Assignment : 2

1. (a) Discuss the representation of graph.

ans- A graph is a non-linear data structure consisting of vertices and edges. The vertices are ~~something~~ sometimes also refferd to as nodes and the edges are linear or arcs that connect any two nodes in the graph.

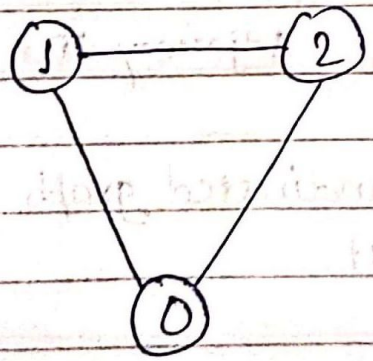
Here are the two most common ways to represent a graph:

1) Adjacency matrix
2) Adjacency list

- Adjacency matrix: It is a way of representing a graph as a matrix of boolean (0's and 1's).

→ If there is an edge from vertex i to j mark adjmat[i][j] as 1 and if there is no edge from vertex i to j mark adjmat[i][j] as 0.

Eg-



Undirected Graph

	0	1	2
0		1	1
1	1		1
2	1	1	

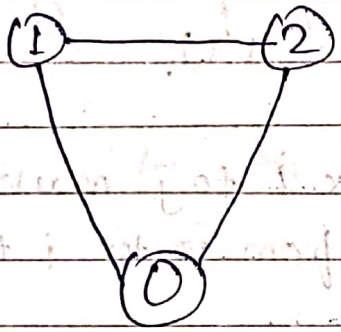
Adjacency matrix

Graph representation of undirected graph to adjacency matrix.

- Adjacency list : An array of list is used to store edges between two vertices. The size of array is equal to the number of vertices. Each index in this array represents a specific vertex in the graph. The entry at the index i of the array contains a linked list containing the vertices that are adjacent to vertex i.

→ adjlist[0] will have all the nodes which are connected to vertex 0.
→ adjList[1] will have all the nodes which are connected to vertex 1 and so on.

Eg.



Undirected graph

Array | Linked list

0	→ 1 → 2 → NULL
1	→ 0 → 2 → NULL
2	→ 0 → 1 → NULL

Adjacency list

Graph representation of undirected graph to adjacency list.

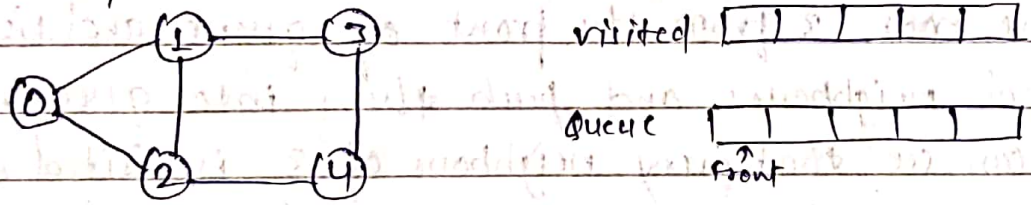
(b) Explain the BFS and DFS in graph with example.

ans:- BFS : The breadth first search algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visit all nodes at the current depth level before moving on to the nodes at the next depth level.

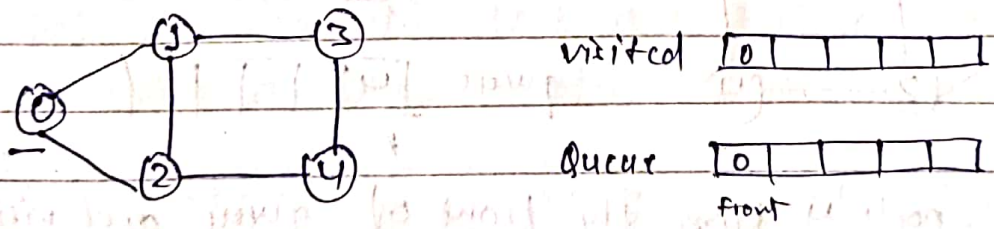
- How does BFS work

All the adjacent unvisited nodes of the current level are pushed into the queue and the node of the current level are marked visited and popped from the queue.

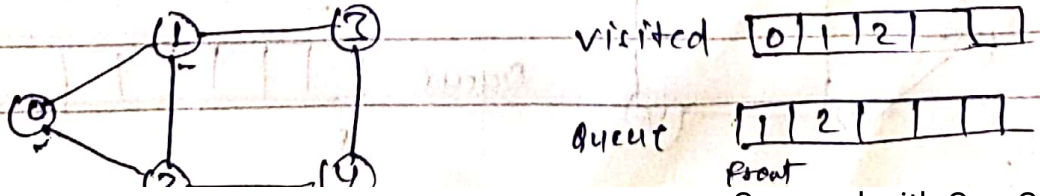
Step 1: Initially queue and visited array are empty



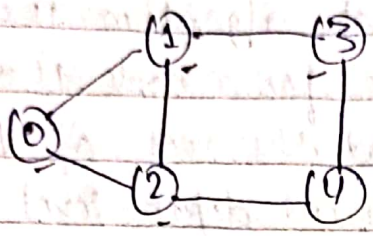
Step 2: Push Node 0 into queue and mark it visited



Step 3: Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.



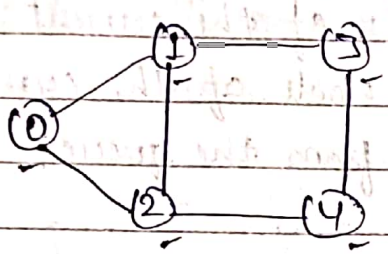
Step 4: Remove node 1 from the front of queue and visit the unvisited neighbour and push them into queue.



visited | 0 | 1 | 2 | 3 | |

Queue | 2 | 3 | | | |
F

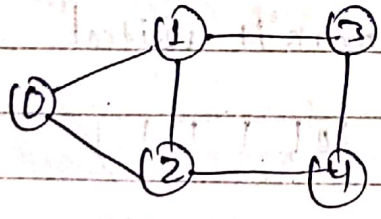
Step 5: Remove node 2 from the front of queue and visit the unvisited neighbour and push them into queue.



visited | 0 | 1 | 2 | 3 | 4 |

Queue | 3 | 4 | | | |
F

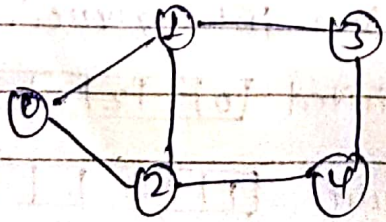
Step 6: Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.
As we can see that every neighbour of 3 is visited, so move to the next node that are in front of queue.



visited | 0 | 1 | 2 | 3 | 4 |

queue | 4 | | | | |
F

Step 7: Remove node 4 from the front of queue and visit the unvisited neighbour and push them into queue.

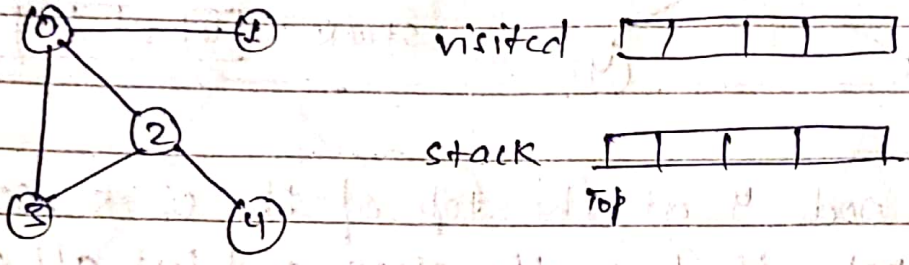


visited | 0 | 1 | 2 | 3 | 4 |

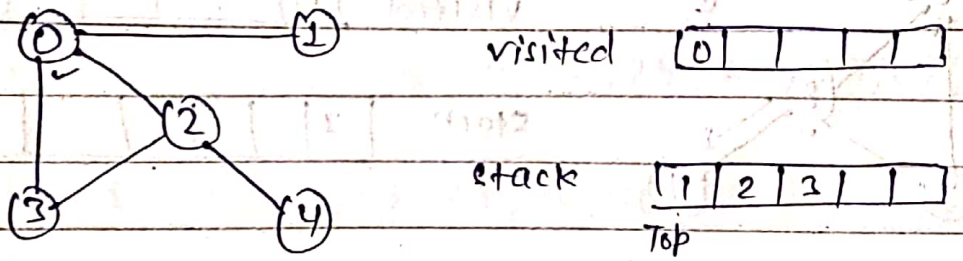
Queue | | | | | |
F

DFS : Depth first search is an algorithm for traversing or searching graph data structure. The algorithm starts at the root node and explores as far as possible along each ~~before~~ branch before backtracking.

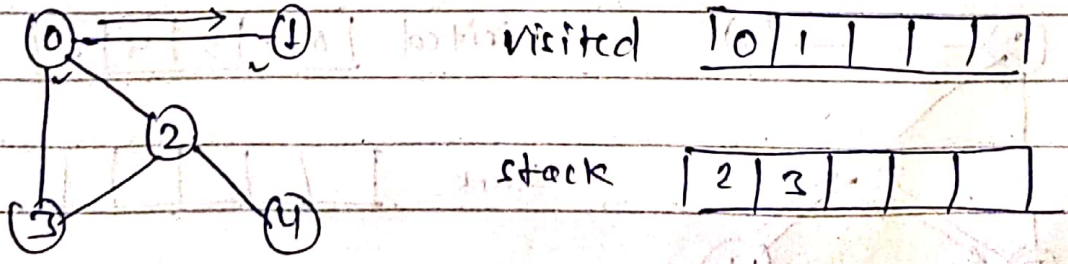
Step1: Initially stack and array are empty



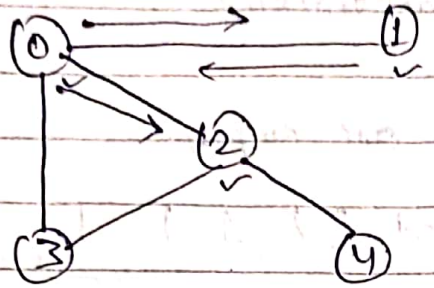
Step 2: visit 0 and put its adjacent nodes which are not visited yet into the stack



Step3: Now node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



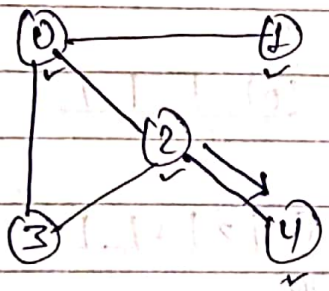
Step 4: Now, node 2 at the top of the stack, so visit node 2 and pop it from the stack and pull all of its adjacent nodes which are not visited in the stack.



visited | 0 | 1 | 2 | | |

stack | 4 | 3 | | | |

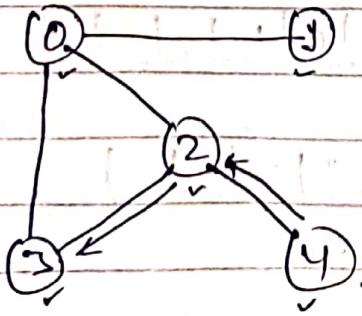
Step 5: Now, node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



visited | 0 | 1 | 2 | 4 | |

stack | 3 | | | | |

Step 6: Now, node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



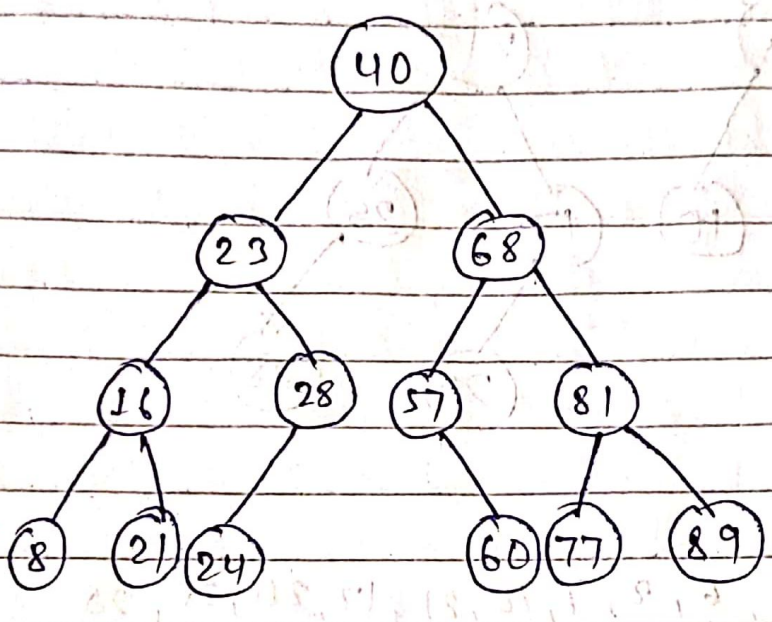
visited | 0 | 1 | 2 | 4 | 3 |

stack | | | | | |

Qno 2: Construct a binary search tree and perform preorder, inorder and post order traversal.

(a) 40, 23, 68, 57, 28, 16, 8, 24, 81, 60, 77, 89, 21

Sol^n-



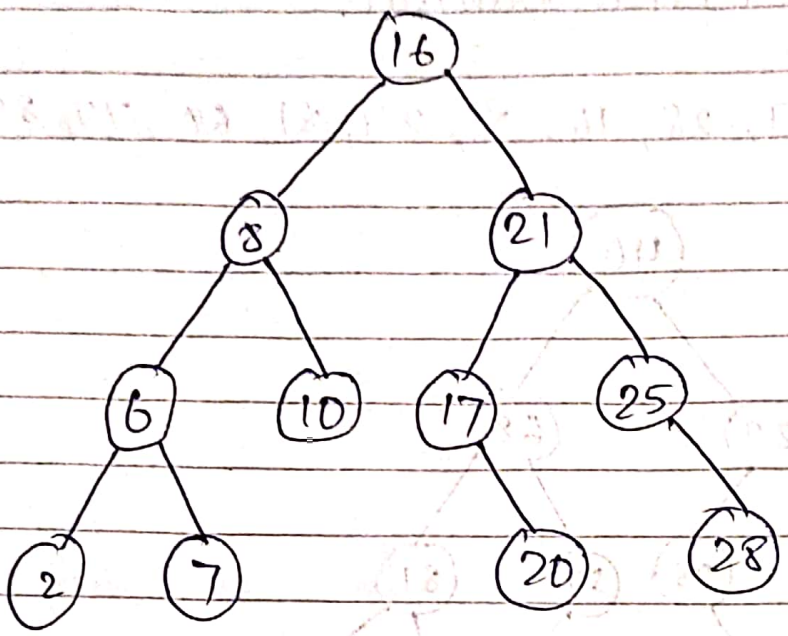
Preorder - 40, 23, 16, 8, 21, 28, 24, 68, 57, 60, 81, 77, 89

Inorder - 8, 16, 21, 23, 24, 28, 40, 57, 60, 68, 77, 81, 89

Postorder - 8, 21, 16, 24, 28, 23, 60, 57, 77, 89, 81, 68, 40

(b) 16, 8, 6, 21, 10, 17, 2, 25, 7, 20, 28

Sol^{n}:-



Preorder - 16, 8, 6, 2, 7, 10, 21, 17, 20, 25, 28 (LRP)

Inorder - 2, 6, 7, 8, 10, 16, 17, 20, 21, 25, 28 (LPR)

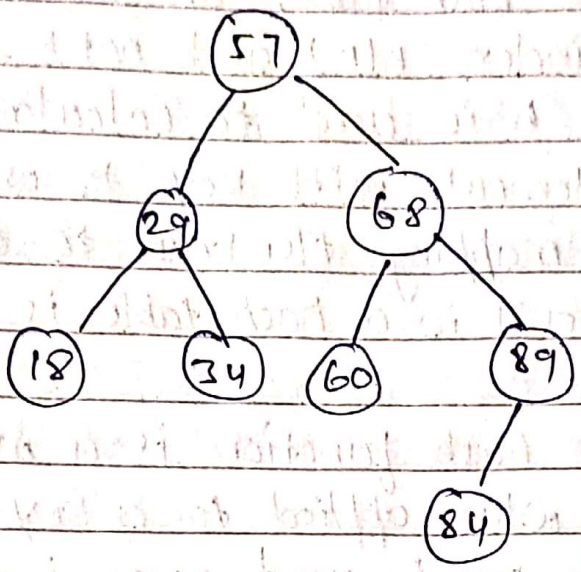
Postorder - 2, 7, 6, 10, 8, 20, 17, 28, 25, 21, 16 (PLR)

Q. Construct a binary search using given data

(a) Preorder : 57, 29, 18, 34, 68, 60, 89, 84

Inorder : 18, 29, 34, 57, 60, 68, 84, 89.

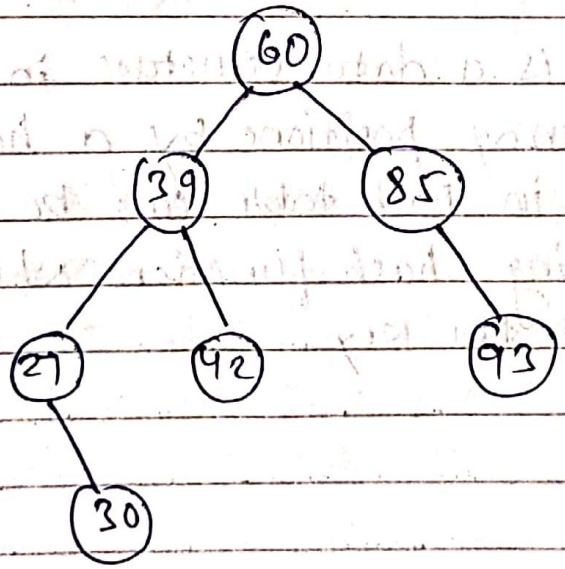
Sol"-



(b) Postorder : 30, 27, 42, 39, 93, 85, 60

Inorder : 27, 30, 39, 42, 60, 85, 93

Sol"-



4 (a) Define following terms : Hashing, Hash function and hash table (in terms of data structure).

Sol:- Hashing :- In a hash table, an element with key K is stored at index h(K) and not K. It means a hash function h is used to calculate the index at which the element with key K will be stored. This process of mapping the keys to appropriate location (or indices) in a hash table is called hashing.

Hash function: A hash function is a mathematical formula which, when applied to a key, produces an integer which can be used as an index for the key in the hash table. The main aim of a hash function is that elements should be relatively, randomly and uniformly distributed.

Hash table: It is a data structure in which keys are mapped to array positions by a hash function. A value stored in hash table can be searched in O(1) time by using hash function which generate an address from the key.

(b) write the characteristics of good hash function. explain different types of hash function.

sol^n - Properties of good hash function -

(i) Low cost : The cost of executing the hash function must be small, so that using the hashing technique become preferable over other approaches.

(ii) Determinism : A hash procedure must be deterministic This means that the same hash value must be generated for a given input value.

(iii) Uniformity : A good hash function must map the keys as evenly as possible over its output range. This means that the probablity of generating every hash value in the output range should roughly be the same.

* Different hash function :

(i) Division method : It is the most simple method of hashing an integer x. This method divides x by m and then uses the reminder obtained.

$$h(x) = x \bmod m$$

(ii) multiplication method :

The steps involved in the multiplication method are follows

step 1 : choose a the constant A such that $0 < A < 1$

Step 2: Multiply the key by A
Step 3: Extract the fractional part of kA
Step 4: Multiply the ~~step~~ result of step 3 by the size of hash table (m).

$$h(k) = [m (kA \bmod 1)]$$

(iii) Mid-square method:
It is a good hash function which works in two steps
step 1: square the value of key, that is k^2
step 2: Extract the middle r digit of the result obtained in step 1.

$h(k) = s$
where s is obtained by selecting r digit from k^2

(iv) Folding method: It works in two steps

Step 1: Divide the key value into a number of parts that is divide k into parts $k_1, k_2, \ldots k_n$ where each part has the same number of digit ~~expect~~ except the last part which may have lesser digit than the other parts.
step 2: Add the individual parts. That is obtain the sum of $k_1 + k_2 + k_3 \ldots + k_n$. The hash value is produced by ignoring the last carry if any

5) Find the array index of given data. Table size is 30.

(key value)

(2, 70)

(42, 80)

(4, 25)

(12, 44)

(14, 32)

(17, 11)

(13, 78)

(37, 98)

Sol^n-

Key, value	Key	Hash	Array Index
(2, 70)	2	2 % 30	2
(42, 80)	42	42 % 30	12
(4, 25)	4	4 % 30	4
(12, 44)	12	12 % 30	12
(14, 32)	14	14 % 30	14
(17, 11)	17	17 % 30	17
(13, 78)	13	13 % 30	13
(37, 98)	37	37 % 30	7

2) Solve the following example using linear probing strategy. Inserting keys [15, 18, 55, 28, 35, 25] using the hash function. Table size is 10. (Hint : f(key) = key % 10.)

Sol:- Inserting keys: {15, 18, 55, 28, 35, 25}

Hash function is f(key) : key % 10.

	Empty	After 15	After 18	After 55	After 28	After 35	After 25
0							25
1							
2							
3							
4							
5		15	15	15	15	15	15
6				55	55	55	55
7						35	35
8			18	18	18	18	18
9					28	28	28