1) Given a graph in adjacency matrix representation pseudocode to find the adjacency list.

create a 2-D array having n rows,
Traverse the given matrix and if matrix[i][j]=1 where i is the row number and j is the column number then add j in the ith row of the 2-D array.

```
matrix-to list (int n, int matrix[][] ) {
    for (int i=0; i<n; i++) {
        for (int j=0 ; j<n ; j++) {
            if (matrix[i][j]==1) {
                adjlist[i].add(j);
    } } }
    return adjlist;
```

Time complexity = $O(n^2)$

2. Given a graph G in adjacency list representation.
Pseudo to find the adjacency matrix.

Adjacency matrix ⇒ where matrix[i][j] = 1 if there is an edge between i and j, and 0 otherwise

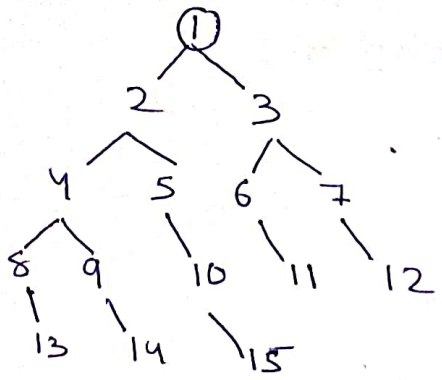
- create a matrix of size n x n and initialize it with zero.
- Traverse the given adjacency list and for every value j in a row i change matrix[i][j] to 1

```
List to matrix (int n, ADJ List [] ) {
    for( int i=0; i<n; i++) {
        for( int j=0; j < adjList[i].size(); j++)
            matrix[i] [adjList[i].get(j)] = 1
    }
}
return matrix;
}
```

Time complexity = O(V+E)

V → Vertex, E = edge

(8)



BFS

S-1 | 1 |

S-2 | 2 | 3 | deque 1

S-3 | 3 | 4 | 5 | deque 2

S-4 | 4 | 5 | 6 | 7 | deque-3

S-5 | 5 | 6 | 7 | 8 | 9 | deque-4

S-6 | 6 | 7 | 8 | 9 | 10 | deque-5

S-7 | 7 | 8 | 9 | 10 | 11 | deque-6

S-8 | 8 | 9 | 10 | 11 | 12 | deque-7

S-9 | 9 | 10 | 11 | 12 | 13 | deque-8

Store the deleted element in an array.

Array

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 11 | 12 | 13 | 14 | 15 |

S-10 | 10 | 11 | 12 | 13 | 14 | deque-9

S-11 | 11 | 12 | 13 | 14 | 15 | deque-10

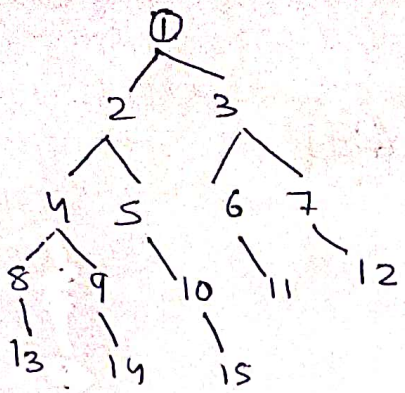
S-12 | 12 | 13 | 14 | 15 | deque-11

S-13 | 13 | 14 | 15 | deque-12

S-14 | 14 | 15 | deque-13

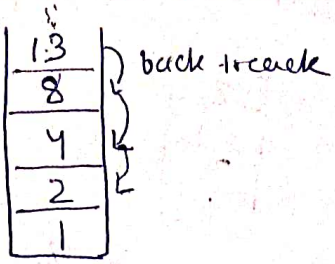
S=15 | 15 | deque-14

S-16 delete 15



Array

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



S-1.

V = 1
W = {2, 3}

S-2

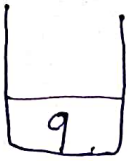
pop → 1
V = 2
W = {4, 5}

S-3

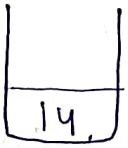
pop → 2
V = 4
W = {8, 9}

V = 8
W = {13}

V = 13
no child.
backtrack

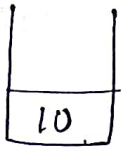


V = 9
W = {14}

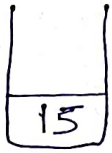


V = 14
no child
backtrack

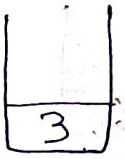
V = 5
W = {10}



V = 10
W = {15}



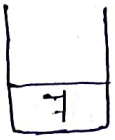
V = 15
no child
backtrack



V = 3
W = {6, 7}

V = 6
W = {11}

V = 11
no child
backtrack



V = 7
W = {12}

no child
pop-12.

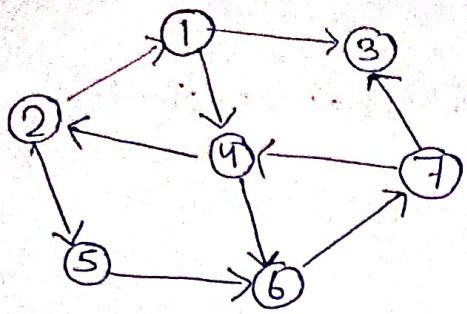
(ii) maximum size of Queue in BFS is 15

maximum size of Stack in DFS is 15

(iii) BFS is preferred if the node to be searched is the node 6.

(iv) DFS is preferred if the node to be searched is 14.

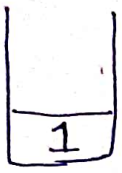
(4)



Array

1	3	4	2	5	6	7

S-1

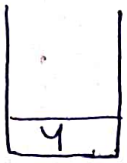


V=1
w={3,4}

S-2

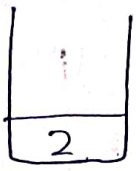
V=3
no child
backtrack

S-3



V=4
w={2,6}

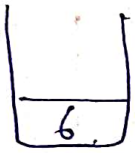
S-4



V=2
w={1,5}

S-5

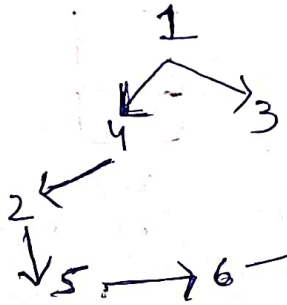
1 already visited
V=5
w={6}



V=6
w={7}

pop=7

(ii)



there are tree edge.

back edge => edge 7-3 is back edge.

↳ (u,v) such that v is ancestor but not a part of DFS tree

forward edge => 4-6 => edge => forward edge

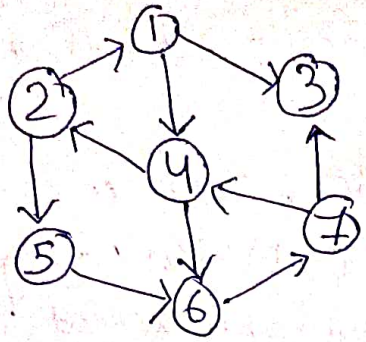
↳ (u,v) such that v is descendant but not a part of DFS tree.

cross-edge => edge 7-4 is a cross edge.

↳ Edge that connects two nodes such that they do not have any ancestor and a descendant relationship,

(iii) parenthesis structure => 1(4,4)1, 1(3 3)1)
4(2 2)4, 2(5 5)2
5(6 6)5, 6(7 7)6

(IV)



1 → [3 | 4]
2 → [1 | 5]
3 → \
4 → [2 | 6]
5 → [6 |]
6 → [7]
7 → [4 | 3]

S-1 Topological sort(3), visited(3), True
List is empty, no more recursive call.

Stack : [3]

S-2 Topological Sort(5), visited(5), True
Topological Sort(6), visited(6), True.

Topological sort(7), visited(7), True

Topological sort(4), visited(4) True

~~Topological Sort(3), visited(3) True~~ 3 already visited.

Topological sort(2) → visited. true

Topological sort(1) - visited(1) true.

Topological Sort(4) 3 and 4 already visited.

Topological Sort(3) → already visited.

Topological Sort(4)

Topological Sort(4).

2 already visited.

Stack : [3 | 5 | 6 | 7 | 4 | 2 | 1]

LIFO order → 1 - 2 - 4 - 7 - 6 - 5 - 3

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

Ans → Let G be a graph with V no. of vertices and E no. of edges. Let (u, v) be the ~~edge~~ minimum weight graph in G with no duplicate edge weight.

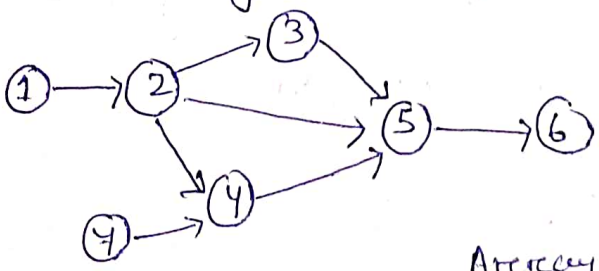
Suppose we make a minimum spanning tree T ; so that (u,v) is not included in T.

Now MST has V-1 edges and no cycle. Now we add the edge (u,v) to the MST which make a cycle In this cycle select an edge whose weight is height and remove it. so we remove the heighest weight edge and add (u,v). In this way the minimum weight edge must be present in every minimum spanning tree.

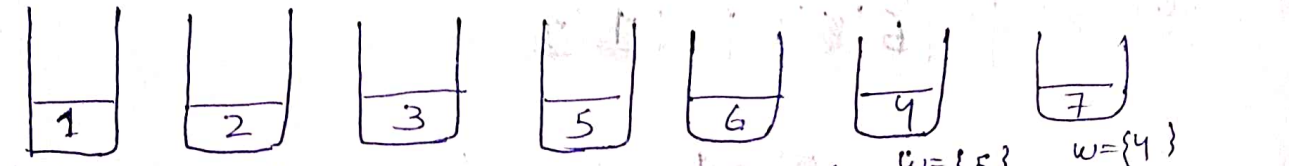
(6) Prove that G is a DAG if and only if the DFS traversal of G has no back edge.

Suppose G has a back edge (u, v), then v is a the ancestor of u in DFS tree. We know that back in DFS tree. forms cycle. So, in Graph G a cycle can be obtained via tree edges.

So G is DAG if and only if the DFS traversal of G has no back edge.



Array => | 1 | 2 | 3 | 5 | 6 | 4 | 7 |



V=1 w={2}

pop 1 v=2 w={3,4,5}

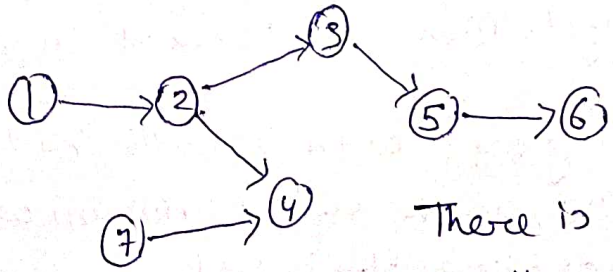
V=3 w={5}

V=5 w={6}

no child backtrack

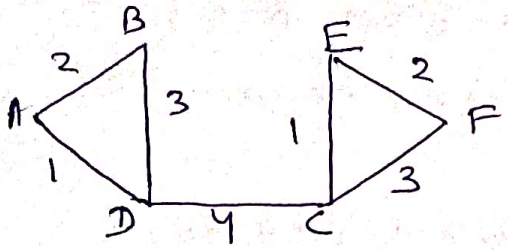
w={5}. already visited. backtrack

w={4} already visited

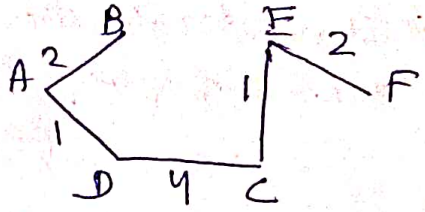


There is no back edge so this graph is DAG.

(11)



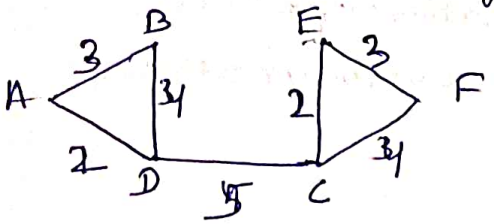
$\Rightarrow$ Spanning tree $\Rightarrow$



G

If we increase the weight of every edge by 1

then

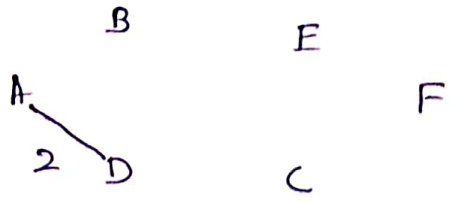


using kruskal-algorithm

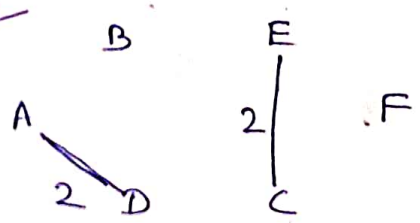
2, 2, 3, ~~3~~, 4, 5

$\longrightarrow$ increasing order of weight

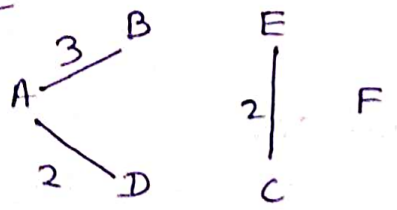
S-1



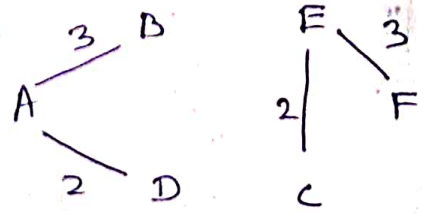
S-2



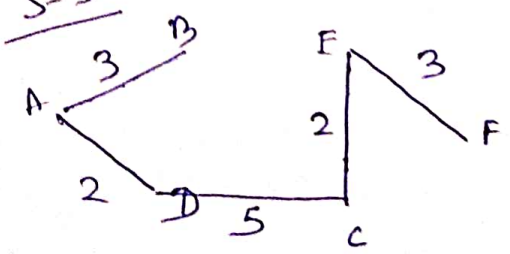
S-3



S-4



S-5



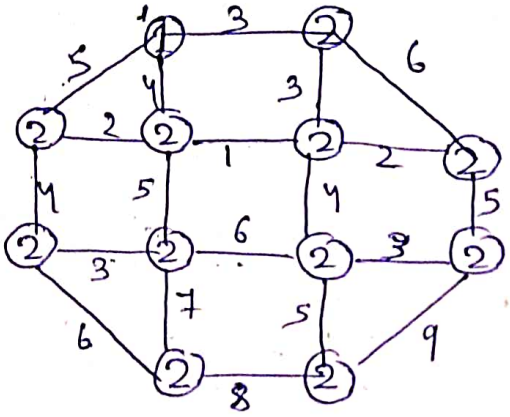
we obtained same path after increasing by 1

So in a graph G if T be the shortest path MST rooted at vertex V. Suppose all the edge weights in G are increased by a constant no. k, then still T is the shortest path spanning tree from V.

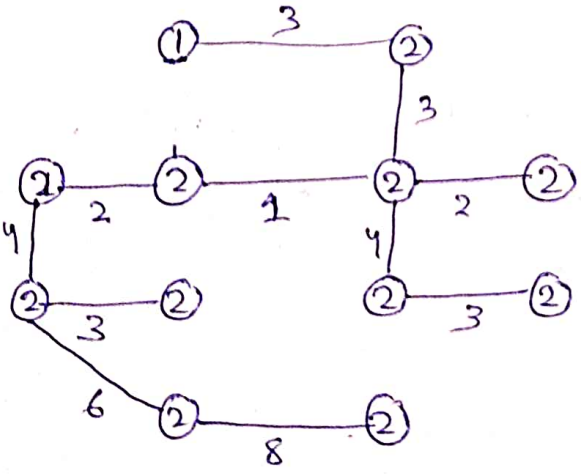
12) Let us denote the tree produced by BFS and DFS be T
Suppose $G \neq T$. This implies there must be an edge
$(u,v) \in G$ such that $(u,v) \notin T$.
Since both BFS and DFS tree are same tree, it follows that one of u and v should be an ancestor of the other and they can differ by only one level.
This implies that the edge connecting them must be in T.
So, it is the contradiction to our assumption.

So $G = T$ (Proved)

(15)

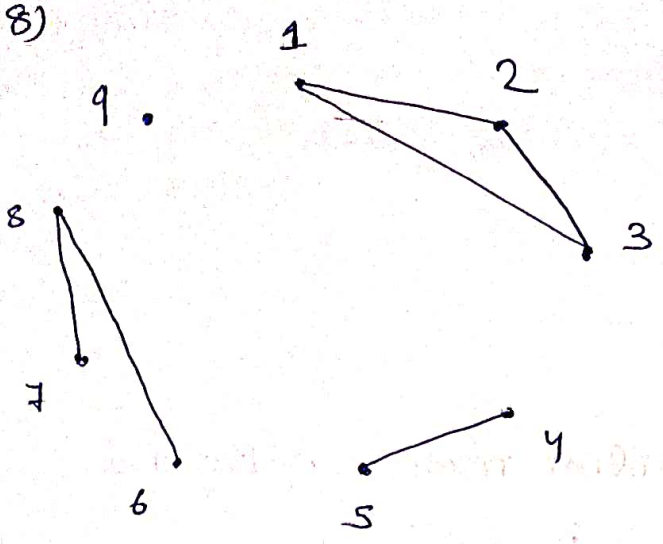


1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5,
6, 6, 6, 7, 8, 9.



total cost = 3 + 3 + 2 + 1
+ 2 + 4 + 3 + 6 + 8
+ 4 + 3 = 39

(8)



(9)(a) Find the transpose of a graph G i.e G^T using adjacency list.

Ans → to find G^T, we need to traverse the adjacency list and as we find a vertex(v) in the adjacency list of vertex(u) which is the indication that an edge from u to v in the graph G of vertices(V), we will add an edge from v to u in its transpose Graph(G^T)

```
display_graph (adj, V);
    for i in range(V);
        print i;
        for j in range (len(adj[i])):
            print adj[i][j].
```

G	G^T
0	
1–2	1
2–5	2–1
3–6	3
4–7	4

* to get transpose.

```
for i in range(V):
    for j in range(len(adj[i])):
        addEdge (transpose, adj[i][j], i)
```

Time complexity ⇒ O(V+E).

(b) using adjacency matrix

Let us consider a graph (G) having vertices (V) and edges(E). Here edges are represented as Row(i) and column (j).

following is the pseudocode for the transpose of Graph(G), where at the end G1(i, j) consists the expected result.

```
for i=0 to i < V[G]
    for j=0 to j < V[G]
        G'(j,i) = G(i,j)
        j = j+1
    i = i+1
```

Time complexity $\Rightarrow O(V^2)$

10) directed graph $G=(V,E)$, incident matrix $\Rightarrow B=b_{ij}$

$$b_{ij} = \begin{cases} -1, & \text{if edge } j \text{ leaves vertex } i \\ 1, & \text{if edge } j \text{ enter vertex } i. \\ 0, & \text{otherwise.} \end{cases}$$

find $BB^T(i,j) = ?$

Ans $\Rightarrow$ $BB^T(i,j) = \sum_{e \in E} b_{ie} b^T_{ej} = \sum_{e \in E} b_{ie} b_{ej}$

- If $i=j$, then $b_{ie} b_{je} = 1$ (it is 1·1 or $(-1)\times(-1)$ when ever e enters or leaves vertex i and 0 otherwise).
- If $i \neq j$, then $b_{ie} b_{je} = 1$ when $e=(i,j)$ or $e=(j,i)$, and 0 otherwise.