SECTION-A (2marks Questions)

1. How do you test for an empty queue?

:- Queue is said to be empty when the value of front is at -1 or the value of front becomes greater than rear (front > rear).

2. Write down the steps to modify a node in linked lists. CO1

- **:-** Split the list from the middle. Perform front and back split. ...
- Reverse the 2nd(back) list.
- Perform the required subtraction while traversing both list simultaneously.
- Again reverse the 2nd list.
- Concatenate the 2nd list back to the end of the 1st list.

3. Difference between arrays and lists. CO1

List	Array
Can consist of elements belonging to different data types	Only consists of elements belonging to the same data type
No need to explicitly import a module for declaration	Need to explicitly import a module for declaration
Cannot directly handle arithmetic operations	Can directly handle arithmetic operations
Can be nested to contain different type of elements	Must contain either all nested elements of same size
Preferred for shorter sequence of data items	Preferred for longer sequence of data items

4. What are the various Operations performed on the Stack? CO2

- **:-** Push: This function adds an element to the top of the Stack.
- Pop: This function removes the topmost element from the stack.
- IsEmpty: Checks whether the stack is empty.
- IsFull: Checks whether the stack is full.
- Top: Displays the topmost element of the stack.

5. Define Circular Queue. CO2

:- A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer

6. List out the steps involved in deleting a node from a binary search tree. CO3

- :- If the root is NULL, then return root (Base case)
- If the key is less than the root's value, then set root->left = deleteNode(root->left, key)
- If the key is greater than the root's value, then set root->right = deleteNode(root->right, key)
- Else check. ...
- Return.

7. List out few of the Application of tree data-structure. CO3

:- Store hierarchical data, like folder structure, organization structure, XML/HTML data. Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item

8. When a graph said to be weakly connected? CO4

:- A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph (i.e, the graph formed when the direction of the edges are removed).

9. What are the two traversal strategies used in traversing a graph? CO4

:- The graph has two types of traversal algorithms. These are called the Breadth First Search and Depth First Search.

10. State the logic of bubble sort algorithm. CO5

:- Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order. Just like the movement of air bubbles in the water that rise up to the surface, each element of the array move to the end in each iteration.

11. Which sorting algorithm is easily adaptable to singly linked lists? Why?

:- Merge sort is often preferred for sorting a linked list.

SECTION-B (16 MARKS)

1. **Explain the insertion operation in linked list. How nodes are inserted after a specified node**.
CO1 5 MARKS

:- First, create a node using the same structure and find the location where it has to be inserted. Now, the next node at the left should point to the new node. Similar steps should be taken if the node is being inserted at the beginning of the list.
Algorithm
Step 1: IF PTR = NULL.
Step 2: SET NEW_NODE = PTR.
Step 3: SET PTR = PTR → NEXT.
Step 4: SET NEW_NODE → DATA = VAL.
Step 5: SET NEW_NODE → NEXT = HEAD.
Step 6: SET HEAD = NEW_NODE.
Step 7: EXIT.
To insert element in linked list last we would use the following steps to insert a new Node at the last of the doubly linked list. Create a new node. Assign its data value. Assign its next node to NULL as this will be the last(tail) node. Check if the list is empty.

inserted after a specified node.

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;

struct node *temp = head;

for(int i=2; i < position; i++) {
  if(temp->next != NULL) {
    temp = temp->next;
  }
}
newNode->next = temp->next;
temp->next = newNode;
```

2. **Write an algorithm to insert a node at the beginning of list? CO1 5 MARKS**

- :- Allocate memory for new node Store data
- Change next of new node to point to head
- Change head to point to recently created node

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = head;
head = newNode;
```

3. **Write the algorithm for converting infix expression to postfix (polish) expression? CO2 5 MARKS**

:- To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.'

infixToPostfix(infix)

Algorithm
Step 1 : Scan the Infix Expression from left to right.
Step 2 : If the scanned character is an operand, append it with final Infix to Postfix string.
Step 3 : Else,
Step 3.1 : If the precedence order of the scanned(incoming) operator is greater than the precedence order of the operator in the stack (or the stack is empty or the stack contains a '(' or '[' or '{'), push it on stack.
Step 3.2 : Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
Step 4 : If the scanned character is an '(' or '[' or '{', push it to the stack.
Step 5 : If the scanned character is an ')'or ']' or '}', pop the stack and and output it until a '(' or '[' or '{' respectively is encountered, and discard both the parenthesis.
Step 6 : Repeat steps 2-6 until infix expression is scanned.
Step 7 : Print the output
Step 8 : Pop and output from the stack until it is not empty.

4. **What is a DeQueue? Explain its operation with example? CO2 5MARKS**

:- Dequeue in data structure : A dequeue is a linear data structure, which stands for Double Ended Queue. Unlike queue, where the data can be only inserted from one end and deleted from another, in a dequeue, the data can be inserted and deleted from both front and rear ends.

A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy. Insertion in the queue is done from one end known as the rear end or the tail, whereas the deletion is done from another end known as the front end or the head of the queue.

There are two types of deque

Input restricted queue :- In input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.

Output restricted queue :- deletion operation can be performed at only one end, while insertion can be performed from both ends.

Operation :-

Insertion at front :-

- If the queue is empty, both rear and front are initialized with 0. Now, both will point to the first element.
- Otherwise, check the position of the front if the front is less than 1 (front < 1), then reinitialize it by front = n - 1, i.e., the last index of the array.

Insertion at rear :-

- If the queue is empty, both rear and front are initialized with 0. Now, both will point to the first element.
- Otherwise, increment the rear by 1. If the rear is at last index (or size - 1), then instead of increasing it by 1, we have to make it equal to 0.

Deletion at front :-

- If the queue is empty, i.e., front = -1, it is the underflow condition, and we cannot perform the deletion. If the queue is not full, then the element can be inserted from the front end by using the below conditions -
- If the deque has only one element, set rear = -1 and front = -1.
- Else if front is at end (that means front = size - 1), set front = 0.
- Else increment the front by 1, (i.e., front = front + 1).

Deletion at rear :-

- If the queue is empty, i.e., front = -1, it is the underflow condition, and we cannot perform the deletion.
- If the deque has only one element, set rear = -1 and front = -1.
- If rear = 0 (rear is at front), then set rear = n - 1
- Else, decrement the rear by 1 (or, rear = rear -1).

6 Marks

5. **Discuss and explain in detail about the real-world applications based on Data Structure and algorithm.**

:- Data structure and Algorithm (DSA) is applied in all disciplines of software development. DSA is the building block of the software development process. It is not limited to a single programming language. Although programming languages evolve or get dormant over time, DSA is incorporated into all of these languages.

APPLICATION OF ARRAYS:-

- Book titles in a Library Management Systems.
- Online ticket booking.
- Contacts on a cell phone.
- For CPU scheduling in computer.
- To store the possible moves of chess on a chessboard.
- To store images of a specific size on an android or laptop.

Application of Strings:

- Spam email detection.
- Plagiarism detection.
- Search engine.
- Digital forensic and information retrieval system
- Spell checkers.

Application of Matrix:-

- For refraction and reflection in science optics.
- Electronic circuit and quantum physics.
- Media player.
- Mailing list.
- Symbol table creation.

Linked List:-

- Train coaches are connected to one another in a doubly-linked list fashion.

It can be used to implement Stacks, Queues, Graphs, and Trees.

- To perform undo operation.
- Back button.[LIFO]
- Syntax in the coding editor.

Graph:-

- Path Optimization Algorithms, BFS, DFS.
- Recommendation Engines.
- Scientific Computations, Flight Networks, Page ranking.
- Google map to find nearest location.
- Facebook to suggest mutual friends

6. **Discuss about latest research on efficient Data Structures.**

:- Data structures provide ways of compactly organizing and efficiently retrieving various kinds of information. Over the years, they have been used effectively in countless practical and conceptual applications. For example, a nineteenth-century data structure known as the phylogenetic tree, nowadays routinely used for representing the evolutionary history of a set of biological species, has helped scientists to understand the mechanisms of evolution. As another example, a twenty-first-century data structure known as the blockchain, which aims at achieving decentralized consensus, has many potentially important applications involving the creation of permanent ledgers for sharing information over the Internet and automated contracts. For this Special Issue of Algorithms, we would like to invite articles dealing with the design, formal analysis, implementation, and experimental evaluation of efficient data structures for all kinds of computational problems. Of particular interest are algorithms for constructing data structures and extracting information from them efficiently. Articles focusing on complexity aspects of data structures related to time-space tradeoffs, information-theoretic entropy, and lower bounds in various models of computation are also welcome**.**

Also explain :-

a) **Probabilistic Data Structures :-** The Probabilistic data structures and algorithms (PDSA) are a family of advanced approaches that are optimized to use fixed or sublinear memory and constant execution time; they are often based on hashing and have many other useful features.

b) **Dynamic Data Structures :-** A dynamic data structure has a dynamic size, that is, its size can grow and shrink based on the elements present in it at runtime. It makes efficient use of memory by using only that amount of space in memory that is required at any time. Dynamic memory allocation can be done in both the stack and the heap.

c) **Distributed Data Structures :-** a DDS has a strictly defined consistency model: all operations on its elements are atomic, in that any operation completes entirely, or not at all. DDS's have one-copy equivalence, so although data elements in a DDS are replicated, clients see a single, logical data item.

SECTION C (8 MARKS)

1. **Explain INORDER & POSTORDER traversals. Construct an expression tree for the expression (a+b*c) + ((d*e+f)*g). Give the outputs when you apply inorder, preorder and postorder traversals**.

:- **INORDER :-**

This technique follows the 'left root right' policy. It means that first left subtree is visited after that root node is traversed, and finally, the right subtree is traversed. As the root node is traversed between the left and right subtree, it is named inorder traversal.

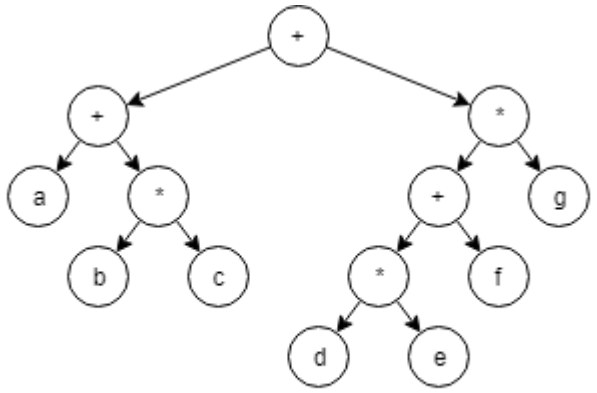
- Traverse the left subtree, i.e., call Inorder(left->subtree)
- Visit the root.
- Traverse the right subtree, i.e., call Inorder(right->subtree)

POSTORDER:-

This technique follows the 'left-right root' policy. It means that the first left subtree of the root node is traversed, after that recursively traverses the right subtree, and finally, the root node is traversed. As the root node is traversed after (or post) the left and right subtree, it is called postorder traversal.

- Traverse the left subtree, i.e., call Postorder(left->subtree)
- Traverse the right subtree, i.e., call Postorder(right->subtree)
- Visit the root

expression (a+b*c) + ((d*e+f)*g).



Expression tree for `(a + b * c) + ((d * e + f) * g)`

OUTPUT FOR INORDER PREORDER & POSTORDER

Inorder : a + b * c + 7

Pre order: + * 7 + c a b

Post order : a b + c * 7 +

SHASHANK SINGH

2. **Write shorts notes on:-**

a) Red-Black Tree :- A red-black tree is a kind of self-balancing binary search tree where each node has an extra bit, and that bit is often interpreted as the color (red or black). These colors are used to ensure that the tree remains balanced during insertions and deletions. Although the balance of the tree is not perfect, it is good enough to reduce

the searching time and maintain it around O(log n) time, where n is the total number of elements in the tree. This tree was invented in 1972 by Rudolf Bayer. It must be noted that as each node requires only 1 bit of space to store the color information, these types of trees show identical memory footprints to the classic (uncolored) binary search tree.

Rules That Every Red-Black Tree Follows:

- Every node has a color either red or black.
- The root of the tree is always black.
- There are no two adjacent red nodes (A red node cannot have a red parent or red child).
- All leaf nodes are black nodes.

b) **AVL Tree :-**

- AVL Tree is invented by GM Adelson - Velsky and EM Landis in 1962. The tree is named AVL in honour of its inventors.
- AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree
- Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

c) **B+ Tree :-**

- A B+ tree is an advanced form of a self-balancing tree in which all the values are present in the leaf level.
- An important concept to be understood before learning B+ tree is multilevel indexing. In multilevel indexing, the index of indices is created as in figure below. It makes accessing the data easier and faster.

Properties of a B+ Tree

- All leaves are at the same level.
- The root has at least two children.
- Each node except root can have a maximum of m children and at least m/2 children.
- Each node can contain a maximum of m - 1 keys and a minimum of ⌈m/2⌉ - 1 keys.

d) **Threaded binary Tree** :- In the linked representation of binary trees, more than one half of the link fields contain NULL values which results in wastage of storage space. If a binary tree consists of n nodes then n+1 link fields contain NULL values. So in order to effectively manage the space, a method was devised by Perlis and Thornton in which the NULL links are replaced with special links known as threads. Such binary trees with threads are known as threaded binary trees. Each node in a threaded binary tree either contains a link to its child node or thread to other nodes in the tree.

3. **Explain Breadth First Search algorithm with example?**

:- Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Applications of BFS algorithm

- BFS can be used to find the neighboring locations from a given source location.

- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network..
- BFS is used to determine the shortest path and minimum spanning tree.
- BFS is also used in Cheney's technique to duplicate the garbage collection.
- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Algo:-

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

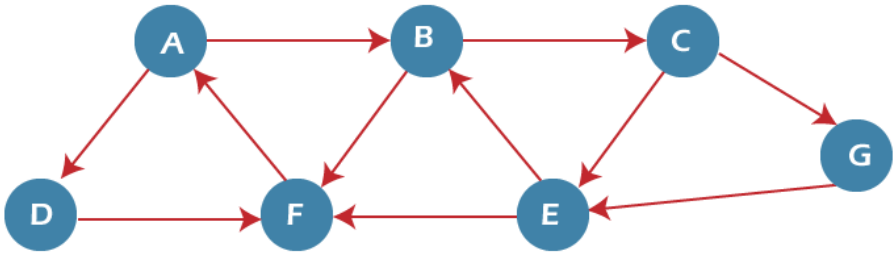
Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2

Example of BFS algorithm

Now, let's understand the working of BFS algorithm by using an example. In the example given below, there is a directed graph having 7 vertices.



Adjacency Lists

A : B, D
B : C, F
C : E, G
G : E
E : B, F
F : A
D : F

In the above graph, minimum path 'P' can be found by using the BFS that will start from Node A and end at Node E. The algorithm uses two queues, namely QUEUE1 and QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.

4. **Write an algorithm to implement Bubble sort with suitable example.**

:- Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n is the number of items.

Algo:-

```
begin BubbleSort(list)

  for all elements of list
    if list[i] > list[i+1]
      swap(list[i], list[i+1])
    end if
  end for

  return list

end BubbleSort
```

Bubble Sort Algorithm is used to arrange N elements in ascending order, and for that, you have to begin with 0th element and compare it with the first element. If the 0th element is found greater than the 1st element, then the swapping operation will be performed, i.e., the two values will get interchanged. In this way, all the elements of the array get compared.

Algorithm

- **Bubble_Sort(list)**
- Pre: list != fi
- Post: list is sorted in ascending order for all values
- for i <- 0 to list:Count - 1
- for j <- 0 to list:Count - 1
- if list[i] < list[j]
- Swap(list[i]; list[j])
- end if
- end for
- end for
- return list
- end Bubble_Sort

5. Explain Depth first and breadth first traversal?

:- BFS :- Breadth-First Search, is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS. Algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Use FIFO.

DFS:- technique used for traversing tree or graph. Here backtracking is used for traversal. In this traversal first the deepest node is visited and then backtracks to it's parent node if no sibling of that node exist. on the other hand, DFS is implemented using a LIFO list.

Sr. No.	Key	BFS	DFS
1	Definition	BFS, stands for Breadth First Search.	DFS, stands for Depth First Search.
2	Data structure	BFS uses Queue to find the shortest path.	DFS uses Stack to find the shortest path.
3	Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
4	Suitablity for decision tree	As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
5	Speed	BFS is slower than DFS.	DFS is faster than BFS.
6	Time Complexity	Time Complexity of BFS = O(V+E) where V is vertices and E is edges.	Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.

S. No.	Parameters	BFS	DFS
1.	Stands for	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2.	Data Structure	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3.	Definition	BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.
4.	Technique	BFS can be used to find a single source shortest path in an unweighted graph because, in BFS, we reach a vertex with a minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
5.	Conceptual Difference	BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.
6.	Approach used	It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).
7.	Suitable for	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.
8.	Suitable for Decision Treestheirwinning	BFS considers all neighbors first and therefore not suitable for decision-making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, and the then explore all paths through this decision. And if this decision leads to win situation, we stop.
9.	Time Complexity	The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.	The Time complexity of DFS is also O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.
10.	Visiting of Siblings/ Children	Here, siblings are visited before the children.	Here, children are visited before the siblings.