TED (10)-3071 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- MARCH, 2012

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. State the function of traversal operation on data structure
   Traversing means accessing each and every element of the array for a specific purpose
2. Write the properties of a double ended queue
   - Elements can be added to or removed from either the front (head) or back (tail)
   - No element can be added and deleted from the middle
3. Write the structure of a node in a singly linked list which stores the name and author of books

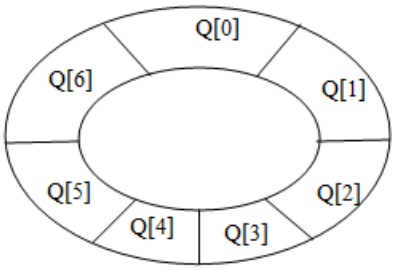
```
struct node
{
        stringbook_name;
        string author;
        struct node *next;
}
```

4. State the use of threads in Binary Search tree
   The space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information. A threaded binary tree is used for storing NULL pointers
5. Differentiate time complexity and space complexity
   Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input
   Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

PART – B

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

1. Explain the advantages of circular queue over linear queue
   In linear queue the insertion can be done at one end called rear. And deletion can be always done from other end called front. For example assume that front = 0 and rear = 9, where the maximum size is 10. Now insertion is not possible since the queue is full. Consider scenario in which 2 successive deletions are made. So front = 2 and rear = 9.now there are 2 empty spaces and we want to insert a new number. But "Queue is full" message still exist because the condition rear = max still hold true. This is known as compaction.



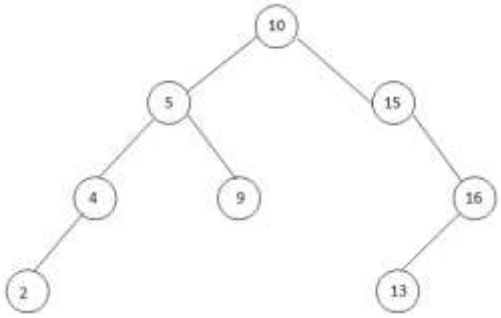
   By using circular queue, the disadvantage of linear queue-compaction can be avoided. A circular queue is implemented in the same manner as a linear queue is implemented.
2. State the applications of stack
   - Reverse the order of data(we have already seen its example in Recursion)
   - Convert infix 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
3. Explain the algorithm to traverse nodes in a singly linked list

```
Step 1:  [INITIALIZE] SET PTR = START
Step 2:  Repeat Step 3 and 4 while PTR! = NULL
Step 3:      Apply Process to PTR-> DATA
Step 4:      SET PTR = PTR -> NEXT
           [END OF LOOP]
```

Step 5: EXIT

In this algorithm we initialize PTR with the address of START. So now, PTR points to the 1st node of linked list. Then in step 2, a while loop is executed which is repeated till PTR process the last node. In step 3 we apply the process to the current node that is the node pointed by PRT. In step 4 we move to the next node by making the PTR variable pointed to the node whose address is stored in the NEXT field.

4. Check whether the given binary tree is BST or not. Justify your answer



Yes. This is a binary tree. Because of

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub-tree each must also be a binary search tree.
- Each node can have up to two successor nodes.
- There must be no duplicate nodes.
- A unique path exists from the root to every other node.

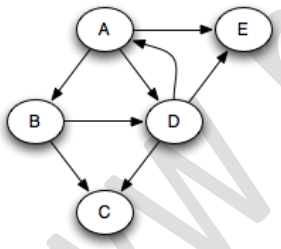
5. Write an algorithm to find the sum of all values stored in the nodes of a Binary Tree

The algorithm for in-order traversal is shown below.

Step 1: [INITIALIZE] SET SUM = 0
Step 2: Repeat step 2 to 4 while TREE! = NULL
Step 3: INORDER (TREE->LEFT)
Step 4: Set SUM = SUM + TREE->DATA
Step 5: INORDER (TREE->RIGHT)
[END OF WHILE]
Step 6: Write SUM
Step 7: END

6. Describe the BFS graph traversal algorithm.

Breadth-first search (BFS) is a graph traversal algorithm that explores nodes in the order of their distance from the roots, where distance is defined as the minimum path length from a root to the node.



In that case, the following sequence of nodes pass through the queue, where each node is annotated by its minimum distance from the source node A. Note that we're pushing onto the right of the queue and popping from the left.

A0 B1 D1 E1 C2

Clearly, nodes are popped in distance order: A, B, D, E, C. This is very useful when we are trying to find the shortest path through the graph to something.

7. Compute the complexity of Bubble sort algorithm

Best-Case Time Complexity

- The scenario under which the algorithm will do the least amount of work (finish the fastest)
- Array is already sorted
- Need 1 iteration with (N-1) comparisons
- O(N)

Worst-Case Time Complexity

- The scenario under which the algorithm will do the largest amount of work (finish the slowest)
- Need N-1 iterations
- (N-1) + (N-2) + (N-3) + …. + (1) = (N-1)* N / 2

- $O(N^2)$

Average-Case Time Complexity

- Bubble sort: O(n2)
- Computation of average-case running time entails knowing all possible input sequences, the probability distribution of occurrence of these sequences, and the running times for the individual sequences
- The average-case running time of an algorithm is an estimate of the running time for an "average" input
- $O(N^2)$

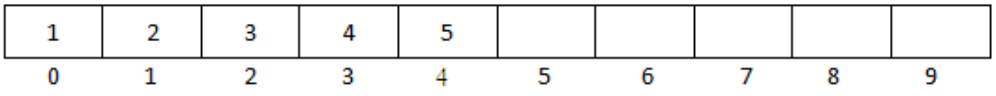
PART – C

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

UNIT – I

III.

a) Explain the operations of a linear queue and the algorithm to implement using array 10

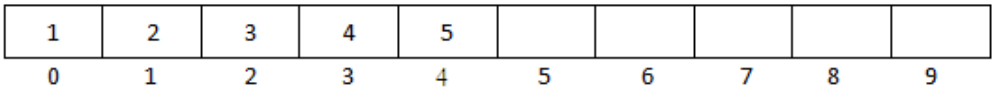


QUEUE

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.

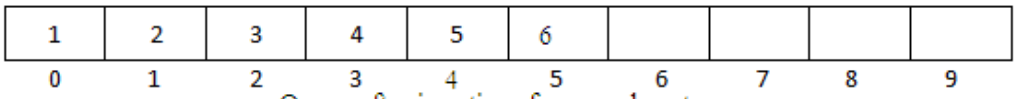
Insert Operation

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



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



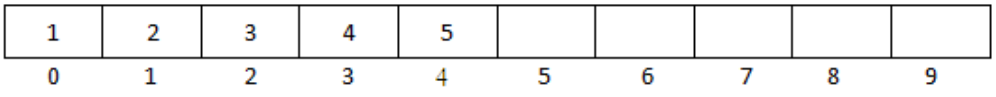
Queue after insertion of a new elemet

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

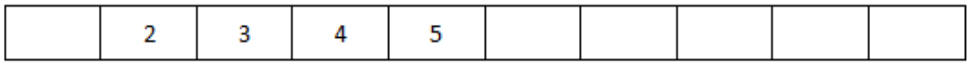
Delete Operation

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.



QUEUE

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



Queue after deletion of a n element

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

b) Convert the expression into prefix and postfix notations
A * B – C / D - E

5

Prefix expression: A * B – C / D – E = A B * C D / E – –

Postfix expression: A * B – C / D – E = – – * A B / C D E

OR

IV. Explain the algorithm for conversion of infix expression to postfix notation 15

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

In both algorithms, we do same thing that is,

- Scan the Infix string from left to right.
- Initialize an empty stack.
- If the scanned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty push the character to stack.
    - If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.
    - Repeat this step till all the characters are scanned.
- (After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
- Return the Postfix string.

UNIT – II

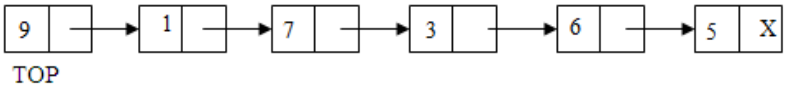
V. Explain about linked stack and write the algorithm to implement a linked stack 15

Stack is an important data structure which stores its elements in an ordered manner. A stack is a linear data structure which can be implemented by either using an array or a linked list. In case the stack is a very small one or its maximum size is known in advance, then the array implementation gives an efficient implementation. But if array size

cannot be determined in advance, the other alternative is linked representation. The elements in a stack are added and removed only from one end, which is called TOP. Hence, a stack is called LIFO (Last-In-First-Out) data structure, as the element that was inserted last is the first one to be taken out.

Every stack has a variable called TOP associated with it. TOP is used to store the address of topmost element of the stack. It is in the position where the element will be added or deleted. There is another variable called MAX, which is used to store the maximum number of elements that the stack can hold.

If TOP = NULL, then it indicate that the stack is empty and if TOP = MAX – 1, then the stack is full.

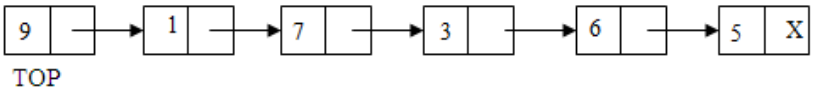


OPERATIONS ON STACK

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

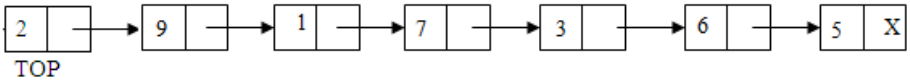
The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. However, before inserting the value we must check if *TOP = MAX – 1*, because if that is the case, then the stack is full and no more insertions can be further done. If an attempt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



Algorithm

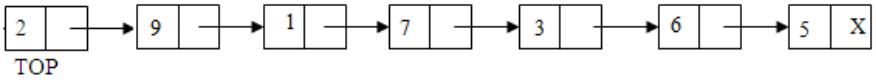
Step 1: Allocate memory for the new node and name it as New_Node
Step 2: SET New_Node->DATA = VAL
Step 3: IF TOP = NULL, then
SET New_Node->NEXT = NULL
SET TOP = New_Node
ELSE
SET New_Node->NEXT = TOP
SET TOP = New_Node
[END OF IF]
Step 4: END

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

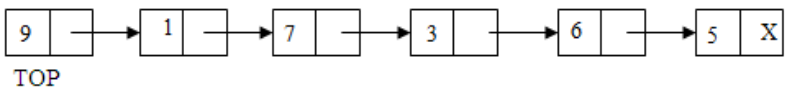
The pop operation is used to delete the topmost element from the stack. However before deleting the value, we must first check if TOP = NULL because if that is the case, the it means the stack is empty and no more deletions can further be done. If an attempt is made to delete a value from a stack that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



Algorithm

Step 1: IF TOP = NULL then
PRINT "UNDERFLOW"
[END OF IF]
Step 2: SET 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



OR

VI.

a) Explain the algorithm to implement polynomial addition using singly 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]

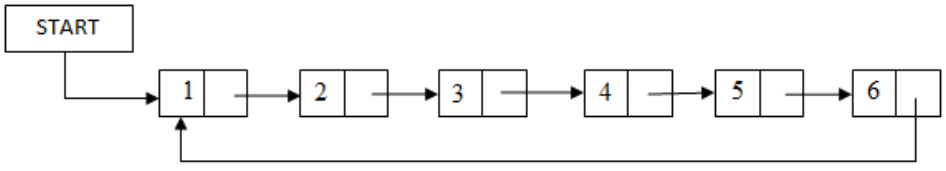
[END OF IF]

Step 3: copy the remaining terms from the non-empty polynomial into the sum Polynomial

Step 4: End

b) Discuss about circular linked list 5

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.

Insertion

To do insertion we will take two cases

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.

Deletion

To do deletion we will take two cases

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.

UNIT – III

VII. Explain tree traversal algorithm with suitable examples 15

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

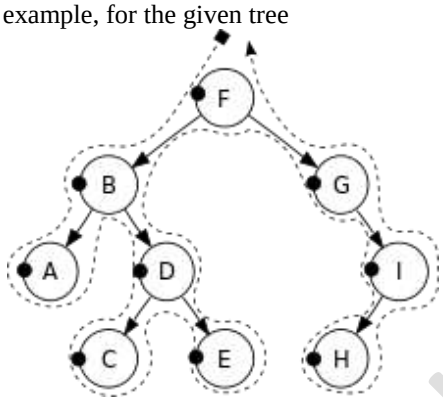
To traverse non empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by:

- 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

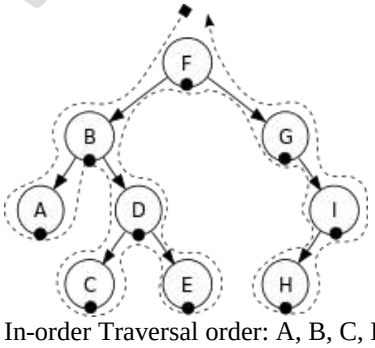
In-order Algorithm

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 pre-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:

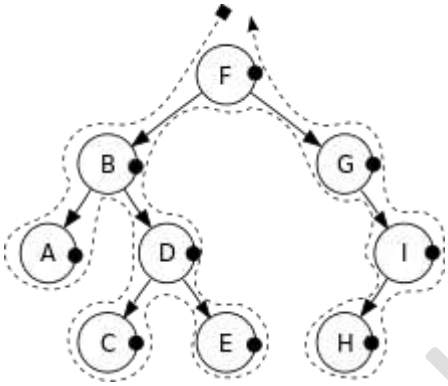
- Traverse the right sub-tree.
- Visit the root.
- Traverse the left sub-tree.

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

OR

VIII. Explain the algorithm for the following operations in BST
a) Insertion b) Deletion c) Search

15

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
```

Deletion

```
Step 1:  IF TREE = NULL, then
                Write "VAL not found in the tree"
           ELSE IF VAL < TREE->DATA
```

```
            Delete (TREE->LEFT, VAL)
        ELSE IF VAL > TREE->DATA
            Delete (TREE->RIGHT, VAL)
        ELSE IF TREE->LEFT AND TREE->RIGHT
            SET TEMP = findLargestNode (TREE->LEFT)
            SET TREE->DATA = TEMP->DATA
            Delete (TREE->LEFT, TEMP->DATA)
        ELSE
            SET TEMP = TREE
            IF TREE->LEFT = NULL AND TREE->RIGHT = NULL
                SET TREE = NULL
            ELSE IF TREE->LEFT!= NULL
                SET TREE = TREE->LEFT
            ELSE
                SET TREE = TREE->RIGHT
            FREE TEMP
      [END OF IF]
Step 2: End
```

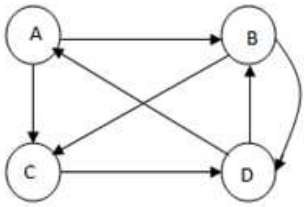
Search

```
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
```

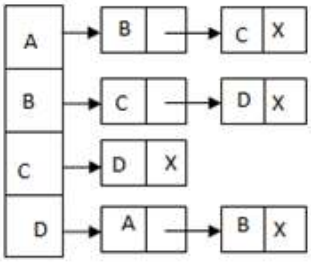
UNIT – IV

IX.

a) Prepare the adjacency matrix and adjacency list for the given graph 8



Adjacency list



Adjacency matrix

	A	B	C	D
A	0	1	1	0
B	0	0	1	1
C	0	0	0	1
D	1	1	0	0

b) Explain the binary search algorithm 7

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.

OR

X.

a) Compare bubble sort and quick sort algorithms

10

Bubble Sort: The simplest sorting algorithm. It involves the sorting the list in a repetitive fashion. It compares two adjacent elements in the list, and swaps them if they are not in the designated order. It continues until there are no swaps needed. This is the signal for the list that is sorted. It is also called as comparison sort as it uses comparisons.

Bubble sort is easy to program, slower, iterative. Compares neighboring numbers swaps it if required and continues this procedure until there are no more swaps

1. For each element A in the array
2. ----For each element B in the array
3. --------If the smaller element of A and B isn't on the left, swap the two
4. ----If there haven't been any swaps done on this run, we're done.

Quick Sort: The best sorting algorithm which implements the 'divide and conquer' concept. It first divides the list into two parts by picking an element a 'pivot'. It then arranges the elements those are smaller than pivot into one sub list and the elements those are greater than pivot into one sub list by keeping the pivot in its original place.

Quick Sort is little difficult to program, Fastest, Recursive. Pivot number is selected, other numbers are compared with it and shifted to the right of number or left depending upon criteria again this method is applied to the left and right list generated to the pivot point number. Select pivot point among that list.

1. Choose partition element in array
2. Ensure that only elements smaller than partition are on the left
3. Recurse on left, right sub-arrays that contain lesser, greater elements respectively

Bubble sort does it's work by bubbling the lowest value to the top or sinking the highest to the bottom (depending on how it is written). It's slow for large numbers of items. But even with large number of elements it can be reasonable if the items are almost sorted. Quick sort does its work by getting a pivot, then Quick sorting elements to the left and quick sorting elements to the right. It's usually fast but can be slow if the items are already sorted (depending on how it is written).

Bubble sort is a very simple algorithm which compares two neighboring elements and swaps them if they are not in the right order. In a worst case scenario when the list is already sorted in reverse order, bubble sort will make n^2 comparisons. Quick Sort is slightly more complex algorithm that divides the array into two parts using a pivot point and recursively sorts sub arrays. Both BS & QS can be implemented as in-place-sort without needing temp memory space, so they are comparable in terms of memory requirements. So in the end key difference is performance Avg & Worst case performance of bubble sort O(n^2)Best case performance of bubble sort is O(n) - when the list is already sorted. Performance of quick sort is O(n log(n))

b) List the applications of graph

5

- In circuit networks where points of connections are drawn as vertices and component wires become the edges of the graph.
- In maps that draw cities/states/regions and adjacency relations as edges.

- The transport networks where stations are drawn as vertices and routers become the edges of the graph
- In program flow analysis where procedures and modules are treated as vertices and calls to these procedures are drawn as edges of graph
- Graphs can be used for finding shortest path, project planning, etc