	1
	0

(2)



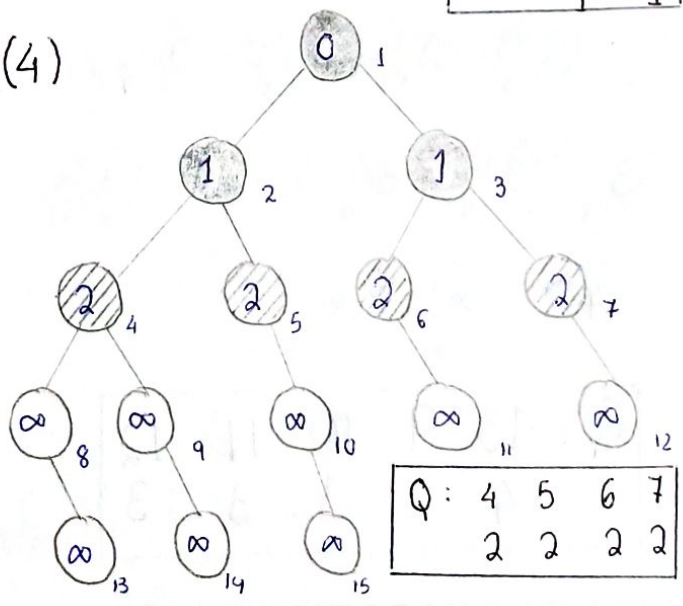
Q:	2	3
	1	1

(3)



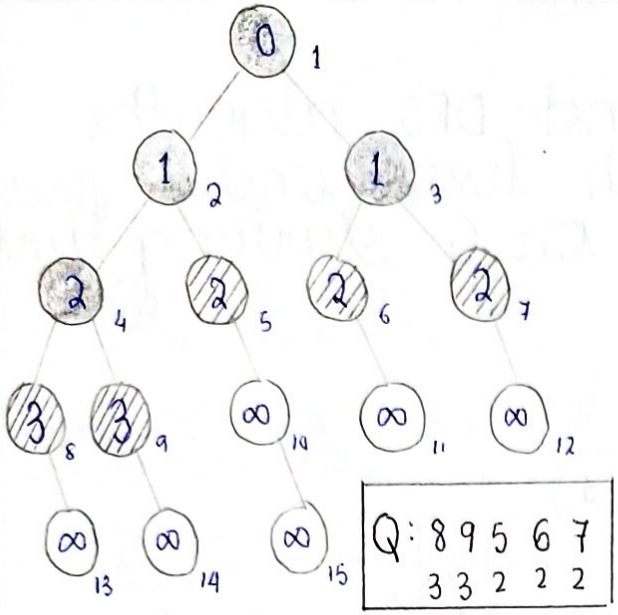
Q:	4	5	3
	2	2	1

(4)

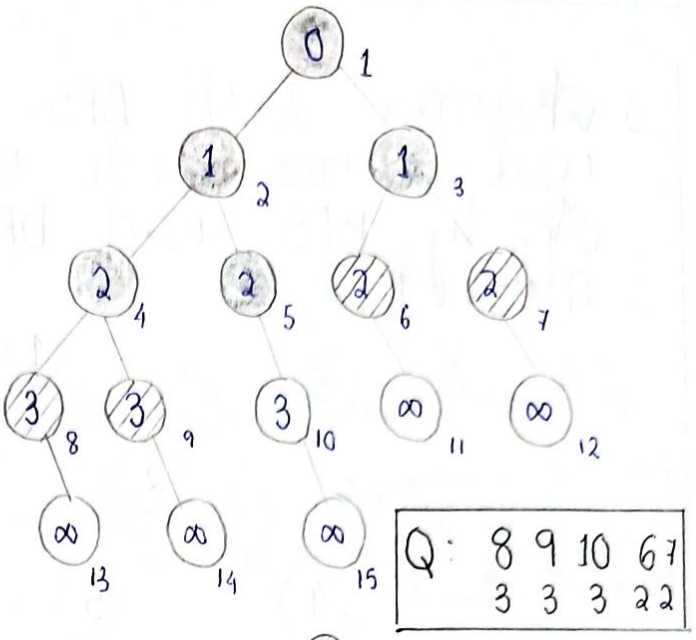


Q:	4	5	6	7
	2	2	2	2

(5)

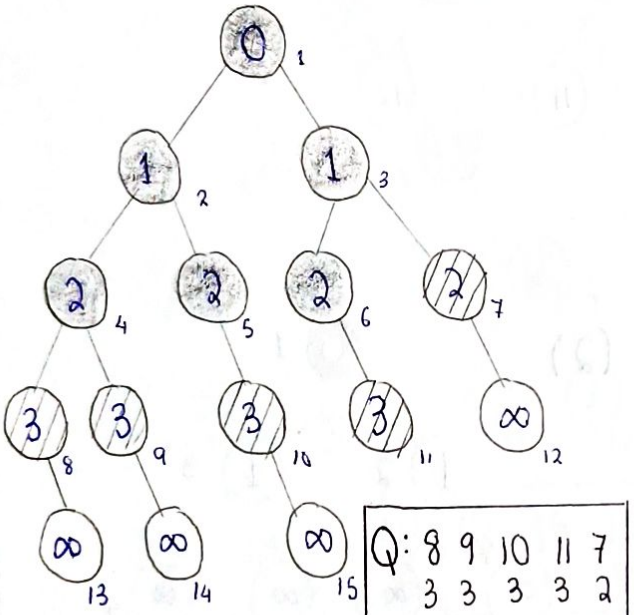


Q:	8	9	5	6	7
	3	3	2	2	2



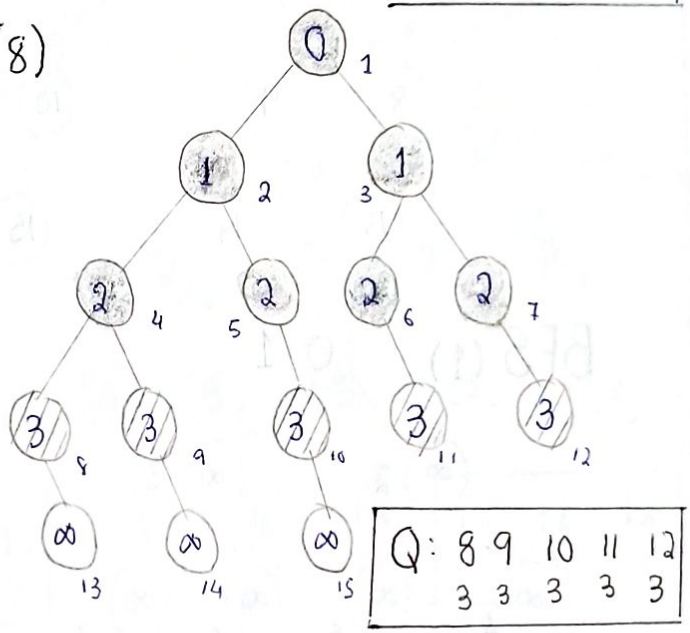
Q:	8	9	10	6	7
	3	3	3	2	2

(7)



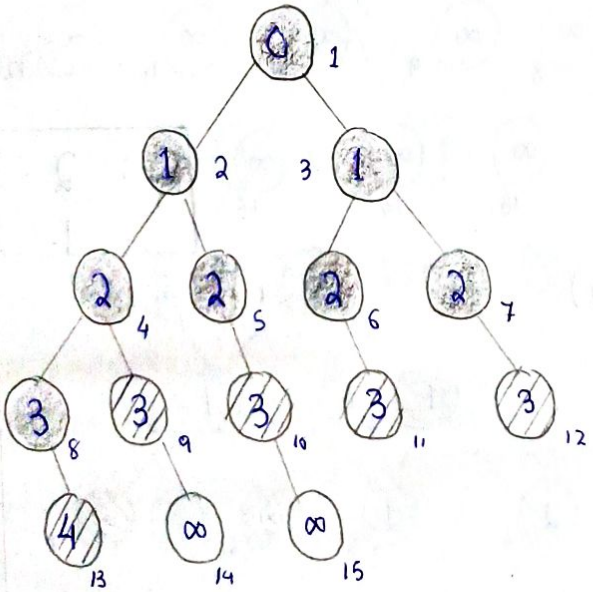
Q:	8	9	10	11	7
	3	3	3	3	2

(8)



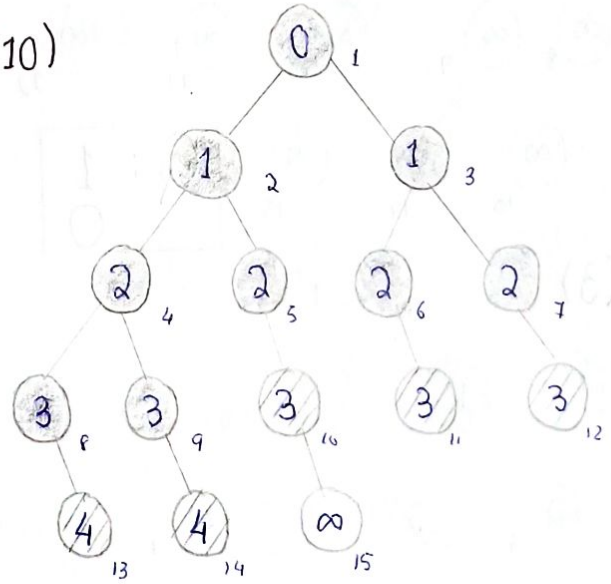
Q:	8	9	10	11	12
	3	3	3	3	3

(9)



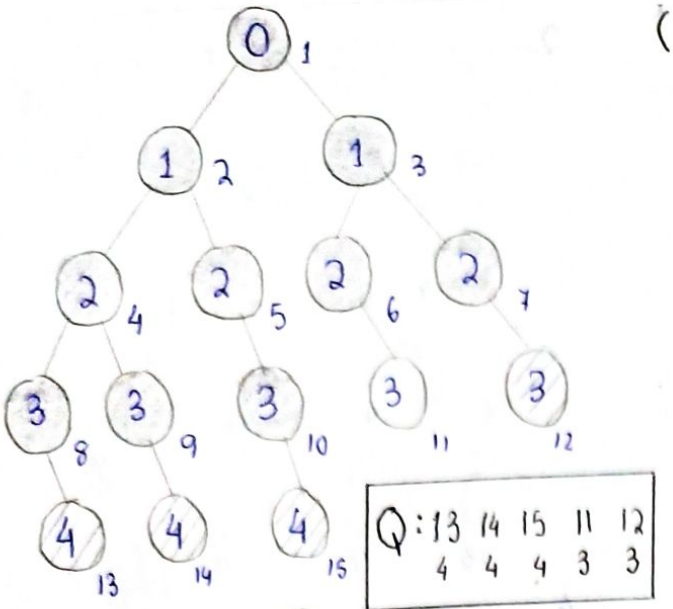
Q:	13	9	10	11	12
	4	3	3	3	3

(10)

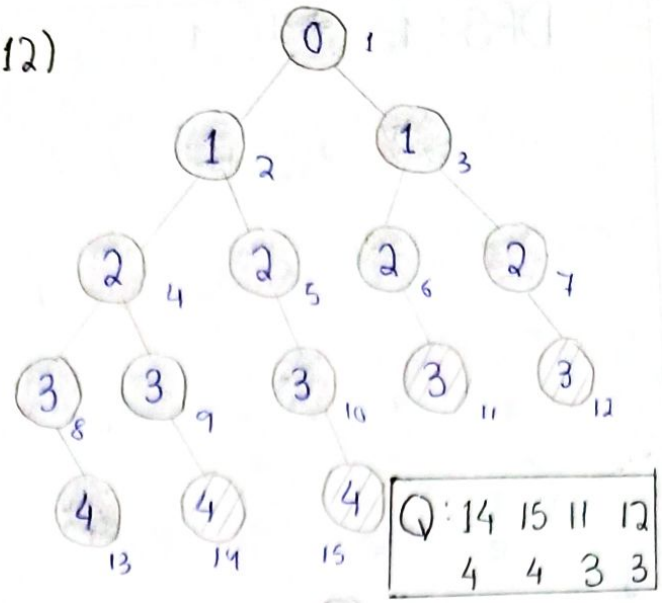


Q:	13	14	10	11	12
	4	4	3	3	3

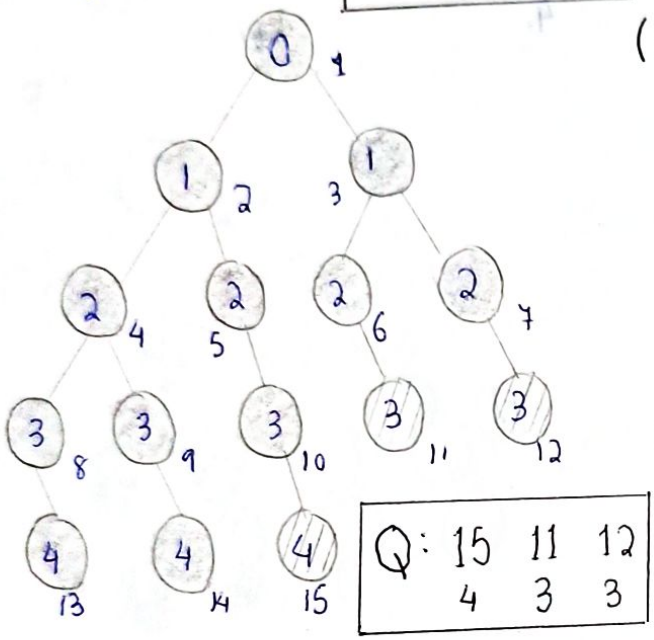
(11)



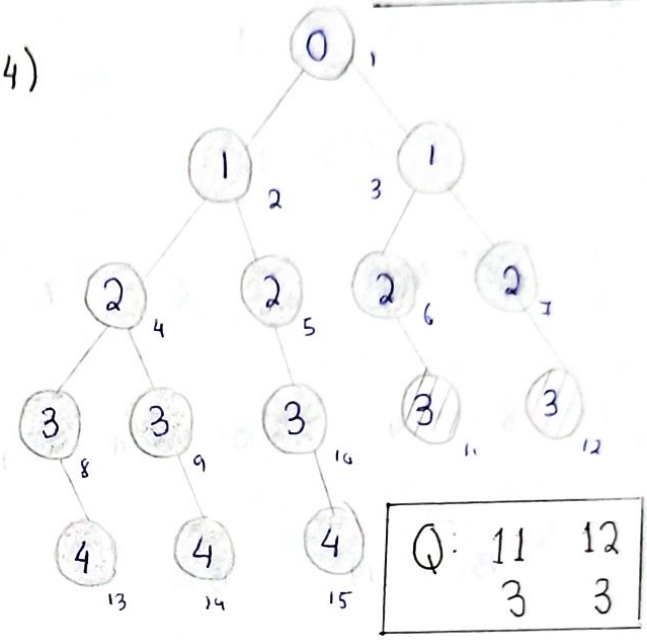
(12)



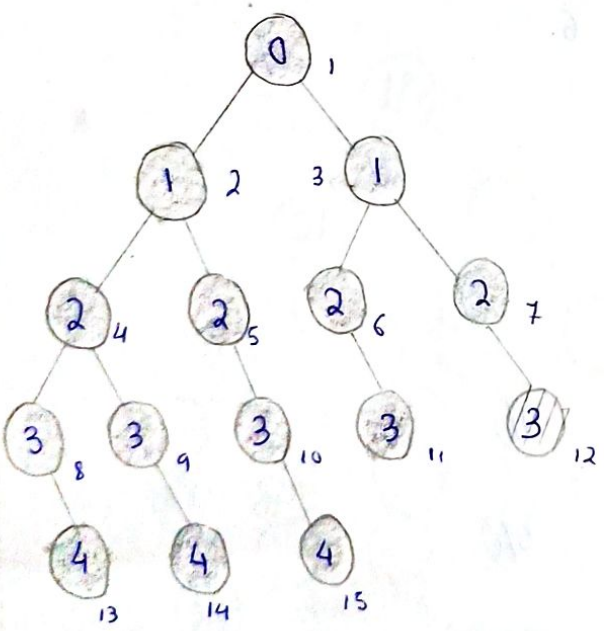
(13)



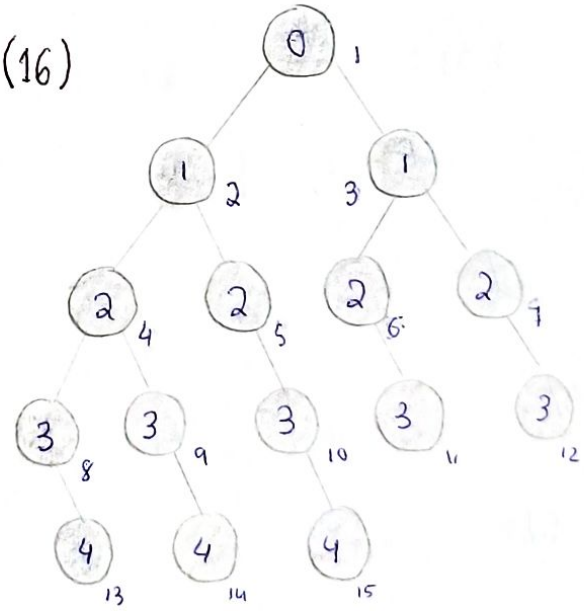
(14)



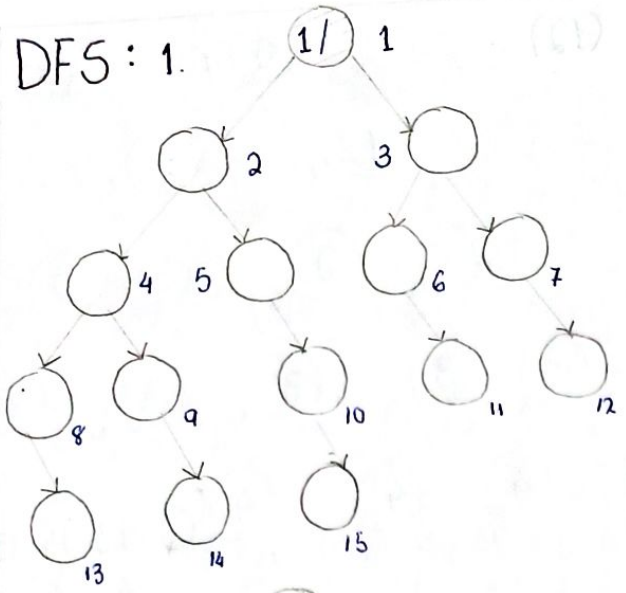
(15)



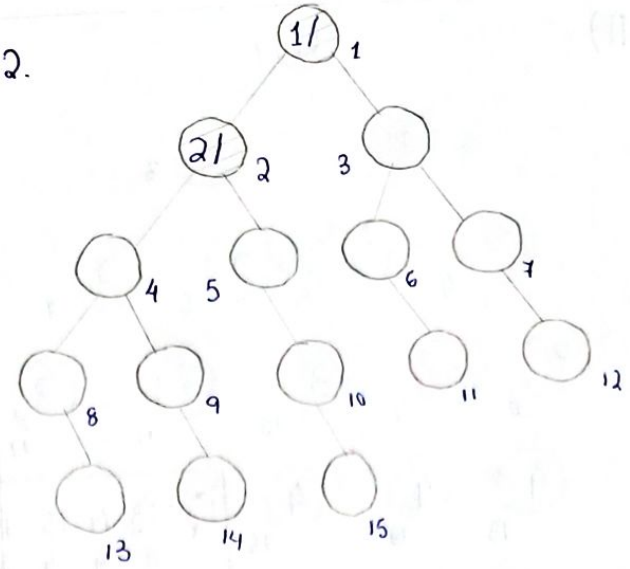
(16)



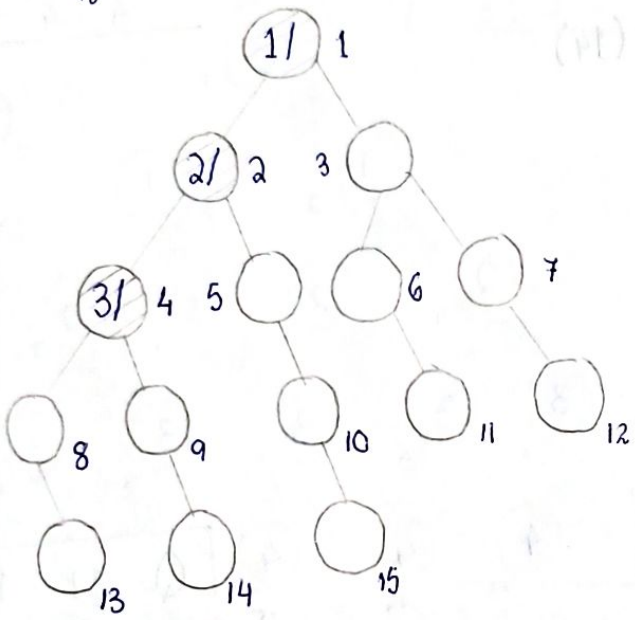
DFS : 1.


1/ 1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

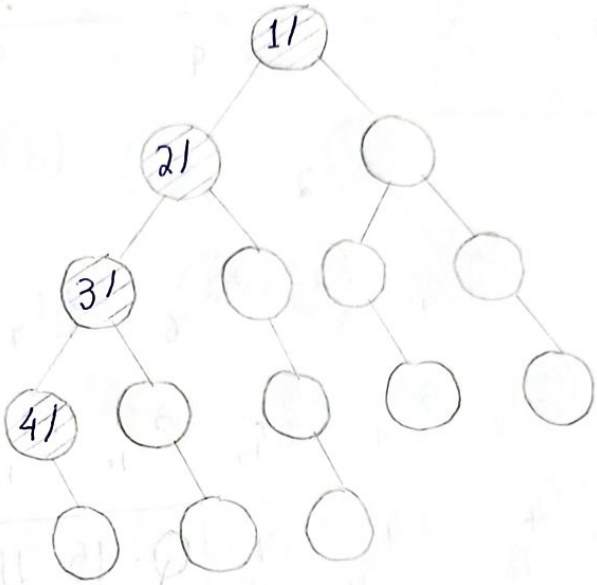
2.


1/ 1
2/ 2
3
4
5
6
7
8
9
10
11
12
13
14
15

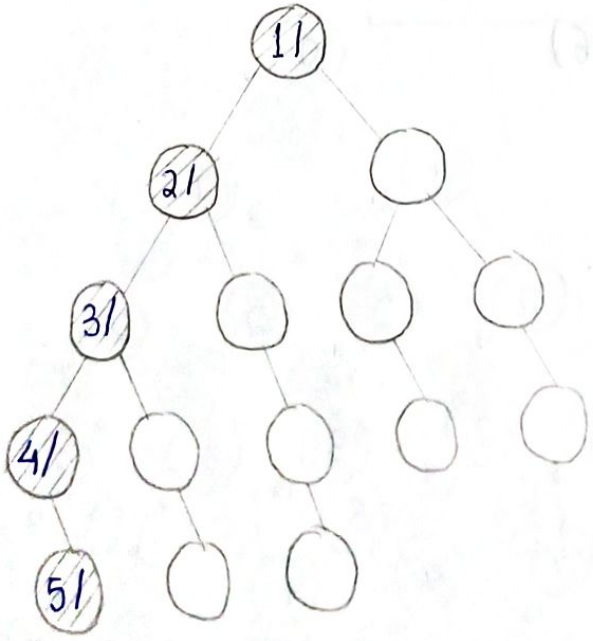
3.


1/ 1
2/ 2
3
3/ 4
5
6
7
8
9
10
11
12
13
14
15

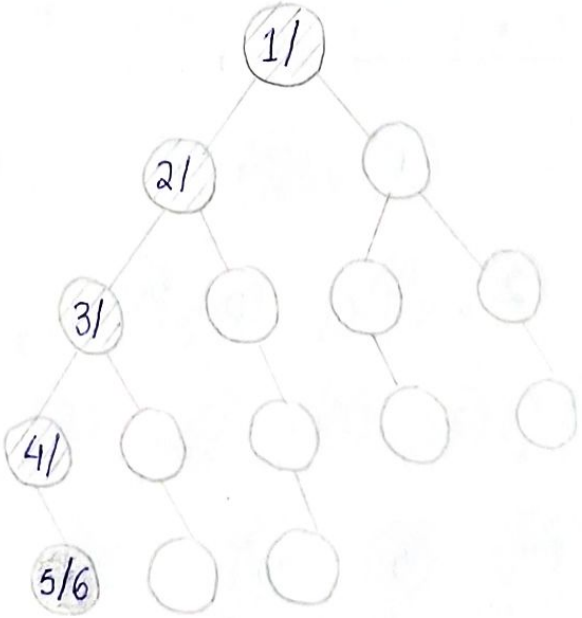
4.


1/
2/
3/
4/

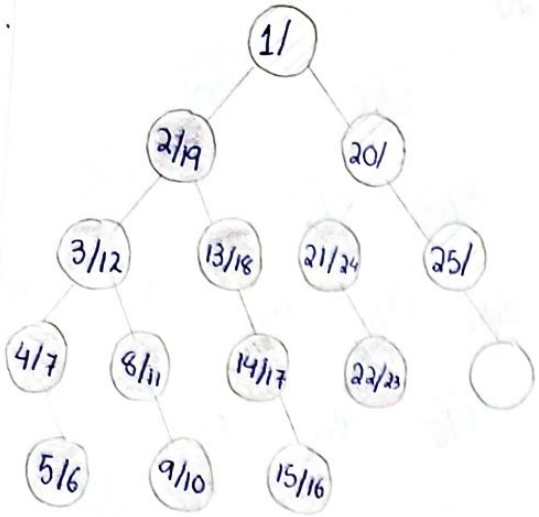
5.


1/
2/
3/
4/
5/

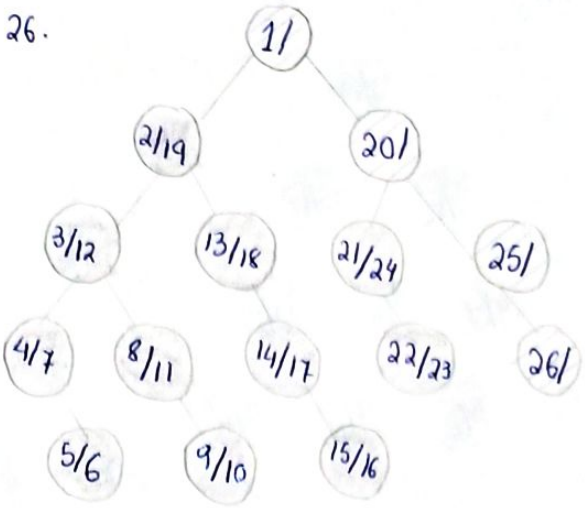
6.


1/
2/
3/
4/
5/6

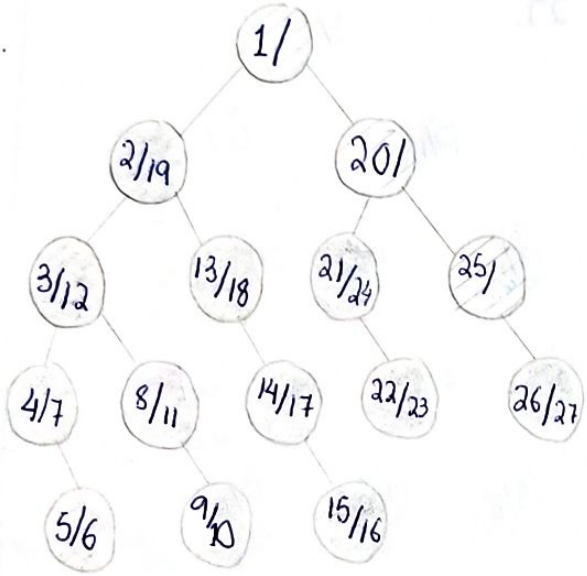
25.



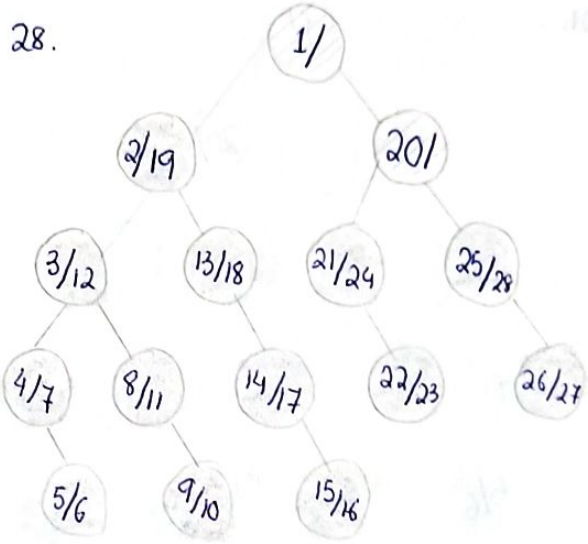
26.



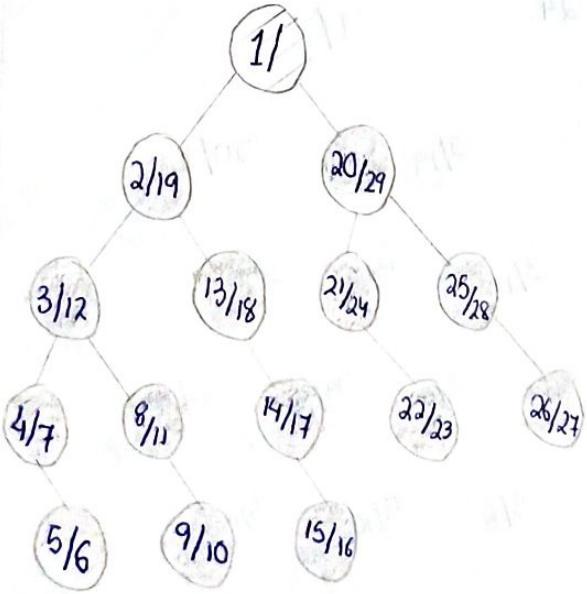
27.



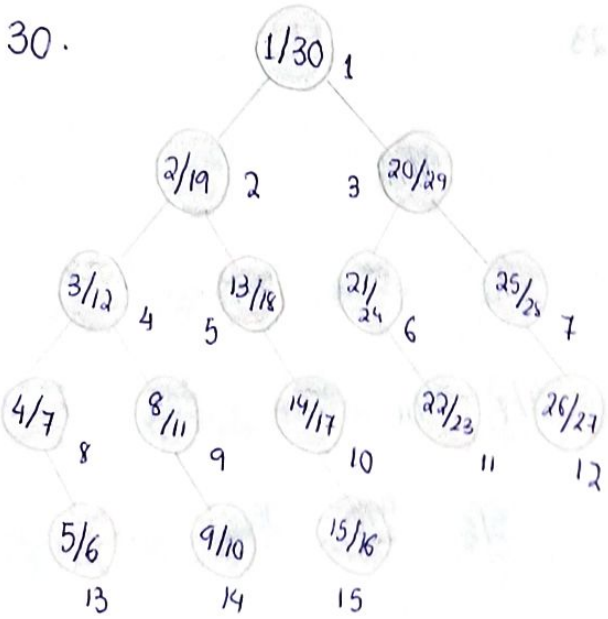
28.



29.



30.



(a) Write sequence in which nodes will be visited in complete traversal of G

BFS
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

DFS
1, 2, 4, 8, 13, 13, 8, 9, 14, 14, 9, 4, 5, 10, 15, 15, 10, 5, 2, 3, 6, 11, 11, 6, 7, 12, 12, 7, 3, 1

(b) What is the max size of queue in BFS and of stack in DFS?

Size of queue in BFS : 5

Stack in DFS : 5

(c) Which algorithm is preferred if node to be searched is 6?

BFS

BFS traversal is the ideal choice for vertices that need to be located close to source vertex.

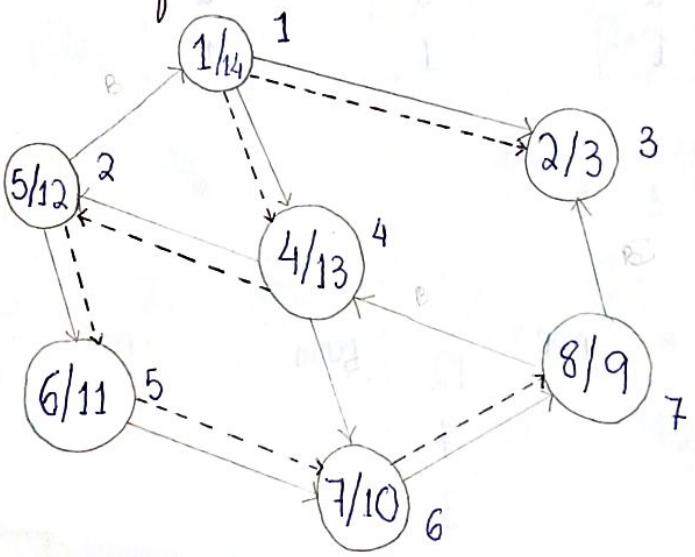
(d) Which algorithm is preferred if node to be searched is 14?

DFS

DFS traversal is optimum method when solutions are found away from source vertex.

4. If there is ever a choice amongst multiple nodes, traversal algorithms will choose node with lesser index first.

(i) Find depth first tree.



(ii) Identify tree edge, back edge, forward edge, cross edge.

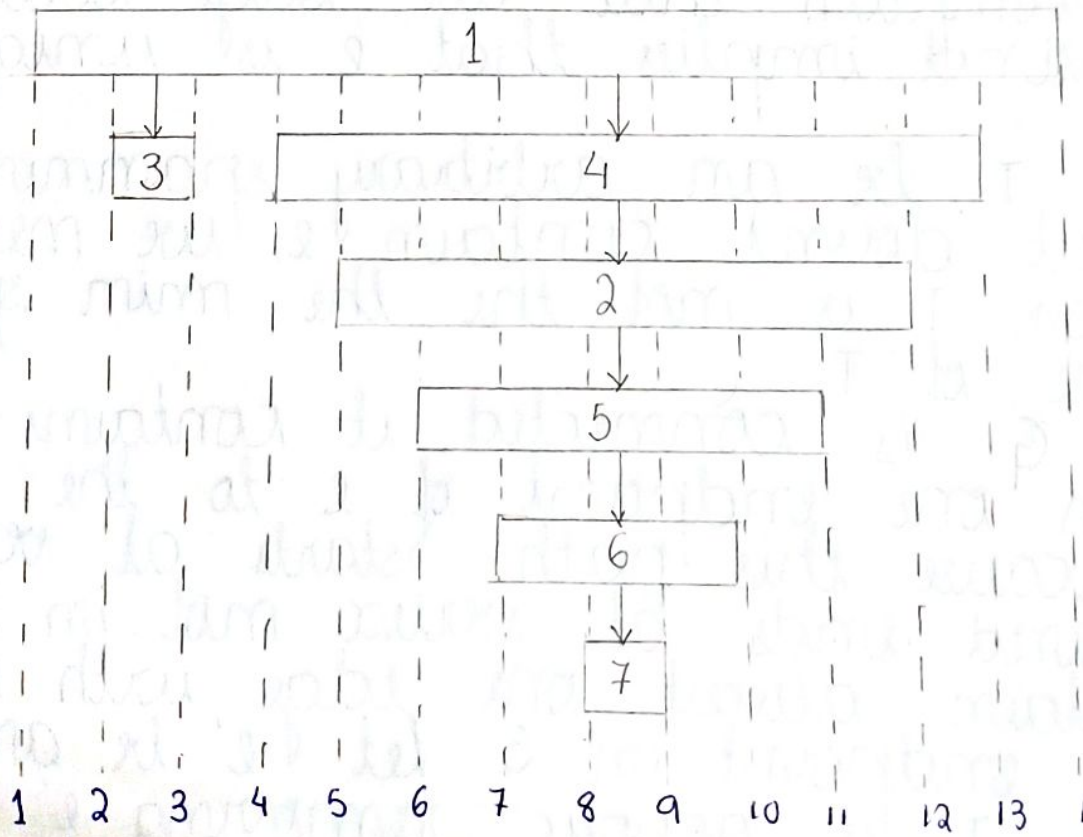
Tree - edge : (1,3), (1,4), (4,2), (2,5), (5,6), (6,7)

Back - edge : (2,1), (7,4)

Forward edge : (4,6)

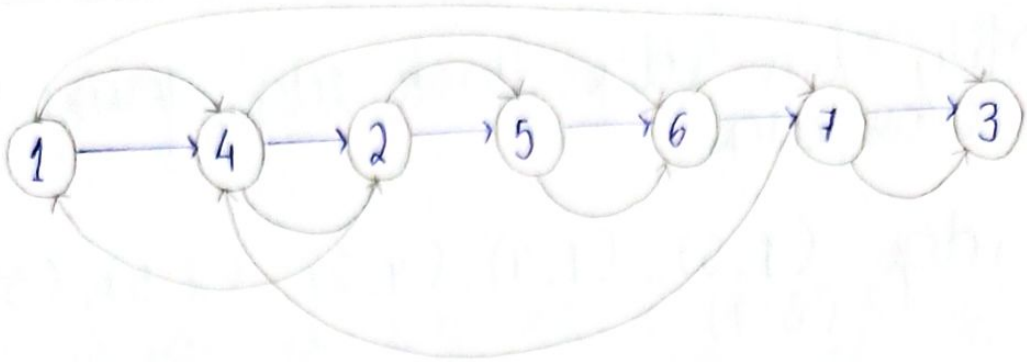
Cross edge : (7,3)

(iii) Find parenthesis structure of complete traversal.



(1 (3 3) (4 (2 (5 (6 (7 7) 6) 5) 2) 4) 1)

(iv) Write nodes of graph in topologically sorted order



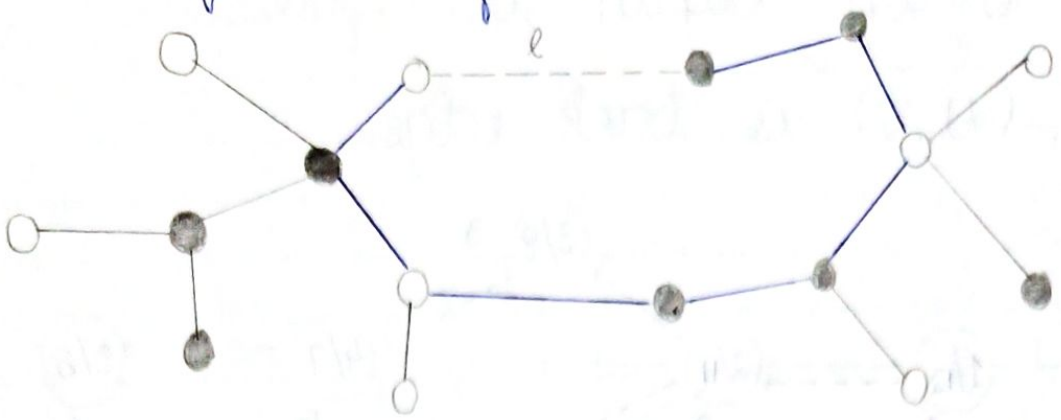
5. Prove that minimum weight edge in graph G with no duplicate edge weights must be present in every spanning tree of G.

Let S be an arbitrary subset of vertices of G and let e be the lightest edge with exactly one edge in S. (Our assumption that all edge weights are distinct implies that e is unique).

Let T be an arbitrary spanning tree that doesn't contain e; we need to prove T is not the the min spanning tree of T.

As T is connected, it contains a path from one endpoint of e to the other. Because this path starts at vertex of S and ends at vertex not in S it must contain atleast one edge with exactly one endpoint in S. Let e' be any such edge. T is acyclic, removing e' from T yields spanning forest with exactly two components, one containing each endpoint of e. Adding e to this forest gives new spanning tree $T' = T - e' + e$. Definition of e implies $w(e') > w(e) \Rightarrow T'$ has smaller weight

than T. So, T is not the minimum spanning tree of G

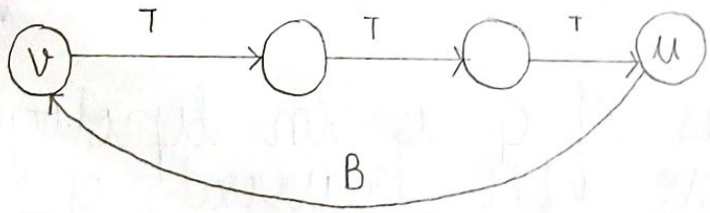


6. Prove that directed graph G is a DAG iff DFS traversal of G has no back edge. Perform DFS on given graph & identify types of edges. Check if it's DAG or not.

⇒ Show that back edge ⇒ cycle

Suppose there is a back edge (u, v). Then v is ancestor of u in depth-first forest.

So, there is a path $v \rightsquigarrow u$. So $v \rightsquigarrow u \rightsquigarrow v$ is a cycle

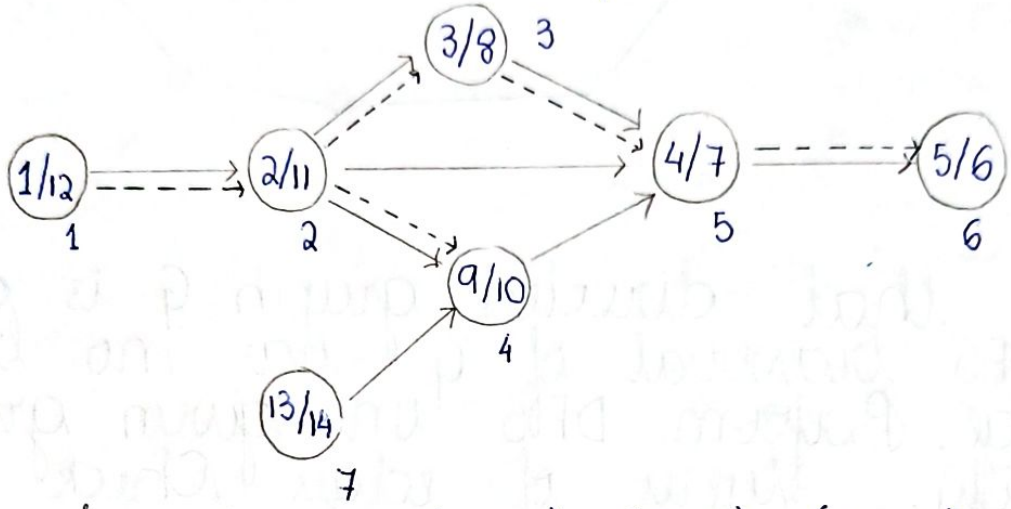


⇐ Show that a cycle implies back edge.

C: cycle in G ; v: first vertex discovered in C, (u, v): preceding edge in C.
At time d[v], vertices of C form white path $v \rightsquigarrow u$.

By white-path theorem, u is descendent of v in depth-first forest.

So, (u, v) is back edge.



Tree edge: (1, 2), (2, 3), (3, 5), (5, 6), (2, 4)

Back edge: (4, 5)

Forward edge: (2, 5)

Cross edge: (7, 4)

The graph G is not a DAG as it has a back-edge which forms a cycle.