TED (10)-3071 Reg. No. ………………………….

(REVISION-2010) Signature ……..……………………

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/
TECHNOLIGY- MARCH, 2014

DATA STRUCTURE

(Common to CT and IF) [*Time:* 3 hours
(Maximum marks: 100) Marks

PART –A (Maximum marks: 10)

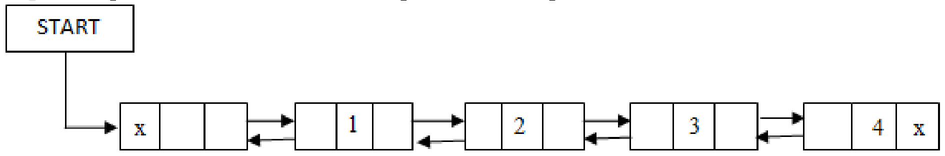
I. Answer all questions in a sentence

1. Define Data structure
Data structure is a particular way of storing and organizing data so that it can be used efficiently.
2. State the use of traversal operation on a data structure
Traversing means accessing each and every element of the array for a specific purpose
3. Write node structure of a linked list to store name and register number of a student.
```
struct node
{
      string name;
      int reg_no;
      struct node *next;
}
```
4. Specify the condition of Binary search tree
In a Binary search tree, all the nodes in left sub tree have a value less than root node. Correspondingly all the nodes in right sub tree have a value greater than root node
5. Define weighted graph
A graph is said to be weighted if every edge in the graph is assigned some data. In a weighted graph, the edges of the graphs are assigned some weight or length

PART – B

II. Answer *any five* questions. Each question carries 6 marks

1. Explain about linear and nonlinear data structure. Give two examples for each.
If the elements of a data structure are stored sequentially, then it is a linear data structure. In linear data structures, we can traverse either forward or backward from a node. Linear data structures can be represented in the memory in two different ways. One way is to have the linear relationship between the elements by means of sequential memory location. Such linear structures are called arrays. The 2nd way is to have the linear relationship between the elements by means of links. Such linear structures are called linked list.
Examples: arrays, stacks
If the elements of a data structure are stored sequentially, then it is non-linear data structure. It branches to more than one node and cannot be traversed in a single run. Examples include trees and graphs
2. Write any three dynamic memory allocation functions with their use and syntax
malloc: Allocate memory and returns a pointer to the first byte of allocated space
Syntax: malloc()
calloc: Allocates space for an array of elements and initializes then to zero. Like malloc(), calloc also returns a pointer to the memory.
Syntax: calloc ()
realloc: Alters the size of previously allocated memory
Syntax: realloc ()
3. With necessary diagram explain about doubly and circular linked lists
Doubly linked list
A doubly linked list or a two – way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. Therefore it consists of three parts, and not just two. Then the three parts are data, a pointer to next node, and a pointer to the previous node.



Doubly linked list

In C, the structure of a doubly linked list is given as,

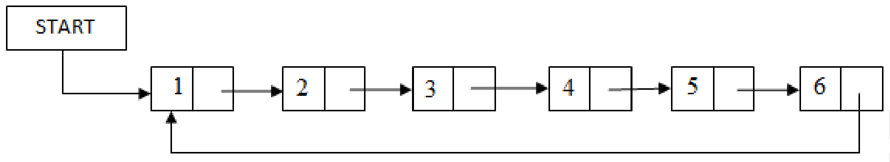
```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
};
```

The prev field of the first node and the next field of the last node will contain NULL. The prev field is used to store the address of the preceding node. This would enable to traverse the list in the backward direction as well.

Thus we see that a doubly linked list calls for more space per node and more expensive basic operations. A doubly linked list provides the ease to manipulate the elements of the list as it maintains pointers to nodes in both the directions (forward and backward). The main advantage of using a doubly linked list is that it makes searching twice as efficient.

Circular linked list

In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly linked list as well as a circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction, forward or backward direction, until we reach the same node where we started. Thus a circular linked list has no beginning and no ending.

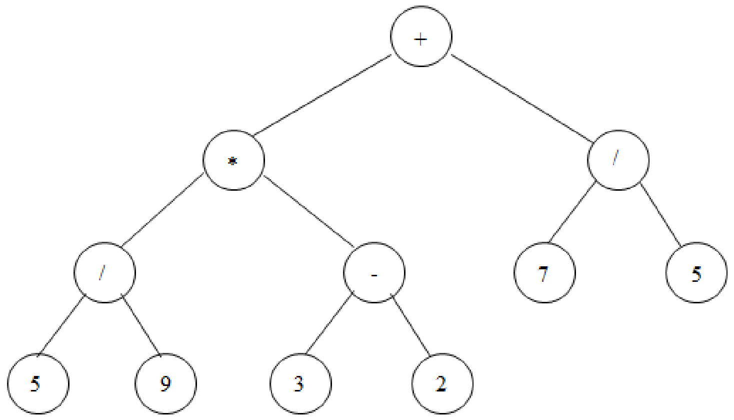


Circular linked list

The only downside of a circular linked list is the complexity of iteration. Note that there is no storing of NULL value in the list.

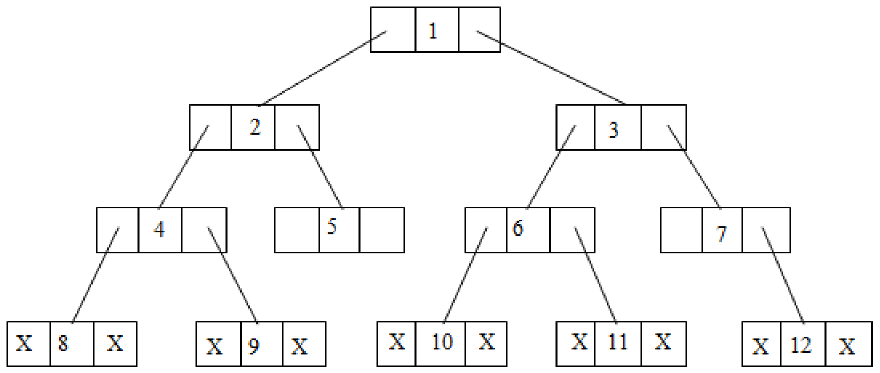
Circular linked lists are widely used in the operating system for task maintenance. Take another example where a circular linked list is used when we are suffering the Net, we can use the Back button and Forward button to move to the previous pages that we have already visited. In this case, a circular linked list is used to maintain the sequence of the list either in forward or backward direction helps to revisit the pages again using Back and Forward buttons. Actually this is done by using either the circular stack or circular queue.

4. Define expression tree. Draw expression tree of 5/9 x (3-2) + 7/5
The algebraic expression which is represented using a binary tree is called expression tree.
Expression tree of 5/9 x (3-2) + 7/5 is



5. Explain about linked representation of a binary tree using an example.
In the linked representation of binary tree, every node has three parts: the data element, a pointer to the left node, and a pointer to the right node.

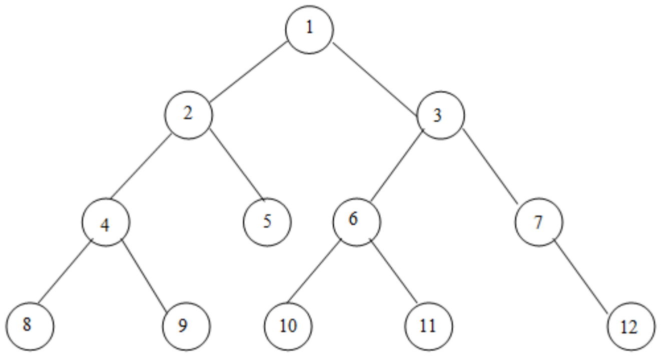
```
struct node
{
    struct node *left;
    int data;
    struct node *right;
};
```



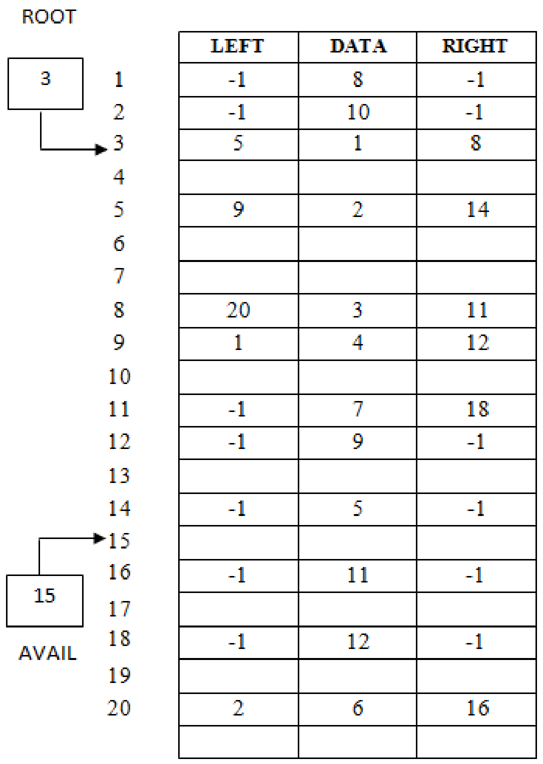
LINKED REPRESENTATION OF A BINARY TREE

In the above figure, left position is used to point to the left child of the node or in technical terms, to store the address of the left child node. The middle position is used to store the data. Finally the right position is used to point to the right child of the node or to sore the address of the right child of the node. Empty sub-trees are represented using x (means NULL)

Example



This tree can be represents in the main memory using a linked list such as



	LEFT	DATA	RIGHT
1	-1	8	-1
2	-1	10	-1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	-1	7	18
12	-1	9	-1
13			
14	-1	5	-1
15			
16	-1	11	-1
17			
18	-1	12	-1
19			
20	2	6	16

6. Explain bubble sort algorithm

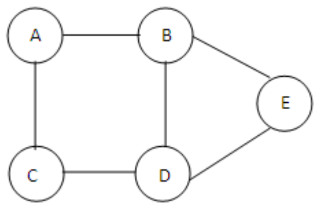
Bubble sort is a very simple method that sorts the array elements that repeatedly moving the largest element to the highest index positionof the array (in case of arraying elements in ascending order).

Algorithm

```
BUBBLE_SORT (A, N)
Step 1: Repeat step 2 FOR I = 0 to N–1
Step 2:         Repeat For J = 0 to N–I
Step 3:                         IF A [J] > A [J + 1], then
                                SWAP A [J] and A [J+1]
                [End of Inner Loop]
        [End of Outer Loop]
Step 4: EXIT
```

In this algorithm, the outer loop is for the total number of passes which is N – 1. The inner loop will be executed for every pass. However, the frequency of the inner loop will decrease with every pass; one element will be in its correct position. There for every pass, the inner loop will executed N – 1times, where N is the number of elements in the array and I is the count of pass.

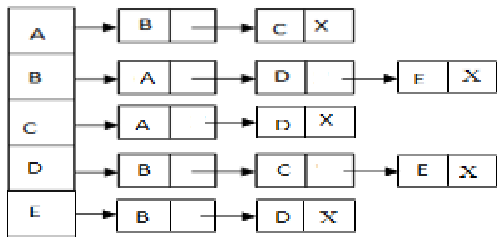
7. Draw adjacency matrix and adjacency list representation of the following graph



Adjacency matrix

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 0 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 0 & 1 & 1 & 0 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

Adjacency list



PART – C

(Answer *one* full question from each unit. Each question carries 15 marks)

UNIT – I

III.

a) Write algorithms of push and pop operations of stack with necessary conditions. 8

Push Operation

The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. However, before inserting the value we must check if *TOP = MAX – 1*, because if that is the case, then the stack is full and no more insertions can be further done. If an attempt is made to insert a value in the stack that is already full, an *OVERFLOW* message is printed

Algorithm

```
Step 1:  IF TOP = MAX - 1 then
PRINT "OVERFLOW"
       [END OF IF]
```

```
Step 2: SET TOP = TOP + 1
Step 3: SET STACK [TOP] = VALUE
Step 4: End
```

In step 1, we first check for the overflow condition. In step 2, *TOP* is incremented so that it points to the next free location in the array. In step 3, the value is stored in the stack array at the location pointed by the *TOP*

1	2	3	4	5					
0	1	2	3	TOP=4	5	6	7	8	9

Stack before insertion

1	2	3	4	5	6				
0	1	2	3	4	TOP = 5	6	7	8	9

Stack after insertion

Pop operation

The pop operation is used to delete the topmost element from the stack. However before deleting the value, we must first check if TOP = NULL because if that is the case, the it means the stack is empty and no more deletions can further be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.

Algorithm

```
Step 1:  IF TOP = NULL then
PRINT "UNDERFLOW"
      [END OF IF]
Step 2: SET VAL = STACK [TOP]
Step 3: SET TOP = TOP - 1
Step 4: End
```

In Step 1 we first check for the underflow condition. In step 2, the value of the location in the stack array pointed by the TOP is stored in VAL. In step 3 TOP is decremented

1	2	3	4	5					
0	1	2	3	TOP=4	5	6	7	8	9

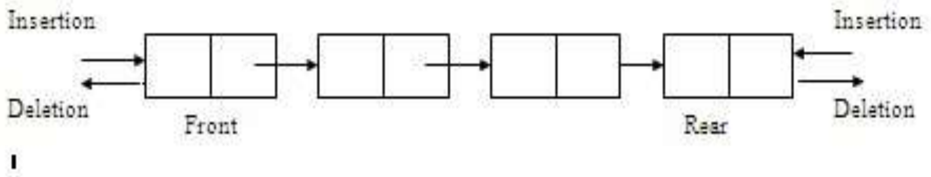
Stack before Deletion

1	2	3	4						
0	1	2	TOP =3	4	5	6	7	8	9

Stack after deletion

b) Explain about dequeue and it's various types. 7

A double-ended queue (dequeue, often abbreviated to deque, pronounced *deck*) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail).[1] It is also often called a head-tail linked list, though properly this refers to a specific data structure implementation



This differs from the queue abstract data type or First-In-First-Out List (FIFO), where elements can only be added to one end and removed from the other. This general data class has some possible sub-types:

- An input-restricted deque is one where deletion can be made from both ends, but insertion can be made at one end only.

- An output-restricted deque is one where insertion can be made at both ends, but deletion can be made from one end only.

Both the basic and most common list types in computing, queues and stacks can be considered specializations of deques, and can be implemented using deques.

OR

IV.

a) Write algorithms to convert infix expression to postfix expression 9

There are 2 algorithms to convert an infix expression into its equivalent prefix expression.

Algorithm 1

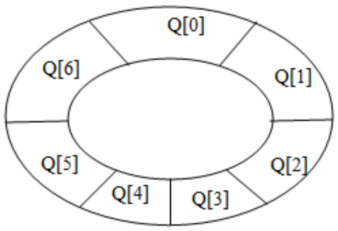
```
Step 1: Scan each character in the infix expression. For this, repeat
        steps 2 – 8 until the end of infix expression.
Step 2: Push the operator into the operator stack, operand into the
        operand stack, and ignore all the left parentheses until a right parenthesis is encountered
Step 3: pop operand 2 from operand stack
Step 4: pop operand1 from operand stack
Step 5: pop operator from operator stack
Step 6: Concatenate operator and operand 1
Step 7: Concatenate result with operand2
Step 6: push result into operand stack
```

Algorithm 2

```
Step 1: Reverse the infix string. Note that while reversing the string, you
        must interchange left and right parenthesis
Step 2: obtain the corresponding postfix expression of the infix
        expression obtained as a result of step 1
Step 3: Reverse the postfix expression to get the prefix expression
```

b) Explain about circular and priority queues 6

Circular queue



In a circular queue, the first index comes right after last index. A circular queue will be full, only when *front = 0*and *rear = max – 1*.A circular queue is implemented in the same manner as a linear queue is implemented. The only difference will be in the code that performs insertion and deletion operations. The only difference will be in the code that performs insertion and deletion operations

Priority queue

A priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

The general rule of processing the elements of a priority queue is:

- An element with higher priority is processed before an element with a lower priority
- Two elements with the same priority are processed on a first-come-first-served(FCFS)

A priority queue is somewhat similar to a queue, with an important distinction: each item is added to a priority queue with a priority level, and will be later removed from the queue with the highest priority element first. That is, the items are (conceptually) stored in the queue in priority order instead of in insertion order.

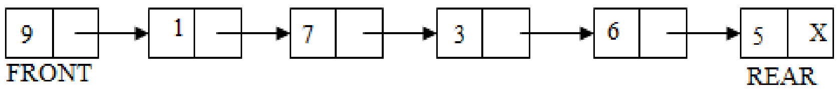
UNIT – II

V.

a) Explain about the implementation of linked queues 8

In case of the queue is a very small one or its maximum size is known advance, then the array implementation of the sack given as efficient implementation. But if the array size cannot be determined in advance, the other alternative, i.e. the linked representation is used

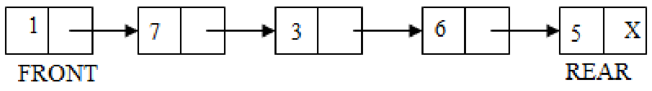
In a linked queue, every element has two parts, one that stores the data and another that stores the address of next element. The START pointer of linked list is used as the FRONT. Here, we will also use another pointer called REAR, which will store the address of last element in the queue. All insertions will be done at the *rear* end and all the deletions are done at the *front* end. If FRONT = REAR = NULL, then it indicates that the queue is empty.



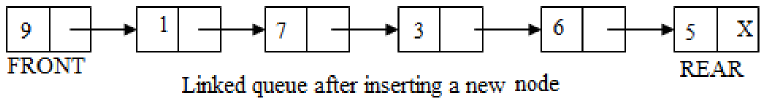
A Queue has two basic operations: *insert* and *delete*. The insert operation adds an element to the end of queue of the stack and the delete operation removes the element from the front or start of the queue.

Insert Operation

The insert operation is used to insert an element into the queue. The new element is added as the last element of the queue.



To insert an element with the value 9, we first check if FRONT = NULL. If the condition holds, then the queue is empty. So we allocate memory for a new node, store the *value* in its *data* part and *null* is in its next part. The new node will then be called the FRONT. However, if FRONT! = NULL, then we will insert the new node at the beginning of the linked stack and name his new node as TOP. Thus the updated stack becomes



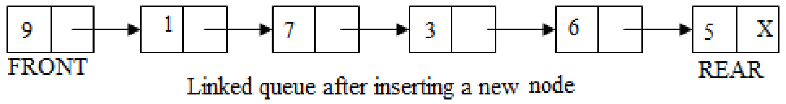
Linked queue after inserting a new node

Algorithm

```
Step 1:  Allocate memory for new node and name it as PTR
Step 2: SET PTR->DATA = VAL
Step 3:  IF FRONT = NULL, then
SET FRONT = REAR = PTR;
SET FRONT ->NEXT = REAR->NEXT = NULL
  ELSE
SET REAR -> NEXT = PTR
SET REAR = PTR
SET REAR->NEXT = NULL
     [END OF IF]
Step 4: EXIT
```

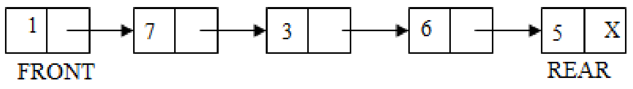
Delete Operation

The delete operation is used to delete the element that was first inserted in a queue. That is the delete operation delete the element whose address is stored in the FRONT. However before deleting the value, we must first check if FRONT = NULL, because if this is the case, then the queue is empty and no more deletions can be done. If an attempt is made to delete a value from a queue that is already empty an UNDERFLOW message is printed.



Linked queue after inserting a new node

To delete an element, we first check if FRONT = NULL. If the condition is false, then we delete the first node printed by FRONT. The FRONT will now pointed to the second element of the linked queue. Thus the updated queue will become



Algorithm

```
Step 1:  IF FRONT = NULL, then
Write "UNDERFLOW"
GOTO STEP 3
     [END OF IF]
Step 2: SET PTR = FRONT
Step 3: FRONT = FRONT->NEXT
Step 4: FREE PTR
Step 5: END
```

b) Write algorithm to search an element of a singly linked list 7

```
Step 1:  [INITIALIZE] SET PTR = START
Step 2:  Repeat Sep 3 while PTR! = NULL
Step 3:          IF POS = PTR
                     Go To step 5
                 ELSE
                     SET PTR = PTR->NEXT
                 [END OF IF]
         [END OF LOOP]
Step 4:  SET POS = NULL
Step 5:  EXIT
```

OR

VI.

a) Write detailed steps to add two polynomials using linked list 10

Every individual term in a polynomial consist of two parts, a coefficient and a power. Every term of a polynomial can be represented as a node of a linked list.

Here we can discuss the addition of two polynomials. Let p and q be the two polynomials used for addition and sum is the 3rd polynomial which is used for store the result represented by the linked list. After checking whether the polynomials are null or not in step 1, we comparing the power of respective term of two polynomials in step 2, if the powers are same, then sum of the terms will be the corresponding term of the sum polynomial. Otherwise if the term of p greater than that of q, term of p will be inserting as the corresponding term of sum, else if the term of p less than that of q, term of q will be insert as the corresponding term of sum. This will continue until p & q become null. After these checking we copy the remaining terms of p & q into the sum polynomial

Algorithm

```
Step 1: while p and q are not null, repeat step 2.
Step 2: IF powers of the two terms ate equal
            IF the terms do not cancel then insert the sum of the terms into
             the sum Polynomial
                Advance p
                Advance q
            Else if the power of the first polynomial> power of second
                Insert the term from first polynomial into sum polynomial
                Advance p
            Else insert the term from second polynomial into sum polynomial
                Advance q
            [END OF IF]
        [END OF IF]
Step 3: copy the remaining terms from the non-empty polynomial into the sum
        Polynomial
Step 4: End
```

b) Compare linked list and arrays 5

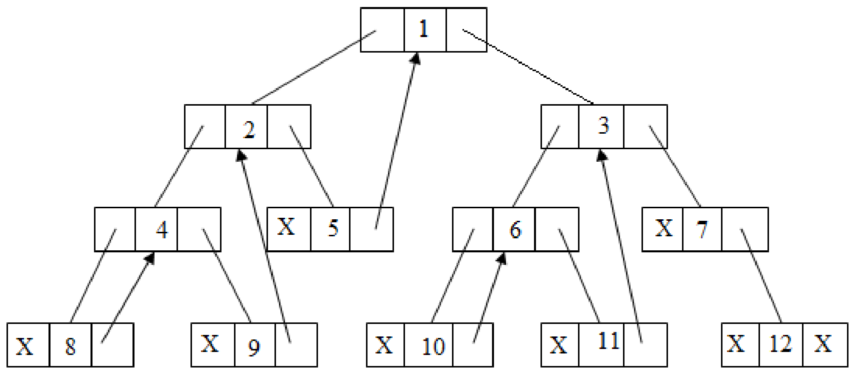
An array is a linear collection of elements and a linked list is a linear collection of nodes. But unlike array a linked list does not store its nodes in consecutive memory locations. Another advantage of a linked list over an array is that a linked list does not allow random access of data. Nodes in a linked list can be accessed only in sequential manner. We can add any number of elements in the list. This is not possible in the case of array. For example, if we declare an array as *int marks [10]*, then the array can store maximum of 10 data elements, but not even one more than that. There is no such restriction in case of a linked list.

UNIT – III

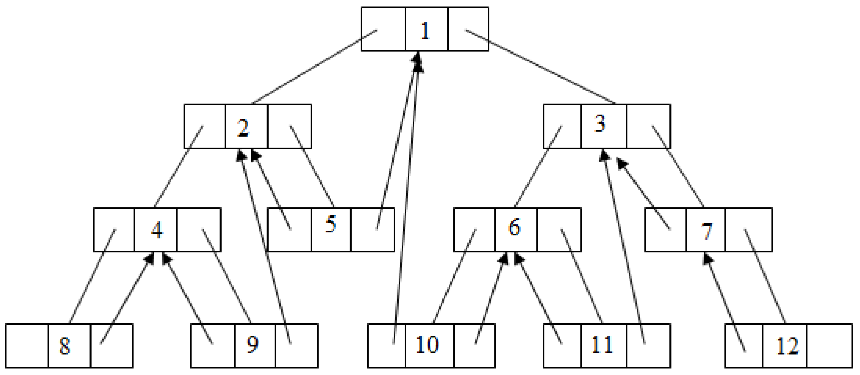
VII.

a) Explain about threaded binary tree. Represent any two types 7

A threaded binary tree is the same as that of a binary tree but with a difference in storing the NULL pointers. In a linked representation, a number of nodes contain a NULLL pointer, either in their left or right fields or in both. This space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information. For example, the NULL entries can be replaced to store a pointer to the in-order predecessor, or the in-order successor of the node. These special pointers are called **threaded trees**. In the linked representation of a threaded binary tree, threads will be represented using dotted lines. A threaded binary tree may correspond to one-way threading or a two-way threading.

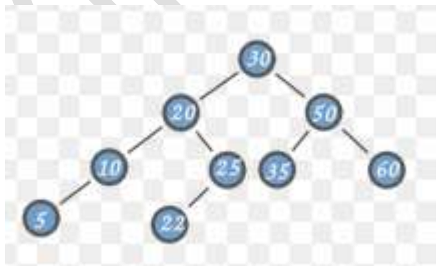


Linked representation of a binary tree with one-way threading



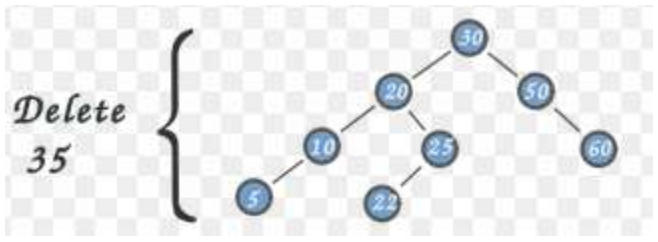
Linked representation of the binary tree with two-way threading

b) Explain various conditions for deleting a node of a binary search tree 8

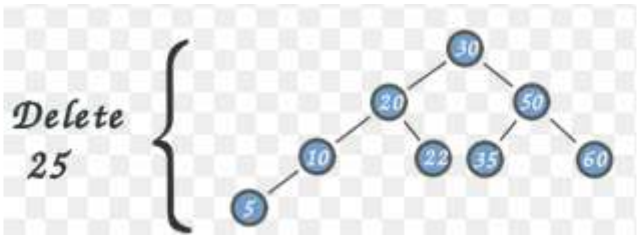


There are three possible cases to consider:

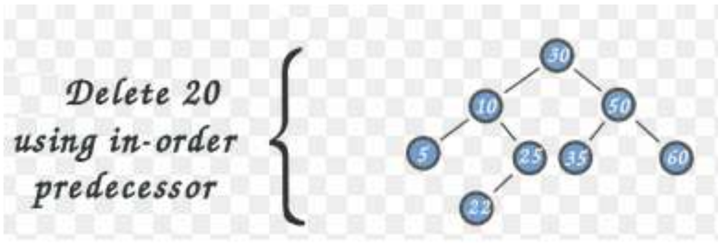
- **Deleting a leaf (node with no children):** Deleting a leaf is easy, as we can simply remove it from the tree.

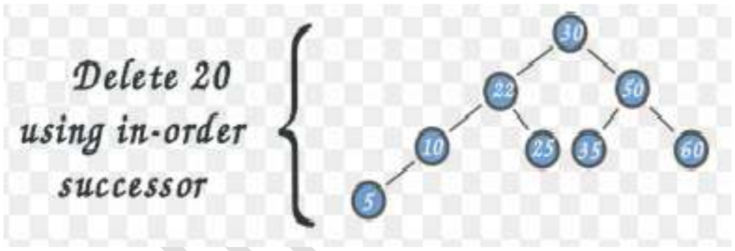


- **Deleting a node with one child:** Remove the node and replace it with its child.



- **Deleting a node with two children:** Call the node to be deleted *N*. Do not delete *N*. Instead, choose either its in-order successor node or its in-order predecessor node, *R*. Copy the value of *R* to *N*, then recursively call delete on *R* until reaching one of the first two cases.





OR

VIII.

a) Explain tree traversal algorithms 9

There are three types of depth-first traversal: pre-order, in-order, and post-order. For a binary tree, they are defined as operations recursively at each node, starting with the root node as follows:

Pre-order Algorithm

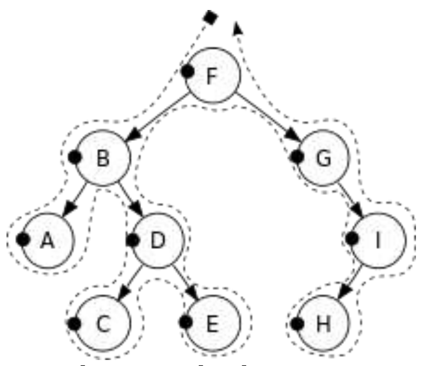
To traverse non empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by:

- Visit the root.
- Traverse the left sub-tree.
- Traverse the right sub-tree.

Pre-order traversal is also called *depth-first traversal.* In this algorithm, the left sub-tree always traversed before the right sub tree. The word 'pre' in the pre-order specifies that the root node is accessed prior to any other nodes in the left and right sub-trees. Pre-order algorithm is also known as the NLR traversal algorithm (Node-Left-Right). Pre-order traversal algorithms are used to extract a prefix notation from an expression tree.
The algorithm for pre-order traversal is shown below.

```
Step 1:  Repeat step 2 to 4 while TREE! = NULL
Step 2:        Write "TREE->DATA"
Step 3:        PREORDER (TREE->LEFT)
Step 4:        PREORDER (TREE->RIGHT)
         [END OF WHILE]
Step 5:  END
```

For example, for the given tree



Pre-order traversal order: F, B, A, D, C, E, G, I, H

In-order Algorithm

To traverse non empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by:

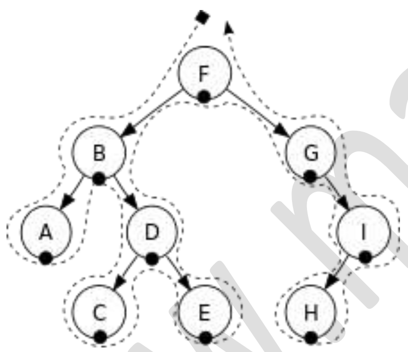
- Traverse the left sub-tree.
- Visit the root.
- Traverse the right sub-tree.

In-order traversal is also called LN traversal algorithm (Left-Node-Right). In-order traversal is usually used to display the elements of a binary search tree. Here all the element with a lower value than a given value are accessed before the elements with a higher value

The algorithm for in-order traversal is shown below.

```
Step 1:  Repeat step 2 to 4 while TREE! = NULL
Step 2:          INORDER (TREE->LEFT)
Step 3:          Write "TREE->DATA"
Step 4:          INORDER (TREE->RIGHT)
            [END OF WHILE]
Step 5:  END
```

For example, for the given tree



In-order Traversal order: A, B, C, D, E, F, G, H, I

Post-order Algorithm

To traverse non empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by:

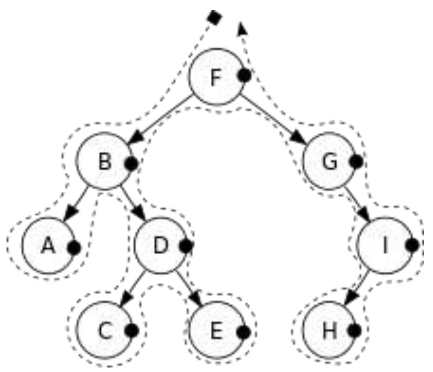
- Traverse the right sub-tree.
- Visit the root.
- Traverse the left sub-tree.

In this algorithm the left sub tree always traversed before the right sub-tree and the root node. The word 'post' in the post-order specifies that the root node is accessed after the left and the right sub trees. Post-order is also known as the LRN traversal algorithm (Left-Right-Node)

The algorithm for post-order traversal is shown below.

```
Step 1:  Repeat step 2 to 4 while TREE! = NULL
Step 2:          POSTORDER (TREE->LEFT)
Step 3:          POSTORDER (TREE->RIGHT)
Step 4:          Write "TREE->DATA"
            [END OF WHILE]
Step 5:  END
```

For example, for the given tree



Post-order traversal order: A, C, E, D, B, H, I, G, F

b) Define binary tree. Write any four terminologies related to binary tree. 6

A binary Tree is a Data structure which defined as a collection of nodes, every node contain left pointer, right pointer and data element. Every binary tree has a root element pointed by a ‘root’ pointer. The root element is a topmost node in the tree. If root = NULL, then the tree is empty.

Sibling: if N is a node in T that has a left successor s_1 and a right successor s_2 then N is called the parent of s_1 and $s_{2.}$ S_1 ands$_2$ are said to be *siblings.*

Degree: the degree of a node equal to the number of the number of children that anode has

Leaf node: A leaf node has no children

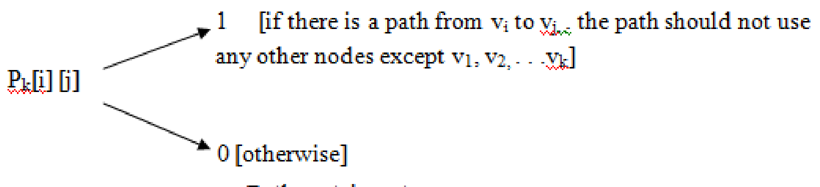
Path: A sequence of consecutive edges is called a path

UNIT – IV

IX.

a) Explain Warshall’s shortest path algorithm 8

If a graph G is given as G = (V, E), where v is the set of vertices and E is the set of edges, the path matrix of G can be found as, $P = A + A^2 + A^3 + \ldots + A^n$. this is a lengthy process, so Warshall has given a very efficient algorithm to calculate the shortest path between two vertices. Warshall’s algorithm defines matrices P_0, P_1,P_2, ...P_n as given in figure


$P_k[i][j]$
1 [if there is a path from v_i to v_j, the path should not use any other nodes except $v_1, v_2, \ldots v_k$]
0 [otherwise]

Path matrix entry

This means that if P_1 [i] [j] = 0, then there exists an edge from node v_i to v_j

If P_1 [i] [j] = 1, then there exists an edge from node v_i to v_j that does not use any other vertex except v_1

If P_2 [i] [j] = 1, then there exists an edge from node v_i to v_j that does not use any other vertex except v_1 and v_2

Note that P_0 is equal to the adjacency matrix of G. if G contains n nodes, then P_n = P which is the path matrix of the graph G

From above discussion we can conclude that P_k [i] [j] is equal to 1 only when either of the two following cases occur:

- There is a path from v_i to v_j that does not use any other node except v_1, v_2,…. V_{k-1}. Therefore
  P_{k-1} [i] [j] = 1
- There is a path from v_i to v_k and a path from v_k to v_j where all the nodes use v_1, v_2,…. V_{k-1}. Therefore
  P_{k-1} [i] [k] = 1 AND P_{k-1} [k] [j] = 1

Hence the path matrix P_n can be calculated with the formula given as:

$P_k [i] [j] = P_{k-1} [i] [j] \vee (P_{k-1} [i] [j] \wedge P_{k-1} [k] [j])$

b) Explain binary search algorithm 7

Binary search algorithm starts with the middle element of the list.

1. If the middle element of the list is equal to the ‘input key’ then we have found the position the specified value.

2. Else if the 'input key' is greater than the middle element then the 'input key' has to be present in the last half of the list.
3. Or if the 'input key' is lesser than the middle element then the 'input key' has to be present in the first half of the list.

Hence, the search list gets reduced by half after each iteration. First, the list has to be sorted in non-decreasing order. [Low, high] denotes the range in which element has to be present and [mid] denotes the middle element.

Initially,
low = 0
high = number_of_elements
mid = floor((low +high)/2).

In every iteration we reduce the range by doing the following until low is less than or equal to high (meaning elements are left in the array) or the position of the 'input key' has been found.

(i) If the middle element (mid) is less than key then key has to present in range [mid+1, high], so low=mid+1, high remains unchanged and mid is adjusted accordingly
(ii) If middle element is the key, then we are done.
(iii) If the middle element is greater than key then key has to be present in the range [low, mid – 1], so high=mid-1, low remains unchanged and mid is adjusted accordingly.

OR

X.

a) Explain quick sort algorithm 8

Quick sort is a divide and conquer algorithm. Quick sort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quick sort can then recursively sort the sub-arrays.

The steps are:

1. Pick an element, called a **pivot**, from the array.
2. Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the **partition** operation.
3. Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

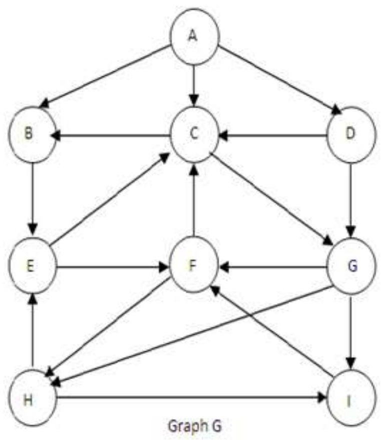
The base case of the recursion is arrays of size zero or one, which never need to be sorted. In pseudo code, a quick sort that sort's elements *i* through *k* (inclusive) of an array *A* can be expressed compactly as

```
quicksort(A, i, k):
  if i < k:
    p := partition(A, i, k)
    quicksort(A, i, p - 1)
    quicksort(A, p + 1, k)
```

b) Explain about DFS algorithm with an example 7

Depth-first search (DFS) is an algorithm for traversing or searching graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking. It is a specialized case of more general graph. The order of the search is down paths and from left to right. The root is examined first; then the left child of the root; then the left child of this node, etc. until a leaf is found. At a leaf, backtrack to the lowest right child and repeat. That is,
Start at some source vertex S.
1. Find (or explore) the first vertex that is adjacent to S.
2. Repeat with this vertex and explore the first vertex that is adjacent to it.
3. When a vertex is found that has no unexplored vertices adjacent to it then backtrack up one level
4. Done when all children have been discovered and examined.

Consider the graph G, and its adjacency lists



Graph G

Adjacency list
A: B, C, D
B: E
C: B, G
D: C, G
E: C, F
F: C, H
G: F, H, I
H: E, I
I: F

The nodes which were printed are:
H, I, F, C, G, B, E