TED (10)-3071 Reg. No. ………………………….

(REVISION-2010) Signature ………………………….

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/
TECHNOLIGY- OCTOBER, 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. List any two linear data structure
   - Queue
   - stack
2. Write any 2 memory allocation functions with their uses
   malloc: Allocate memory and returns a pointer to the first byte of allocated space
   calloc: Allocates space for an array of elements and initializes then to zero. Like malloc(), calloc also returns a pointer to the memory.
3. Define singly linked list
   A singly linked list is a collection of data elements. These data elements are called nodes
4. Specify the conditions of binary tree
   All the nodes in the left sub tree have a value less than that of the root node. Correspondingly all the nodes in the right sub tree have value either greater than or equal to the root node.
5. Define directed graph
   Directed graphs: G=(V,E) where E is composed of ordered pairs of vertices; i.e. the edgeshave direction and point from one vertex to another.

PART – B

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

1. Explain various scheme for representing expressions

   Infix, postfix and prefix notations are three different but equivalent notations of writing algebraic expressions.

   While writing an arithmetic expression using the **infix notation**, the operator is placed between the two operands A and B. Although it is easy for us to write expressions using infix notation, but computers find it difficult to parse because they need a lot of information to evaluate the expression. For example, A + B; here the '+' operator is placed between the two operands A and B

   In the **postfix notation**, the operator is placed after the operands. For example, if an expression is written as A +B in infix notation, the same expression can be written as AB+ in postfix notation

   A **prefix expression** is also evaluated from left to right; the operators in this case are placed before the operands. For example, if A + B is an expression in infix notation, then the corresponding expression in prefix notation is given by + AB

2. Explain about array representation of queue with diagram

   A queue can be easily represented using linear arrays. Every queue has front and rear variables that point to position where deletion and insertion can be done respectively.

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

QUEUE

Here front = 0 and rear = 5. Suppose we want to add another element with the value 6, then the rear would be incremented by 1 and the value would be stored at the position pointed by the rear. The queue would be

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

Queue after insertion of a new elemet

Here front = 0 and rear = 6
If we want to delete an element from the queue, then the value of the front will be incremented. Deletions are done from only this end of queue

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

Queue after deletion of a n element

Here front = 1 and rear = 4

3. Compare arrays and linked list

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.

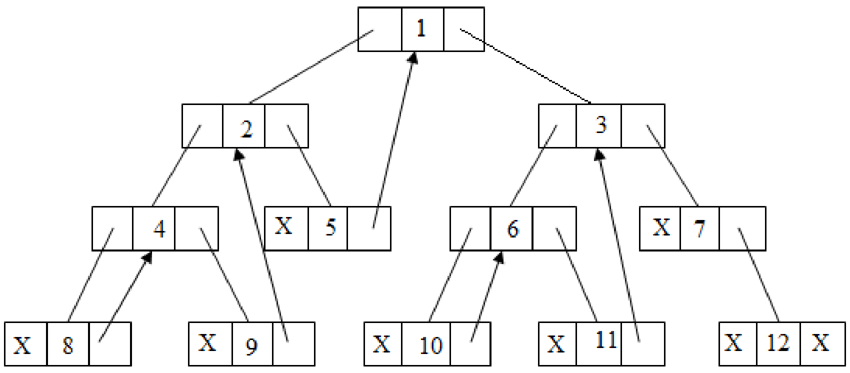
Another advantage of a linked list over an array is that a linked list does not allow random access of data. Nodes in a linked list can be accessed only in sequential manner. We can add any number of elements in the list. This is not possible in the case of array. For example, if we declare an array 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.

4. Write an algorithm to search a node in singly linked list

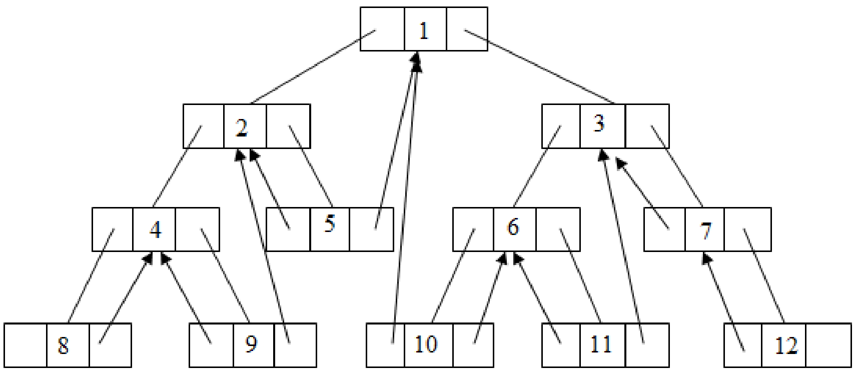
```
    Step 1:  [INITIALIZE] SET PTR = START
    Step 2:  Repeat Sep 3 while PTR! = NULL
    Step 3:              IF POS = PTR
                              Go To step 5
                         ELSE
                              SET PTR = PTR->NEXT
                         [END OF IF]
     [END OF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

5. Define threaded binary tree. Explain any 2 types

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

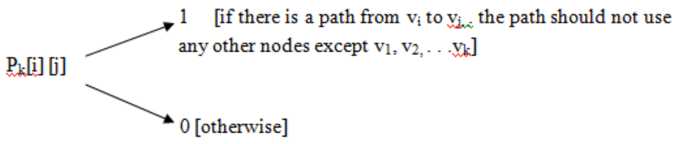


Linked representation of the binary tree with two-way threading

6. Explain shortest path algorithm

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


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, . . .v_k]
P_k[i] [j]
0 [otherwise]

Path matrix entry

7. Explain binary search algorithm

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

a. If the middle element of the list is equal to the 'input key' then we have found the position the specified value.
b. 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.
c. 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.

PART – C

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

UNIT – I

IV.

a) Explain algorithms of insertion and deletion in a linear queue 8

Insertion

```
Algorithm
Step 1:  IF REAR = MAX – 1, then;
                    WRITE "OVERFLOW"
          [END OF IF]
Step 2: IF FRONT == -1 and REAR = -1, then;
                    SET FRONT = REAR = 0
          ELSE
                    SET REAR =REAR + 1
         [END OF IF]
Step 3: SET QUEUE [REAR] = NUM
Step 4: EXIT
```

To insert an element with a value, 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.

Deletion

```
Algorithm
Step 1:  IF FRONT = – 1, OR FRONT > REAR, then;
                    WRITE "ONDERFLOW"
          ELSE
                    SET FRONT =FRONT + 1
                    SET VAL = QUEUE [FRONT]
         [END OF IF]
Step 2: EXIT
```

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.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

b) List applications of stack and queue 7

Stack

- Reverse the order of data(we have already seen its example in Recursion)
- Convert infix expression into postfix
- Convert postfix expression into infix
- Backtracking problem
- System stack is used in every recursive function
- Converting a decimal number into its binary equivalent

Queue

- Priority queues of process:Priority queues are widely used in operating system to execute the highest priority process first
- Online reservation for railway and other
- The linear queue data structure issued for software designing. In linear queue the first coming value will be delete first. Hence the order of coming has priority

OR

V.

a) Explain the algorithm of stack operations 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 UNDERFLOWmessage 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

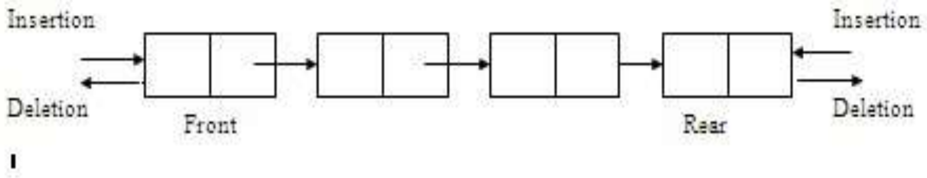
Stack before Deletion

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

Stack after deletion

b) Explain dequeue and priority queue

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). It is also often called a head-tail linked list, though properly this refers to a specific data structureimplementation



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.

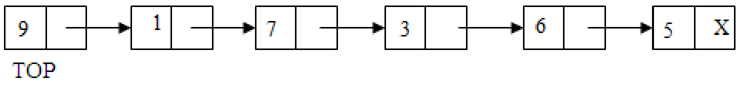
A **priority queue** is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same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

UNIT – II

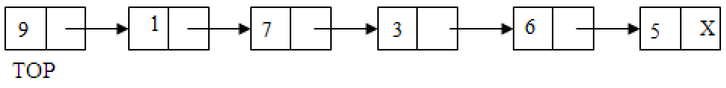
III.
a) Explain about implementation of linked stack

8

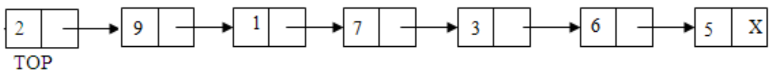
In case of the stack is a very small one or its maximum size is known advance, then the array implementation of the sack given as efficient implementation. But if the array size cannot be determined in advance, the other alternative, i.e. the linked representation is used



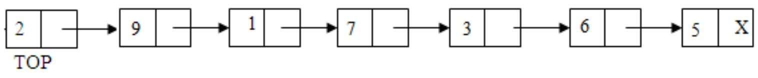
OPERATIONS ON STACK
The stack has major two operations: push and pop. The push operation adds an element to the top of stack and the pop operation remove the element from top of stack
Push Operation



Algorithm
```
Step 1:  Allocate memory for the new node and name it as New_Node
Step 2: SET New_Node->DATA = VAL
Step 3: IF TOP = NULL, then
SET New_Node->NEXT = NULL
SET TOP = New_Node
  ELSE
SET New_Node->NEXT = TOP
SET TOP = New_Node
  [END OF IF]
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*



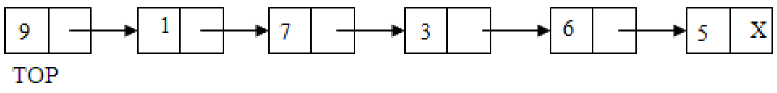
Pop operation



Algorithm

```
Step 1:  IF TOP = NULL then
PRINT "UNDERFLOW"
      [END OF IF]
Step 2: SET PTR = TOP
Step 3: SET TOP = ->NEXT
Step 4: FREE PTR
Step 5: 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



b) Write an algorithm to insert and delete a node in a singly linked list 7

Insertion

```
Step 1. If (START == NULL) Then
Step 2.Print: Linked-List is empty. It must have at least one node
Step 3. Else
Step 4. Set PTR = START, NEW = START
Step 5.Repeat While (PTR != NULL)
Step 6.                If (PTR->INFO == N) Then
Step 7.                     NEW = New Node
Step 8.                     NEW->INFO = ITEM
Step 9.                     NEW->LINK = PTR->LINK
Step 10.                    PTR->LINK = NEW
Step 11.                    Print: ITEM inserted
Step 12.               ELSE
Step 13.                    PTR = PTR->LINK
                       [End of Step 6 If]
                     [End of While Loop]
        [End of Step 1 If]
Step 14. Exit
```

Deletion

```
Step 1. If (START == NULL) Then [Check whether list is empty]
Step 2.                Print: Linked-List is empty.
Step 3. Else If (START->INFO == ITEM) Then
                       [Check if ITEM is in 1st node]
Step 4.                PTR = START
Step 5.                START = START->LINK [START now points to 2nd node]
Step 6.                Delete PTR
Step 7. Else
Step 8.                PTR = START, PREV = START
Step 9.                Repeat While (PTR != NULL)
Step 10.                    If (PTR->INFO == ITEM) Then [If ITEM matches with PTR->INFO]
Step 11.                           PREV->LINK = PTR->LINK [Assign LINK field of PTR to PREV]
Step 12.                           Delete PTR
Step 13.                     Else
Step 14.                           PREV = PTR [Assign PTR to PREV]
Step 15.                           PTR = PTR->LINK [Move PTR to next node]
                            [End of Step 10 If]
                       [End of While Loop]
Step 16.                Print: ITEM deleted
[End of Step 1 If]
Step 17. Exit
```

OR

IV.

a) Write detailed steps to add two polynomials using linked list 9

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.

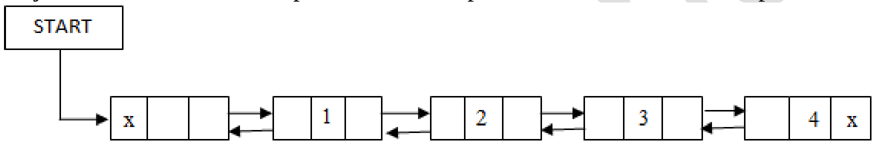
Here we can discuss the addition of two polynomials.Let p and q be the two polynomials used for addition and sum is the 3rd polynomial which is used for store the result represented by the linked list. After checking whether the polynomials are null or not in step 1, we comparing the power of respective 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
```

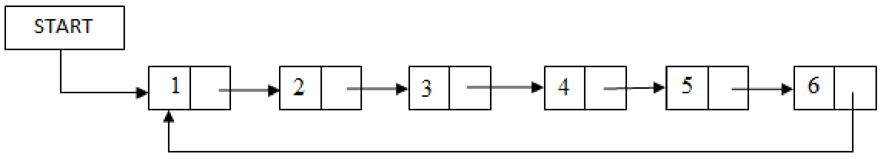
b) Explain about doubly and circular linked list 6

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 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

UNIT – III

V.

a) Explain algorithm to insert and search of a BST 8

Insertion

a. Pre – order

```
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
```

b. Post-order

```
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
    c.  In-order
        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
```

Search

```
    Step 1:  IF TREE->DATA = VAL OR TREE = NULL then
                     Return TREE
             ELSE
                 IF VAL < TREE->DATA
                         Return searchElement(TREE->LEFT, VAL)
                 ELSE
                         Return searchElement(TREE->RIGHT, VAL)
                 [END OF IF]
             [END OF IF]
    Step 2: End
```

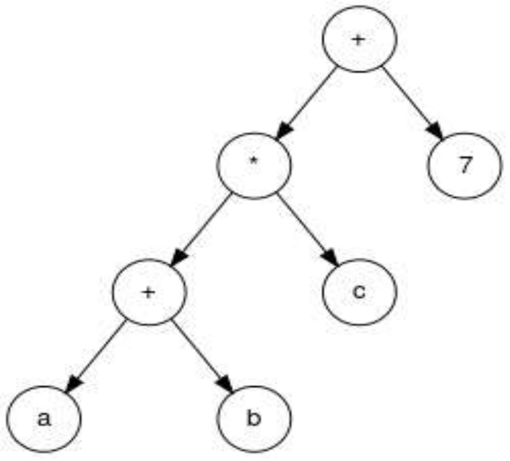
b) Define expression tree. Write an infix expression and draw its expression 7

A **binary expression tree** is a specific application of a binary tree to evaluate certain expressions. Two common types of expressions that a binary expression tree can represent are algebraic and Boolean. These trees can represent expressions that contain both unary and binary operators.

In general, expression trees are a special kind of binary tree. A binary tree is a tree in which all nodes contain zero, one or two children. This restricted structure simplifies the programmatic processing of Expression trees.

The leaves of a binary expression tree are operands, such as constants or variable names, and the other nodes contain operators. These particular trees happen to be binary, because all of the operations are binary, and although this is the simplest case, it is possible for nodes to have more than two children. It is also possible for a node to have only one child, as is the case with the unary minus operator. An expression tree, *T*, can be evaluated by applying the operator at the root to the values obtained by recursively evaluating the left and right sub-trees

For example, the expression tree of **(a + b) * c + 7** is



OR

VI.

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:

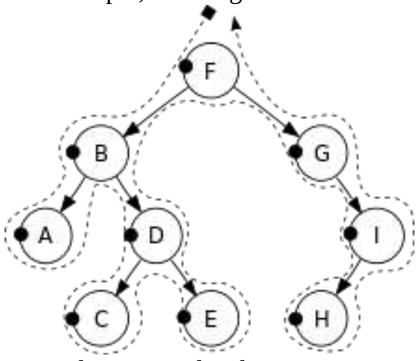
- Visit the root.
- Traverse the left sub-tree.
- Traverse the right sub-tree.

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

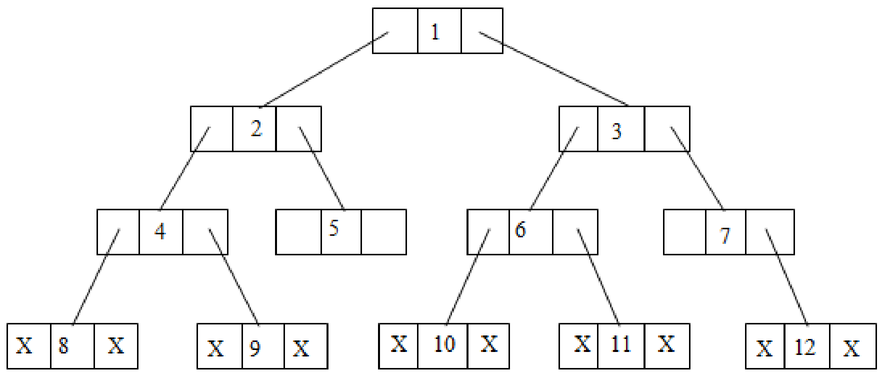
Post-order traversal order: A, C, E, D, B, H, I, G, F

b) Explain about linked representation of binary tree with suitable example 6

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 {
```

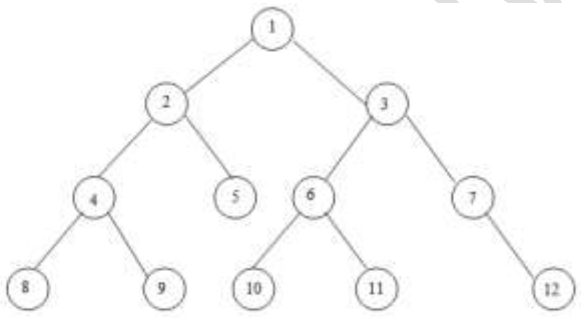
```
struct node *left;
int data;
struct node *right;
};
```



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



UNIT – IV

VII.

a) Explain quick sort algorithm 8

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

The steps are:

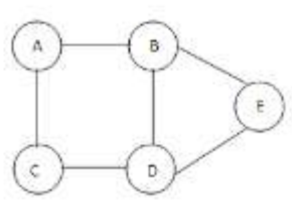
1. Pick an element, called a **pivot**, from the array.
2. Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final 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) Draw a graph and represent it using adjacency matrix and adjacency list 7

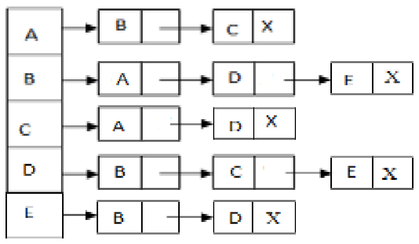
Graph



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



OR

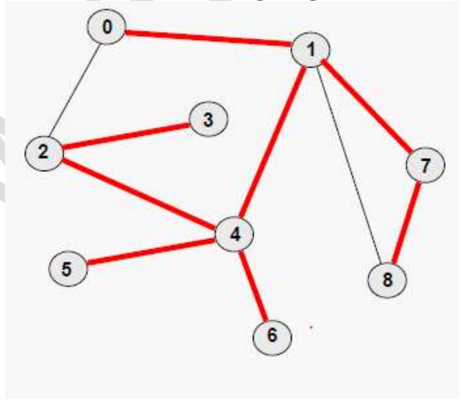
VIII.

a) Explain about DFS & BFS graph traversal algorithms 9

Depth-first search

Assume a particular node has been designated as the starting point.Let A be the last node visited and suppose A has neighbors N1, N2, …,Nk.A depth-first traversal will:

- visit N1, then
- proceed to traverse all the unvisited neighbors of N1, then
- Proceed to traverse the remaining neighbors of A in similar fashion.



Assuming the node labeled **0** has beendesignated as the starting point, a depth-firsttraversalwould visit the graph nodesin the order:

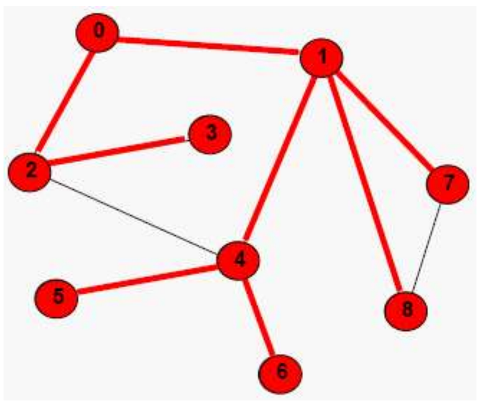
0 1 2 3 4 5 6 7 8

Note that if the edges taken during thedepth-first traversal are marked, theydefine a tree (notnecessarily binary)which includes all the nodes of the graph.Such a tree is called a spanning tree forthe graph.

Breadth-first search

Assume a particular node has been designated as the starting point.Let A be the last node visited and suppose A has neighbors N1, N2, …,Nk.A breadth-first traversal will:

- visit N1, then N2, and so forth through Nk, then
- proceed to traverse all the unvisited immediate neighbors of N1, then
- Traverse the immediate neighbors of N2, …Nk in similar fashion.



If we pick node **0** as our starting point:

0 1 2 4 7 8 3 5 6

b) Explain linear search algorithm 6

Linear search is the simplest search algorithm. It is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.

```
LINEAR_SEARCH (A, N, VAL, POS)
Step 1: [INITIALIZE] SET POS = -1
Step 2: [INITIALIZE] SET I = 0
Step 3:         Repeat Step 4 while I<N
Step 4:             IF A [ I ] = VAL, then
                        SET POS = I
                        PRINT POS
                        Go to step 6
                    [END OF IF]
                [END OF LOOP]
Step 5: PRINT "Value Not Present In the array"
Step 6: EXIT
```