પ્રશ્ન
પેટા પ્રશ્ન
ક્રમાંક

	+		+		+		+		=	

BCA Semester: 3

Roll Number: 30

Student Name:
Jat Kishanbhai Laxmanbhai

Subject: Data Structure

Subject Code: 301

પ્રશ્ન પેટા પ્રશ્ન ક્રમાંક

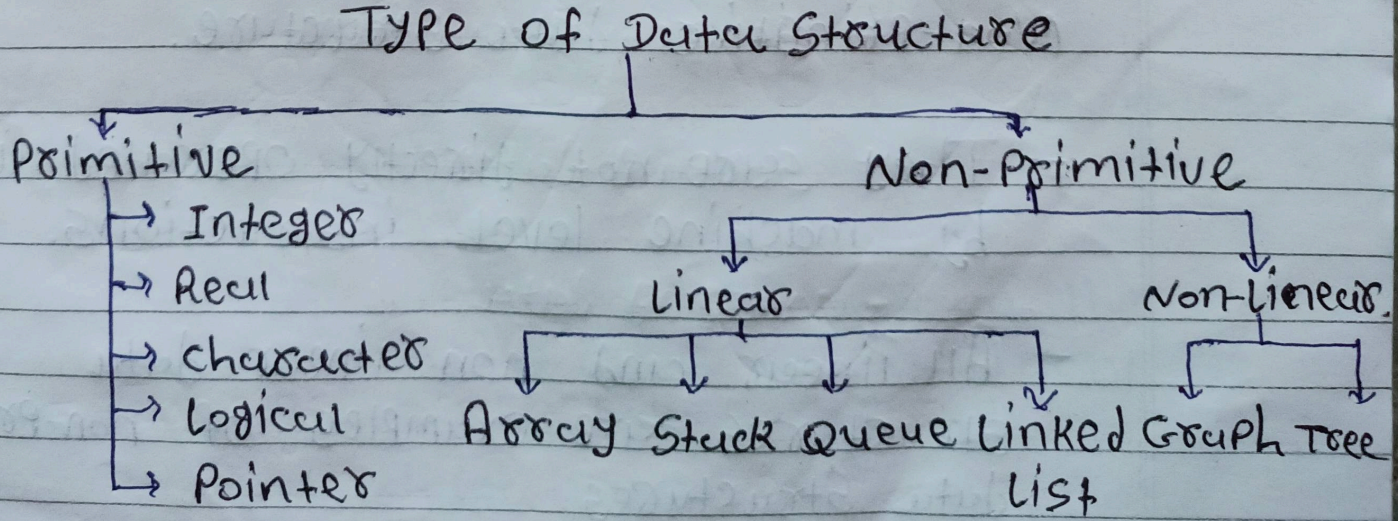
[ ] + [ ] + [ ] + [ ] + [ ] = [ ]

Q 1. Explain Primitive and non-primitive data structure in detail.

A → Data may be organized in different ways where the logical or mathematical model of particular organization of data is called Data Structure.

The choice of particular data structure depends on two considerations.

- It must be rich enough in structure to mirror the actual relationships of the data in real world.
- The structure should be rich enough that one can effectively process the data when necessary.



- Primitive and Non-Primitive Data structure

Primitive data structure ~~includes~~

Primitive data structure includes data at their most basic level within a computer, i.e. the data structures that are directly operated upon by machine level instructions.

Eg. Integer, Real, char, Pointer, Logical.

Non-primitive data structure

A data structure which use the primitive data structure to build based on the logic is called - Non-primitive data structure.

- They are not directly operated upon by machine level instructions.

- All linear and non linear data structures are examples of non-primitive data structures.

* Linear and Non-Linear Data Structures

- In linear data structure the data items are arranged in a linear sequence like in an array, linked list, etc.

- In a non-linear, the data items are not in sequence.
  - An example of a non linear data structure is a tree or a graph.

Primitive and Non-Primitive Data Structures and Operations

- The Primitive data structures are the basic data structures that are available in most of the programming languages as built-in data types.

  - The primitive data structures are used to represent single values.

  - Basic operations allowed on the Primitive data structures include -Addition, -Subtraction, -multiplication and -division etc.

- The data structures that are derived from primary data structures are known as non-primitive data structures.

  - These data structures are used to store group of values.

  - Operations on non-primitive data structures include:

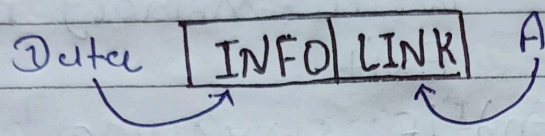
    - Insertion of an item into the data structure.
    - Update an item
    - Delete an item
    - Traversal i.e iterationg through all the items in a data structure.
    - Searching an item
    - Sorting data
    - merging data.

પ્રશ્ન
પેટા પ્રશ્ન
ક્રમાંક

[ ] + [ ] + [ ] + [ ] + [ ] = [ ]

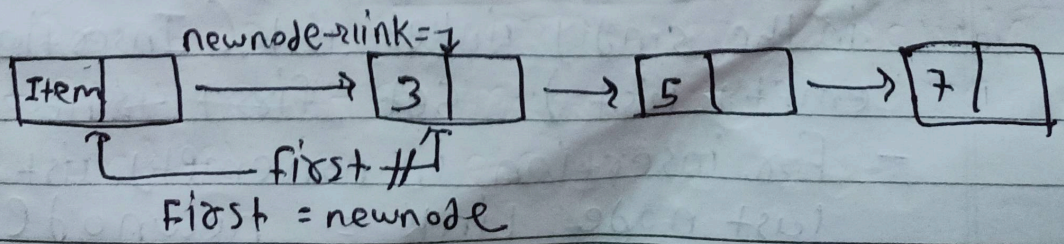
Q 2. What is singly linked-list? Explain insert and delete algorithms of singly link list.

A → List is a data structure that is based on sequences of items.



Node Structure of a Singly Linked List
The node consists of two parts.
- Information Part (INFO): Contains entire record of data items. It may cosist of more than one part.
- Pointer to ~~net~~ next node (LINK): Contains address of next node in the list.

Insertion
main tasks in this algorithm are: (start)
- Create new node and copy item to its information part.
- Set the link part of the new node to point to current first node.
- Reset the First pointer so that it points to new node.



પ્રશ્ન પેટા પ્રશ્ન ક્રમાંક

□ + □ + □ + □ + □ = □

Common Step for Insertion. —(Common)

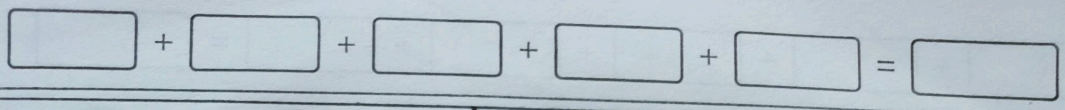
Step 1: [Allocate] memory for newnode]
NewNode := Create new node

Step 2: [Check for successful allocation of memory]
If NewNode = Null then
write "memory not allocated"
Exit
[End If structure]

Step3: [set information part of new node]
Info[NewNode] := Item

InsBegSll (Item) —— Insertion first
Step 1,2,3 —— [Common Step create and modify newNode]
Step 4: [Link the new node to list]
Link[NewNode] := First
Step 5: [update first node]
First := NewNode
Step 6: [Finished]
Exit

- Insertion - node at end of singly linked list.
  - If new node is the first node to be inserted in the singly linked list (Insertion in the empty list) set first as newnode as first.
  - For insertion in nonempty list, set last node link = newnode.



InsEndSll (Item)

Step 1, 2: —— [Same as Common Step 1, 2]

Step 3: [Prepare a new node]

Info[NewNode] := Item, Link[Newnde] := Null

Step 4: [~~Find last node~~ Insertion in empty list]

If First = Null then

First := NewNode and Exit

[End of if structure]

Step 5: [Find last node]

Temp := First

while Link[Temp] != Null then

Temp := Link[Temp]

[End of while structure]

Step 6: [Add the new node and Exit]

Link[Temp] := NewNode and Exit

- InsPosSll (Pos, Item) – Insert At Given Position

Step 1: [Find the predecessor of the new node]

If Pos > 1 then

while(count < Pos-1 and Link[Prev] != Null

Prev := Link[Prev]

count := count++; or Count + 1

[End while]

If count + 1 != Pos then

write "cannot insert node at given position.", Exit

[End if structure]

Else If Pos = 1 then

Prev := NULL

પ્રશ્ન
પેટા પ્રશ્ન
ક્રમાંક

□ + □ + □ + □ + □ = □

Step 2, 3, 4: —— Same as Common Step 1,2,3
Step 5: [Insert new node]
IF Prev != NULL Then
Link[NewNode] := Link[Prev]
Link[Prev] := NewNode
Else Link[NewNode] := First
First := NewNode
Step 6: [Finished] Exit

DelPos(Pos) - Deletion at any Position.
Step 1: [check for underflow (Empty list?)]
If First = NULL then
write "Underflow on delete" Exit
Step 2: [The node to be deleted is first node?]
IF Pos = 1 then
Temp := First ; First := Link[First]
Free(Temp); Exit
Step 3: [Walk to deleted node]
Temp := Link(First) ; Count := 1
Repeat while (Count < Pos-1 AND Link
[Temp] != Null)
Count ++;
Prev := Temp; Temp := Link[Temp]
[End of while loop]
Step 4: [Delete node]
Link[Prev] := Link[Temp]
free [Temp]
Step 5: [Finished]
Exit

પ્રશ્ન
નં. પ્રશ્ન
ક્રમાંક

[ ] + [ ] + [ ] + [ ] + [ ] = [ ]

Q 3. Explain traversal of binary tree in detail.

A → A tree is a binary tree if each node of the tree can have at the most two branches.
- We can also say that if every node of a tree can have at most degree two, then it is called a binary tree.

Traversal of a Tree

Traversal means processing each node of a tree exactly once in a systematic manner.

- There are three main ways of binary tree traversals.
  - Preorder traversal
  - Inorder traversal and
  - Postorder traversal.

Preorder Traversal (Root - Left - Right)

In preorder traversal, the root node is processed first, then the left sub-tree is traversed means we traverse all the roots of the left sub-tree child
- is traversed then its left child. After traversing the left child, the right child.

પ્રશ્ન
પેટા પ્રશ્ન
ક્રમાંક

+ + + + =

is traversed and all the nodes are processed in preorder.

- In short, the preorder traversal of a binary tree is defined as follows:
  a. Process the root node
  b. Traverse the left subtree in preorder
  c. Traverse the right subtree in preorder

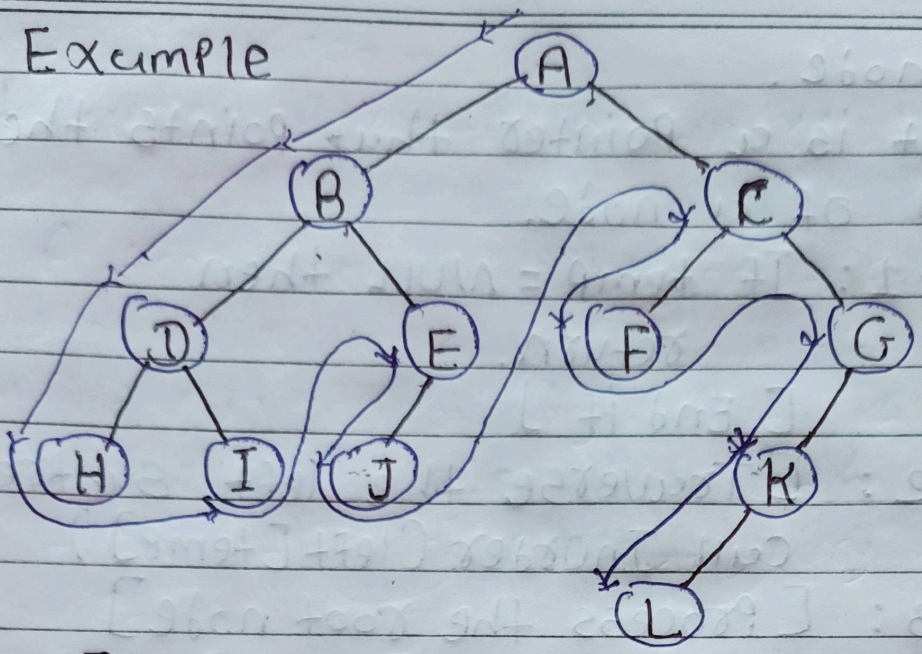
* Algorithm for preorder traversal:
PreOrder (temp)

- Initially temp points to the first node in a binary tree.
- left is a pointer that points to the left child of a node.
- right is a pointer that points to the right child of a node.

Step 1: If temp = NULL then
return
[End if]
Step 2: [process the root node:]
Output data[temp]
Step 3: [Traverse the left subtree]
Call preorder (left[temp])
Step 4: [Traverse the right subtree]
Call preorder (right[temp])
Step 5: Exit

□ + □ + □ + □ + □ = □

- Example



- Its preorder traversal will be as follows:
  A B D H I E J C F G K L

Inorder Traversal (Left - Root - Right)

In inorder traversal, the left subtree is traversed first in inorder.
- Then process the root and then traverse the right subtree in inorder.
- In short, the inorder traversal of a binary tree is defined as follows:
  a. Traverse the left subtree in inorder
  b. Process the root node.
  c. Traverse the right subtree in inorder.

* Algorithm for Inorder traversal:
  InOrder(temp)
  - Initially temp points to the first node in a binary tree.
  - left is a pointer that points the left child.

પ્રશ્ન
પેટા પ્રશ્ન
ક્રમાંક

+ + + + =

of a node.
- right is a pointer that points the right child of a node.

Step 1: If temp = NULL then
return
[End if]

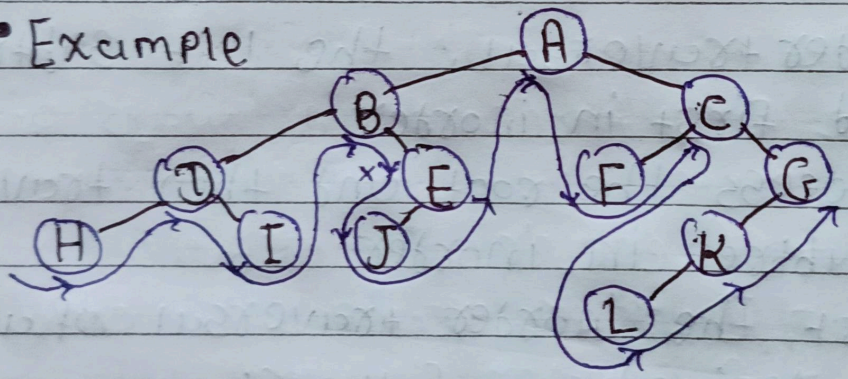
Step 2: [Traverse the left subtree]
Call InOrder (left[temp])

Step 3: [Process the root node]
Output data[temp]

Step 4: [Traverse the right subtree]
Call InOrder (right[temp])

Step 5: Exit

• Example



Complete Inorder traversal:
H D I B J E A F C L K G

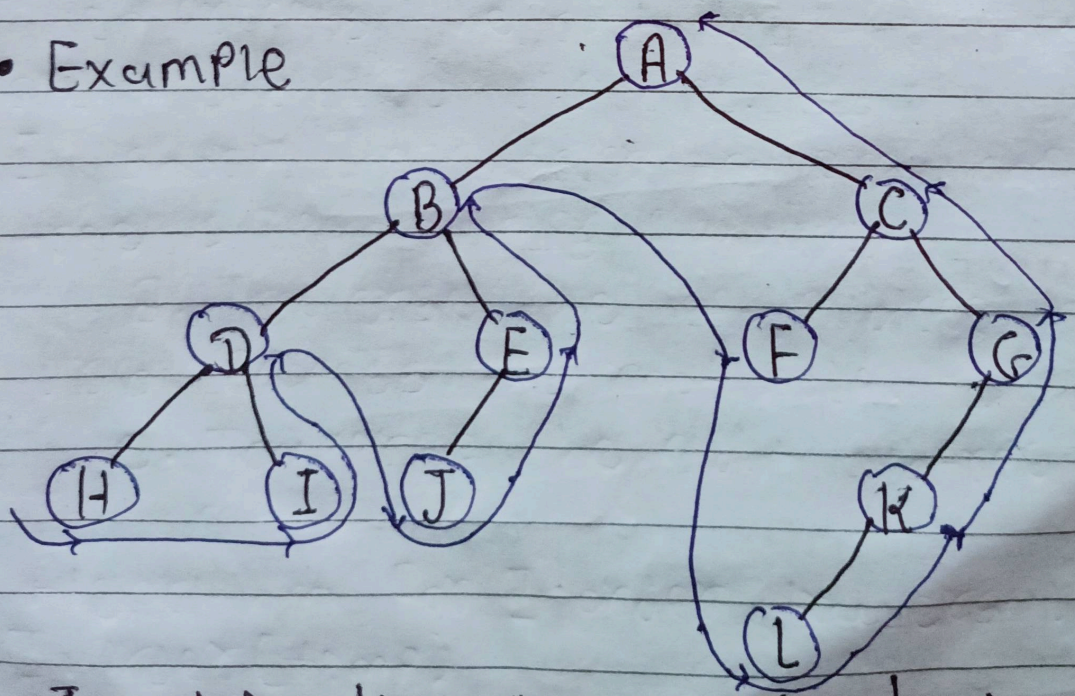
Postorder Traversal (Left-Right-Root)

In Postorder traversal, the left subtree is traversed first in postorder.
- Then traverse the right subtree in postorder and then process the root.

- In short
  a. Traverse the left subtree in postorder.
  b. Traverse the right subtree in postorder.
  c. process the root node.

- Algorithm for postorder traversal:
  step 1: IF temp = NULL then
  return
  [End if]
  step 2: [Traverse the left subtree]
  call postorder(left[temp])
  step 3: [Traverse the right subtree]
  call postorder(right[temp]).
  step 4: [Process the Root node]
  Output data[temp]
  step 5: Exit

- Example



- Inorder traversal will be:
  H I D J E B F L K G C A

Q 4. Explain binary search in detail.

A → A faster way to search a sorted array is to use a binary search.

- To perform binary search, the table must be in sorted order.
- First the middle element from the table is taken.
- It is compared with the key (the number which we ~~wn~~ want to search).

-* If the key is less then the middle element..

- the key has to be found out in the first half of the table
- and this process is repeated on the first half until the required key is found

-* If the key is grater than the middle element.

- the key has to found in the

Second ~~half until the table~~
second half of the table

- and this process is repeated on the second half until the required key is found.

- This process continues until the desired key is found or the search interval becomes empty.

Algorithm to search a number using Binary Search Technique.

Binary_Search (Arr, n, no)
- Here n is the size of the array.
- Arr is the array of n element.
- no is the number that we want to search.
- Flag is an integer variable. Initially it is 0.
- After processing, if its value is 0 means number is not found.
- If it is 1 then the number exists in the given array. LB is the Lowest Bound and UB is the Upper Bound for the array.
- ~~Aai~~ mid is the numeric variable.

Step 1 : [Initialize]
LB := 0
UB := n-1
Flag := 0
Step 2 : Repeat through Step 4 while (LB <= UB)
Step 3 : Mid := (LB+UB) / 2
Step 4 : if (no < Arr[mid] then
UB := mid-1
Else if (no > Arr[mid] then
LB := mid+1
Else if (no = Arr[mid]) then
Flag = 1
Exit from while loop
[End if]

Step 5 : if flag = 1 then
output "Number Exists"
Else
output "Number does not exist"
[End if]

Step 6 : Exit

- Example

Arr = 11, 22, 33, 44, 55, 66, 77, 88, 99

- Suppose we want to find key 88. For that first we have to find the middle element fo from the table