TED (10)-3071 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/
TECHNOLIGY- OCTOBER, 2013

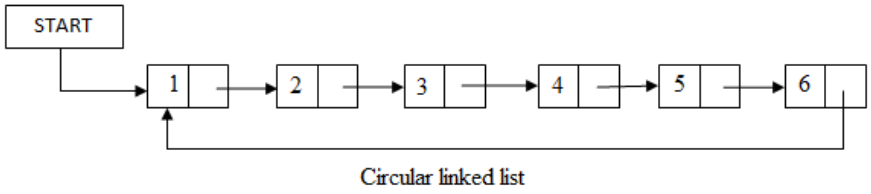
DATA STRUCTURE

(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. Formulate the prefix notation of (a + b) * (c – d)
   (a + b) * (c – d) = [+ab] * [– cd]
   = * + ab – cd
2. State any two drawbacks when arrays are used to create Data structures.
   We cannot add any number of elements in Data structures when arrays are used.
   We must store data in consecutive memory locations in the case of array
3. Draw the pictorial representation of a Circular Linked List



Circular linked list

4. Distinguish between a tree and a graph.
   Tree is a Data structure which defined as a collection of nodes, every node contain left pointer, right pointer and data.
   Graph is an abstract data structure that is the collection of vertices (nodes) and edges that connect these vertices.
5. Define the term Degree of a Node in a Directed Graph
   The degree of a node, written as deg (u) is equal to the sum of in-degree and out-degree of that node. Therefore deg (u) = indeg (u) + outdeg (u)

PART – B

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

1. Summarize the basic Data structure operations.
   The basic Data structure operations are
   - Insertion : Inserting a data element to a Data structure
   - Deletion: Deleting a data element to a Data structure
   - Traversal :Means accessing each and every element from a Data structure
   - Search : Searching an element in Data structure
   - Insert at front : Inserting a data element to a Data structure in its first position
   - Insert after : Inserting a data element to a Data structure in to given position
   - Delete at front : Delete an element from first position of a Data structure
   - Delete given data : Deleting a data element of a Data structure from given position
   - Sorting : Sorting the elements of a Data structure either in ascending or descanting order
2. Write short note on 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)

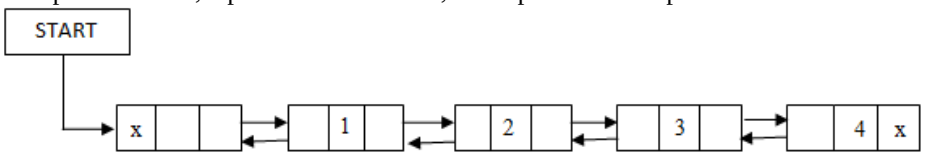
   A priority queue is somewhat similar to a queue, with an important distinction: each item is added to a priority queue with a priority level, and will be later removed from the queue with the highest priority element first. That is, the items are (conceptually) stored in the queue in priority order instead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

   Create a priority queue. The queue must support at least two operations:

   1. Insertion. An element is added to the queue with a priority (a numeric value).

   Top item removal. Deletes the element or one of the elements with the current top priority and return it.
3. Write short note on 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.

START
1

	DATA	PREV	NEXT
1	H	-1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	-1

In figure we see that a variable START is used to store the address of the first node. Here in this example, START = 1, so the first data is stored at address 1, which is H. since this is the first node, it has no previous node and hence stores NULL or -1 in the prevfield. We will traverse the list until we reach a position where the NEXT entry contains -1 or NULL. This denotes the end of the linked list, that is, the node that contains the address of the first node is actually the last node of the list. When we traverse the DATA and NET in this manner, we will finally see that the linked list in the above example stores characters that when put together forms the word HELLO.

Insertion

To do insertion we will take five cases

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.
- Case 3: The new node is inserted after a given node.
- Case 4: The new node is inserted before a given node.
- Case 5: The new node is inserted in a sorted linked list.

Deletion

To do deletion we will take five cases

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.
- Case 3: The node after a given node is deleted.
- Case 4: The node before a given node is deleted.

Case 1: The node is deleted from assorted linked list.

4. Write the algorithm to delete the head Node of singly Linked List

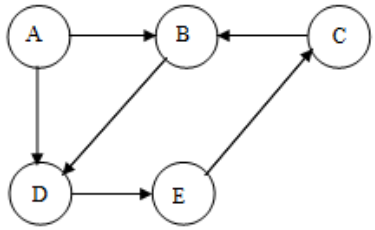
Step 1: IF START = NULL, then
Write “UNDERFLOW”

Go to step 5
[END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START->NEXT
Step 4: FREE PTR
Step 2: EXIT

5. Write the algorithm to find the sum of all elements stored in a BST

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. Draw a Directed Graph and create its adjacency matrix



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

7. Compare the complexity of Quick sort in worst case and average case

In average case the running time of quick sort can be given as o (n log n). The partitioning of the array which simply loops over the elements of the array once uses O(n) times.
Practically the efficiency of quick sort depends on the element which is chosen as pivot. Its worst case efficiency is given as O (n^2). The worst case occurs when the array is already sorted(either in ascending or descending order) and the leftmost element is chosen as pivot.

PART – C

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

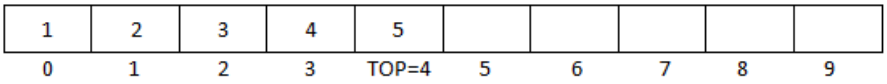
UNIT – I

III. Explain on stack and the algorithm to implement its operations using Array 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. 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 TOPassociated 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.



Stack

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: IF TOP = MAX - 1 then
PRINT "OVERFLOW"

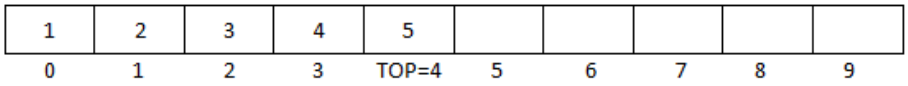
[END OF IF]

Step 2: SET TOP = TOP + 1

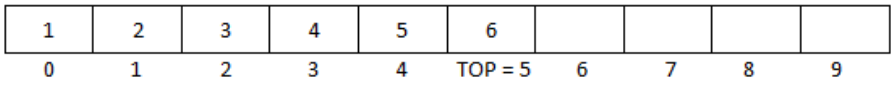
Step 3: SET STACK [TOP] = VALUE

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*



Stack before insertion



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"

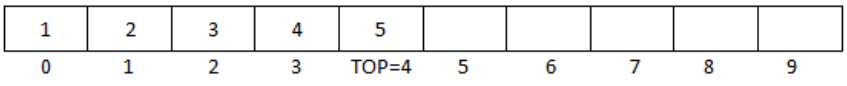
[END OF IF]

Step 2: SET VAL = STACK [TOP]

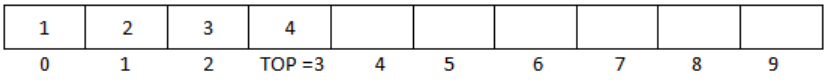
Step 3: SET TOP = TOP - 1

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



Stack before Deletion



Stack after deletion

OR

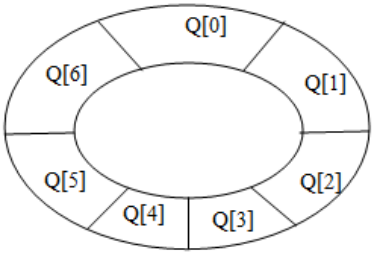
IV.

(a) Summarize the application of Queue 8

- Priority queues of process
  Priority queues are widely used in operating system to execute the highest priority process first. The priority of the process may be set based on the CPU time. It requires to get executed completely. For example, there are 3 processes, where the first process needs5 ns to complete, the second process needs 4ns and the 3rd process needs 7ns, then the second process will have the highest priority and will thus be the first to be executed. However, CPU time is not the only factor that determines the priority, rather than it is just one among several factors.
- Online reservation for railway and other
  The first booking person gets preference. This reservation process taking places in a queue manner. The first booking person gets first reserved. The last booking maintaining in the waiting
- 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

(b) Explain the advantages of Circular Queue over linear queue 7

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.



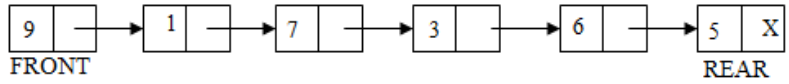
By using circular queue, the disadvantage of linear queue-compaction can be avoided. A circular queue is implemented in the same manner as a linear queue is implemented.

UNIT – II

V. Explain about linked queue and write the algorithm to implement a Linked Queue 15

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

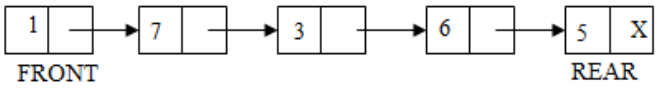
In a linked queue, every element has two parts, one that stores the data and another that stores the address of next element. The START pointer of linked list is used as the FRONT. Here, we will also use another pointer called REAR, which will store the address of last element in the queue. All insertions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



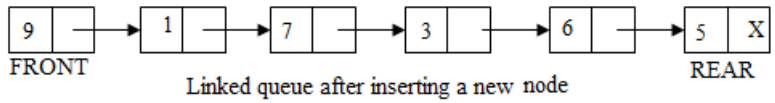
A Queue has two basic operations: *insert* and *delete.* The insert operation adds an element to the end of queue of the stack and the delete operation removes the element from the front or start of the queue.

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



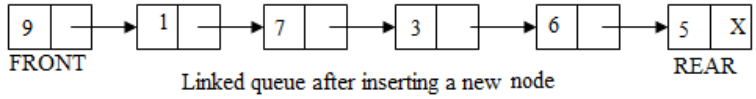
Linked queue after inserting a new node

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

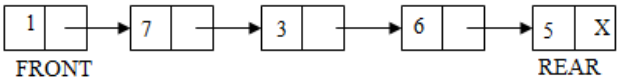
Delete Operation

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

OR

VI.

(a) Develop an algorithm to count the occurrence of a given data item in a singly linked list 10

Algorithm

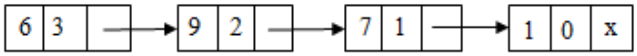
Step 1: [INITIALIZE] SET PTR = START
Step 2: [INITIALIZE] SET COUNT = 0
Step 3: Repeat Sep 4 while PTR ! = NULL
Step 4: IF VAL = PTR->DATA
SET COUNT = COUNT + 1
ELSE
SET PTR = PTR->NEXT
[END OF IF]
[END OF LOOP]
Step 5: Write COUNT
Step 6: EXIT

In step 1 we initialize a pointer variable PTR with START that contains the address of the first node and COUNT with 0. In step 2 a while loop is executed which will compare every node's DATA with the VAL for which the search is being made. In every successful search, that is the VAL has been found, then the variable COUNT will be incremented by 1. At last the number of occurrence which is stored in CONUT will be printed.

(b) Draw the linked representation of a Polynomial. 5

Polynomial: $6x^3 + 9x^2 + 7x + 1$

Linked representation:



UNIT – III

VII. Explain the binary tree traversal algorithms 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:

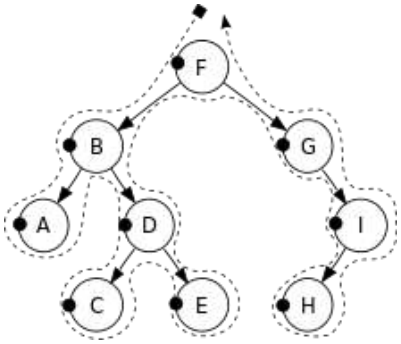
- Visit the root.
- Traverse the left sub-tree.
- Traverse the right sub-tree.

Pre-order traversal is also called *depth-first traversal.* In this algorithm, the left sub-treealways 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:

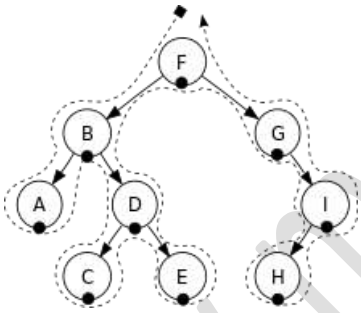
- Traverse the left sub-tree.
- Visit the root.
- Traverse the right sub-tree.

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:

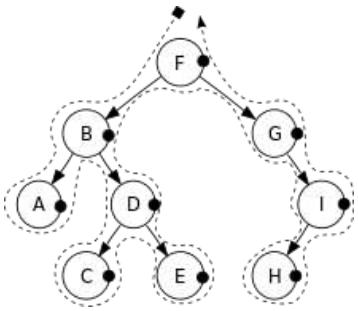
- Traverse the right sub-tree.
- Visit the root.
- Traverse the left sub-tree.

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

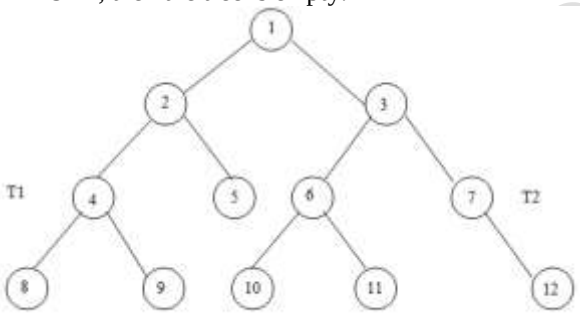
OR

VIII.

(a) Distinguish between Binary tree and Threaded binary trees. 8

Binary trees

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.

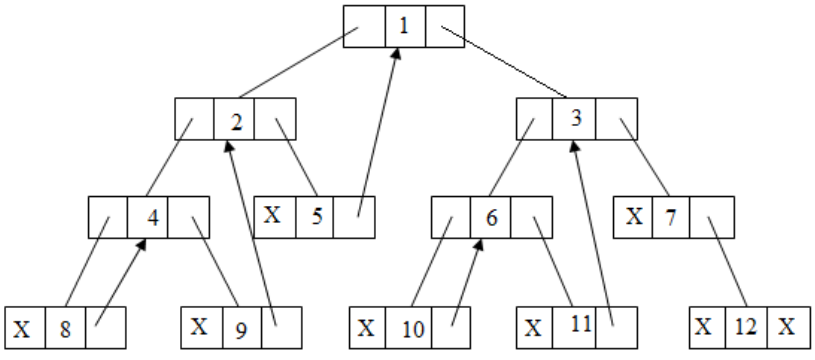


If the root node is R is not null, then the two trees T_1 and T_2 are called the left and right sub-trees of R. if T_1 is not empty, then T_1 is called left successor of R. likewise if T_2 is not empty, then itis called left successor of R.

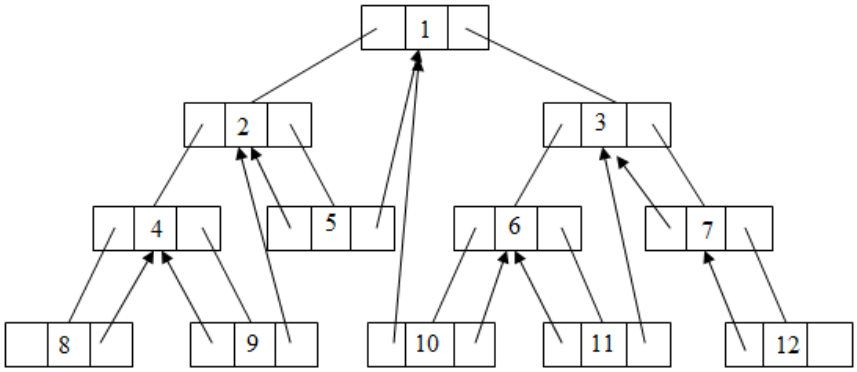
In above figure, node 2 is left successor and 3 is the right successor of the root node 1. Left sub-tree of the root node consist of the nodes 2, 4, 5, 8 and 9. Similarly, the right sub-tree of the root node consist of the nodes 3, 6, 7, 10, 11 and 12

Threaded binary trees

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 peace 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

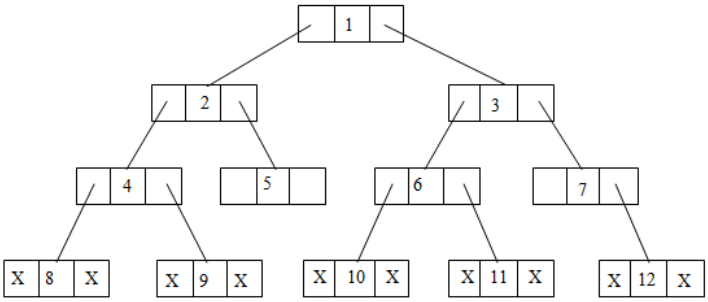


Linked representation of the binary tree with two-way threading

(b) Explain on Linked representation of Binary Tree with example. 7

In the linked representation of binary tree, every node have three parts: the data element, a pointer to the left node, and a pointer to the right node.

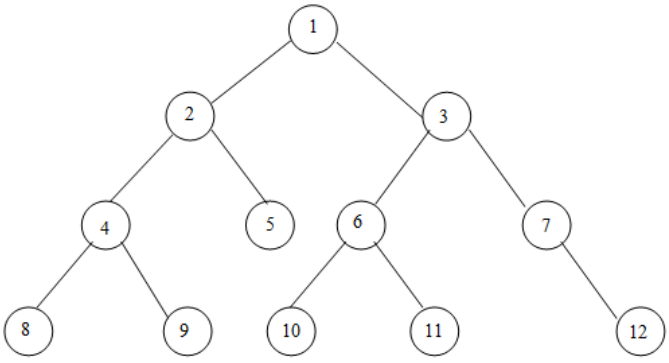
```
struct node {
    struct node *left;
    int data;
    struct node *right;
    };
```



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

ROOT

3

15

AVAIL

	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

UNIT – IV

IX. Explain Warshall's algorithm to find shortest path of weighted graph 15

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 $P_0, P_1, P_2, \ldots \ldots P_n$ 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

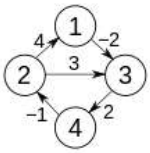
From above discussion we can conclude that $P_k[i][j]$ is equal to 1 only when either of the two following cases occur:

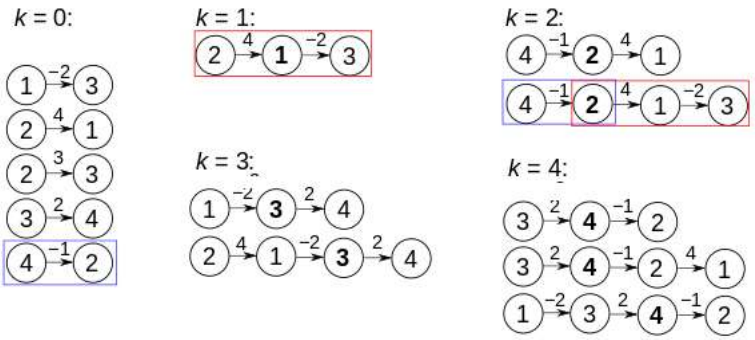
- There is a path from v_i to v_j that does not use any other node except $v_1, v_2, \ldots V_{k-1}$. Therefore
  $P_{k-1}[i][j] = 1$
- There is a path from v_i to v_k and a path from v_k to v_j where all the nodes use $v_1, v_2, \ldots V_{k-1}$. Therefore
  $P_{k-1}[i][k] = 1$ AND $P_{k-1}[k][j] = 1$

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

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

For example, for the below graph





Prior to the first iteration of the outer loop, labeled *k*=0 above, the only known paths correspond to the single edges in the graph. At k=1, paths that go through the vertex 1 are found: in particular, the path 2→1→3 is found, replacing the path 2→3 which has fewer edges but is longer. At k=2, paths going through the vertices {1,2} are found. The red and blue boxes show how the path 4→2→1→3 is assembled from the two known paths 4→2 and 2→1→3 encountered in previous iterations, with 2 in the intersection. The path 4→2→3 is not considered, because 2→1→3 is the shortest path encountered so far from 2 to 3. At k=3, paths going through the vertices {1,2,3} are found. Finally, at k=4, all shortest paths are found.

OR

X. Compare Binary search Algorithm and Linear search algorithm in searching for occurrence of element 45 in the set of values {20, 5, 35, 15, 50,75, 25, 30, 10, 12} 15

45 in the set of values {20, 5, 35, 15, 50, 75, 25, 30, 10, 12}

A **binary search** algorithm finds the position of a specified input value (the search "key") within an array sorted by key value. For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned

- So for checking the occurrence of 45 in the set of values {20, 5, 35, 15, 50, 75, 25, 30, 10, 12} we 1st want to sort the list. So before applying binary searching algorithm, apply a sorting algorithm here.
- Applying sorting algorithm we get a list such as {5, 10, 12, 15, 20, 25, 30, 35, 50, 75}
- When using binary searching algorithm we have to check the element 45 with the values greater than pivot (25).
- In next step the pivot will be 35, our search key is not equal to and greater than pivot. So the steps will repeats, after a few iterations the searching will be complete and search key will not be found.
- 4 checking is required after sorting

Linear searching

- In linear search we compare each and every element with the search key from first to last.
- There is no need of sorting for these set of elements
- 10 checking are required
- Here we compare the search key with 1st element, i.e. 20. And the comparison continues until checking last element in the set, because of the search key is not present in the list.