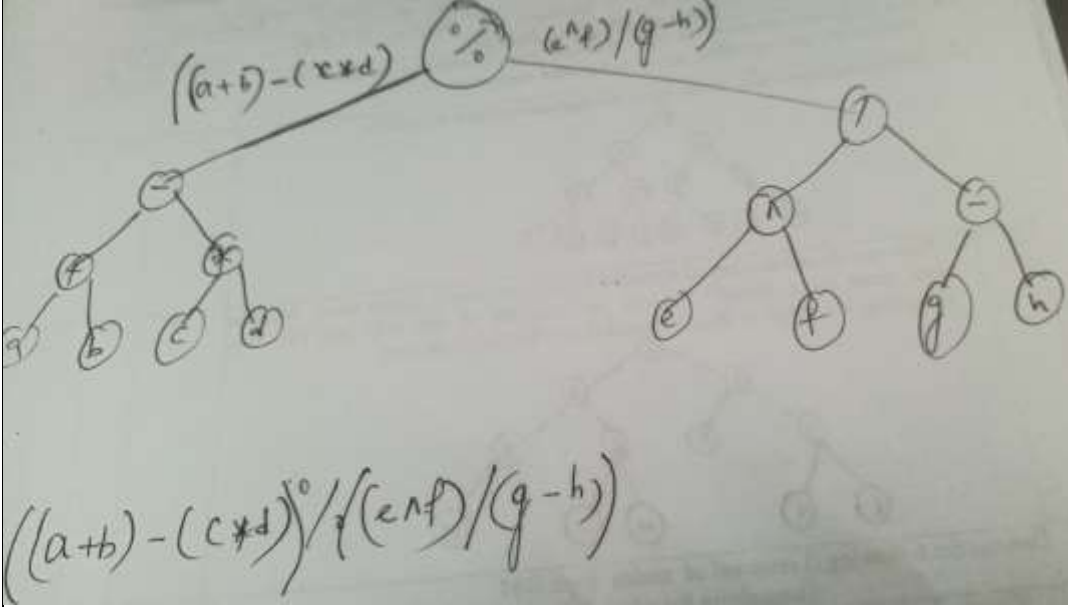
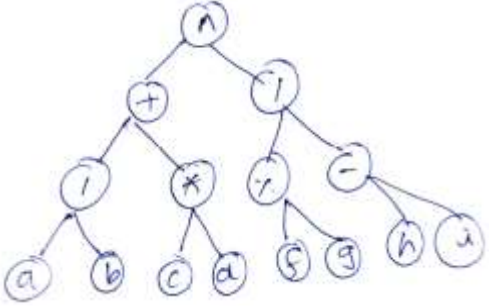
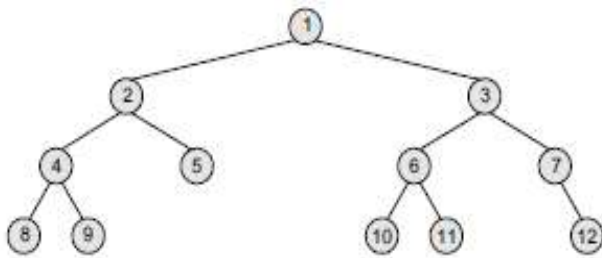


**Internal Assessment Test III–Feb 2023**

**Answer key**

Sub:	Data Structures and Applications	SubCode:	21CS32	Branch:	ISE
Date:	6.2.2023	Duration:	90min's	MaxMarks:	50
Sem/Sec:					III A, B &C
Answer any FIVE FULL Questions					OBE
	MARKS	CO	RBT		
1 a	For the given expression, $((a+b)-(c*d))^{\wedge}((e^{\wedge}f)/(g-h))$ , construct the binary tree. Find out the corresponding prefix and postfix expression.	6	CO4	L2	
 <p>Prefix: <math>\%^{-+}ab^*cd^{\wedge}ef-gh</math>                  Postfix: <math>ab+cd^*-df^{\wedge}gh-/\%</math></p>					
1b	Define expression tree. Write down the expression that it represents:	4	CO4	L1	
 <p><b>expression tree :</b>                  The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand  <math>(a/b)+(c*a)^{\wedge}(f\%g)/(h-i)</math></p>					
2	What are the advantages of threaded binary tree over a binary tree?. What do you mean by a thread? Explain the concepts of one way and two way threading. Construct a double threaded tree for the given tree:	10	CO4	L2	

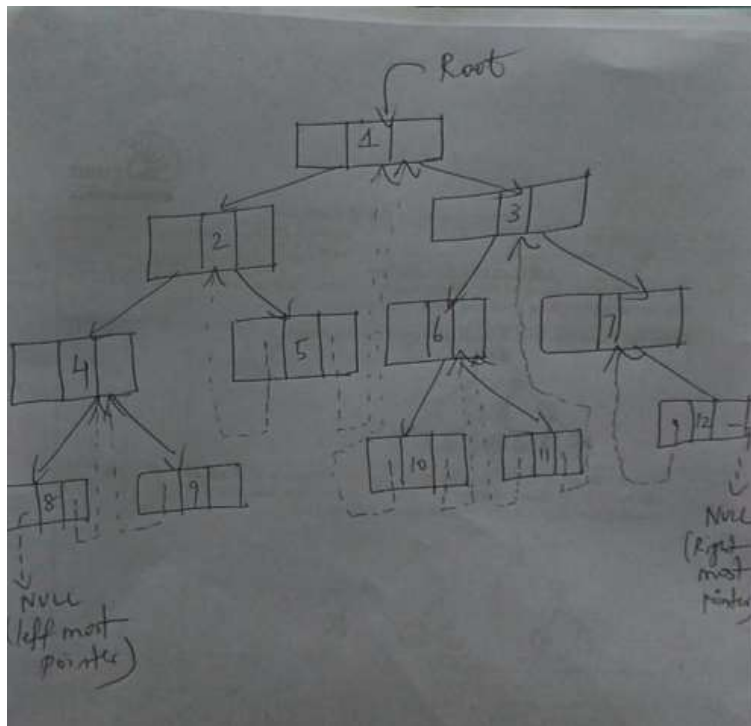


**Solution:**

In a Threaded Binary Tree, the nodes will store the in-order predecessor/successor instead of storing NULL in the left/right child pointers.

So the basic idea of a threaded binary tree is that for the nodes whose right pointer is null, we store the in-order successor of the node (if-exists), and for the nodes whose left pointer is null, we store the in-order predecessor of the node(if-exists).

One thing to note is that the leftmost and the rightmost child pointer of a tree always points to null as their in-order predecessor and successor do not exist.



**Types of Threaded Binary tree**

There are two types of Threaded Binary Trees:

- Single-Threaded Binary Tree
- Double-Threaded Binary Tree

**1. Single-Threaded Binary Tree**

In this type, if a node has a right null pointer, then this right pointer is threaded towards the in-order successor's node if it exists.

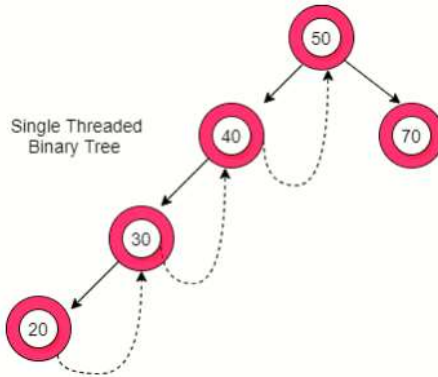
**Node Structure of Single-Threaded Binary Trees:** The structure of a node in a binary threaded tree is quite similar to that of a binary tree, but with some modifications. In threaded binary trees, we need to use extra boolean variables in the node structure. For single-threaded binary trees, we use only the **rightThread** variable.

```

struct Node{
    int value;
    Node* left;
    Node* right;
    bool rightThread;
}

```

The following diagram depicts an example of a Single-Threaded Binary Tree. Dotted lines represent threads.



## 2. Double-Threaded Binary Tree

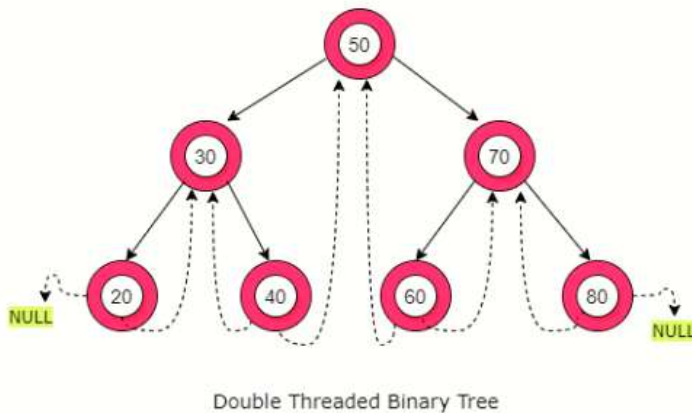
In this type, the left null pointer of a node is made to point towards the in-order predecessor node and the right null pointer is made to point towards the in-order successor node.

**Node Structure of Double-Threaded Binary Trees:** For the double-threaded binary tree, we use two boolean variables: **rightThread** and **leftThread**

```

struct Node{
    int value;
    Node* left;
    Node* right;
    bool rightThread;
    bool leftThread;
}

```



## Advantages of Threaded Binary Tree

Let's discuss some advantages of a Threaded Binary tree.

- **No need for stacks or recursion:** Unlike binary trees, threaded binary trees do not require a stack or recursion for their traversal.
- **Optimal memory usage:** Another advantage of threaded binary tree data structure is that it decreases memory wastage. In normal binary trees, whenever a node's left/right pointer is NULL, memory is wasted. But with

	<p>threaded binary trees, we are overcoming this problem by storing its inorder predecessor/successor.</p> <ul style="list-style-type: none"> <li>○ <b>Time complexity:</b> In-order traversal in a threaded binary tree is fast because we get the next node in <math>O(1)</math> time than a normal binary tree that takes <math>O(\text{Height})</math>. But insertion and deletion operations take more time for the threaded binary tree.</li> <li>○ <b>Backward traversal:</b> In a double-threaded binary tree, we can even do a backward traversal.</li> </ul> <h2>Disadvantages of Threaded Binary tree</h2> <p>Let's discuss some disadvantages that might create a problem for a programmer using this.</p> <ul style="list-style-type: none"> <li>○ <b>Complicated insertion and deletion:</b> By storing the inorder predecessor/successor for the node with a null left/right pointer, we make the insertion and deletion of a node more time-consuming and a highly complex process.</li> <li>○ <b>Extra memory usage:</b> We use additional memory in the form of <i>rightThread</i> and <i>leftThread</i> variables to distinguish between a thread from an ordinary link. (However, there are more efficient methods to differentiate between a thread and an ordinary</li> <li>○</li> </ul>			
3	<p>Discuss the following i) removal of nodes from BST ii) searching for a key in BST with C function</p> <p><b>CODE:</b></p> <pre> struct node *delete (struct node *root, int key) {     if (root == NULL)     {         return root;     }     if (key &lt; root-&gt;data)     {         root-&gt;left=delete (root-&gt;left, key);     }     else if (key &gt; root-&gt;data)     {         root-&gt;right=delete (root-&gt;right, key);     }     else     {         struct node *temp;         if (root-&gt;left == NULL&amp;&amp; root-&gt;right == NULL)         {             temp=root;             root=NULL;             free(temp);         }         else if (root-&gt;left == NULL)         {             temp = root;             root=root-&gt;right;             free(temp);         }         else if (root-&gt;right == NULL)         { </pre>	10	CO4	L3

```

temp = root;
root=root->left;
free(temp);
}
else
{
temp = smallest_node(root->right);
root->data = temp->data;
root->right = delete (root->right, temp->data);
}
}
return root;
}

```

// search the given key node in BST

```

int search(int key)
{
struct node *temp = root;

while (temp != NULL)
{
if (key == temp->data)
{
return 1;
}
else if (key > temp->data)
{
temp = temp->right;
}
else
{
temp = temp->left;
}
}
return 0;
}

```

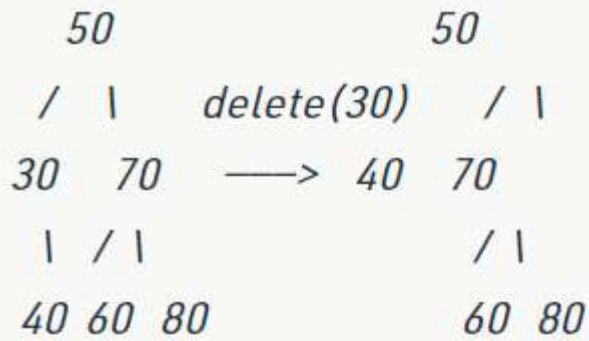
**1) Node to be deleted is the leaf:** Simply remove it from the tree.

```

      50                50
     / \   delete(20) / \
    30 70  —>  30 70
   / \ / \       | / \
  20 40 60 80    40 60 80

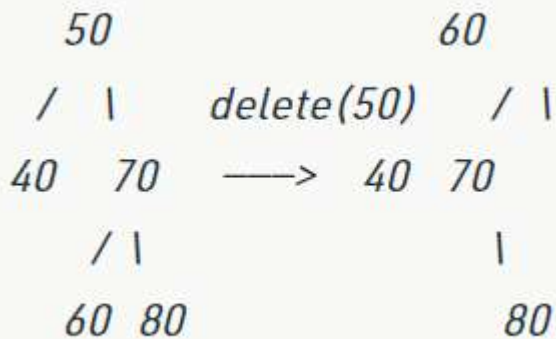
```

**2) Node to be deleted has only one child:** Copy the child to the node and delete the child



**3) Node to be deleted has two children:** Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor.

**Note:** Inorder predecessor can also be used.



4a

Construct a binary tree from the traversal order given below:

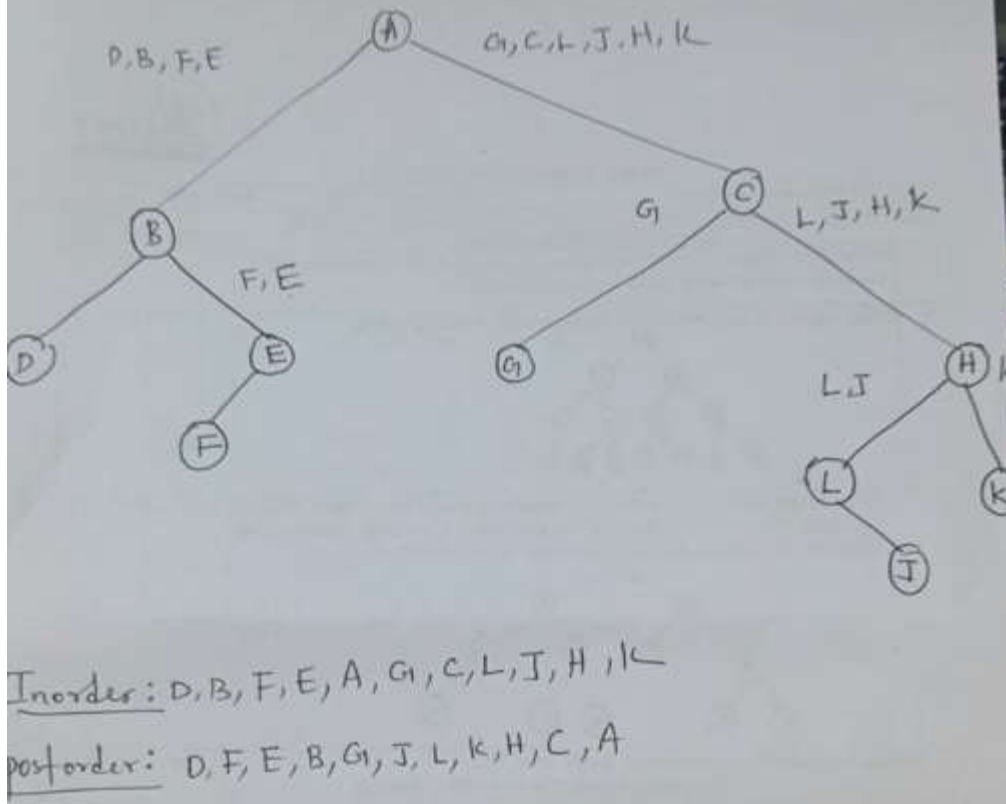
POSTORDER	D	F	E	B	G	J	L	K	H	C	A
INORDER	D	B	F	E	A	G	C	L	J	H	K

**Solution:**

4

CO4

L2



4b

Describe the process of construction of BST with an example and a C routine.

6

CO4

L3

Solution:

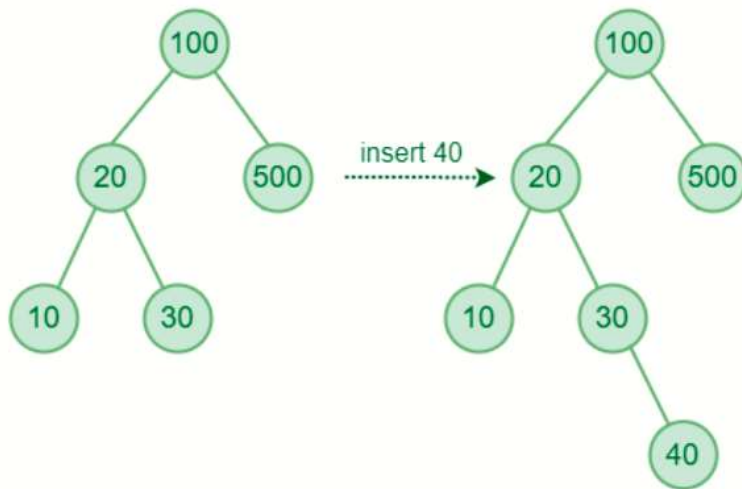
A binary Search Tree is a special type of binary tree data structure that has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

### Insert a value in a Binary Search Tree:

A new key is always inserted at the leaf by maintaining the property of the binary search tree. We start searching for a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node. The below steps are followed while we try to insert a node into a binary search tree:

- Check the value to be inserted (say **X**) with the value of the current node (say **val**) we are in:
  - If **X** is less than **val** move to the left subtree.
  - Otherwise, move to the right subtree.
- Once the leaf node is reached, insert **X** to its right or left based on the relation between **X** and the leaf node's value.



### Source Code:

```

struct node *create_node(int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));

    if (new_node == NULL)
    {
        printf("\nMemory for new node can't be allocated");
        return NULL;
    }

    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;

    return new_node;
}

// inserts the data in the BST
void insert(int data)
{
    struct node *new_node = create_node(data);

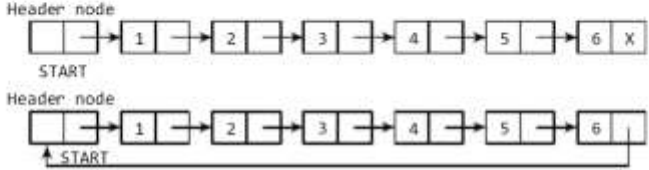
    if (new_node != NULL)
    {
        // if the root is empty then make a new node as the root node
        if (root == NULL)
        {
            root = new_node;
            printf("\n* node having data %d was inserted\n", data);
            return;
        }

        struct node *temp = root;
        struct node *prev = NULL;

        // traverse through the BST to get the correct position for insertion
        while (temp != NULL)
        {
  
```



	<pre> prev = temp; if (data &gt; temp-&gt;data) {     temp = temp-&gt;right; } else {     temp = temp-&gt;left; } }  // found the last node where the new node should insert if (data &gt; prev-&gt;data) {     prev-&gt;right = new_node; } else {     prev-&gt;left = new_node; }  printf("\n* node having data %d was inserted\n", data); } } </pre>			
--	---	--	--	--

5	<p>Define header list. Explain its types with diagram. Mention the two properties of circular header lists. Write and explain the algorithm to traverse a circular header lists.</p> <p><b>Solution:</b></p> <p><b>HEADER LINKED LISTS</b></p> <p>A header linked list is a special type of linked list which contains a header node at the beginning of the list. So, in a header linked list, <i>START</i> will not point to the first node of the list but <i>START</i> will contain the address of the header node. The following are the two variants of a header linked list:</p> <ul style="list-style-type: none"> <li>• <i>Grounded header linked list</i> which stores <i>NULL</i> in the next field of the last node.</li> <li>• <i>Circular header linked list</i> which stores the address of the header node in the next field of the last node. Here, the header node will denote the end of the list.</li> </ul> <p>Look at Fig. 6.65 which shows both the types of header linked lists.</p>  <p>The diagram illustrates two types of header linked lists. In the first, a 'Header node' (represented by a box with an arrow) points to a sequence of nodes containing values 1, 2, 3, 4, 5, and 6. The last node (6) has an arrow pointing to 'X', representing NULL. A 'START' pointer also points to the first node (1). In the second, the 'Header node' points to the same sequence of nodes (1-6). The last node (6) has an arrow pointing back to the 'Header node', forming a cycle. A 'START' pointer points to the first node (1).</p>	10	CO3	L2
---	---	----	-----	----

```

struct node *display(struct node *start)
{
    struct node *ptr;
    ptr=start;
    while(ptr!=NULL)
    {
        printf("\t %d", ptr->data);
        ptr = ptr->next;
    }
    return start;
}

```

6 Write short notes on i) Sparse Matrix representation using Header Lists

10

CO3

L3

**Solution:**

Sparse Matrices  
sparse Matrix Representation

→ we represent each column of a sparse matrix as a circularly linked list, with header node.

→ Similar representation for each row of  $r$  nodes and empty nodes.

next
down   right

Fig. Header node

→ links header nodes together

→ To link into a column list

→ To link into row list

row	col	val
down	right	

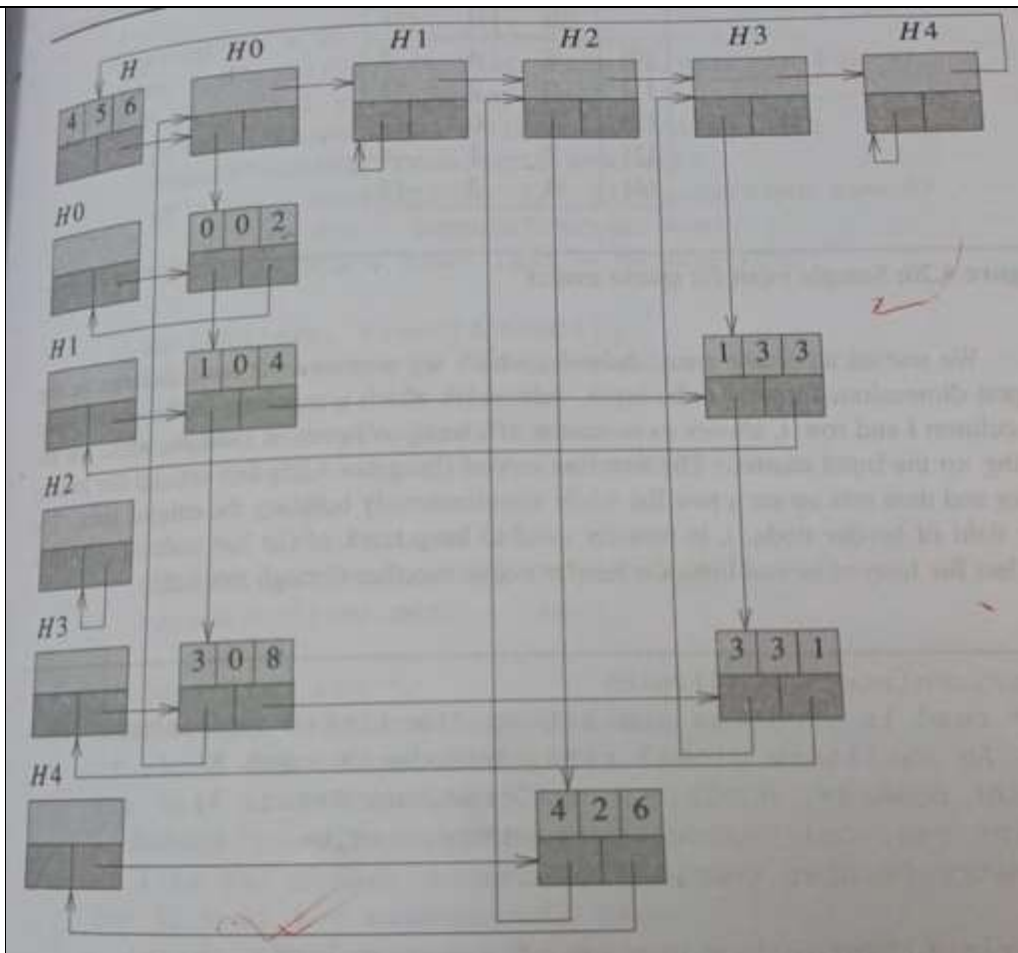
Fig. element node

→ To link to the next non-zero term in the same row.

→ To link to the next non-zero term in the same column.

→ Header node for row  $i$  also, the header node for column  $i$  and the total no. of header nodes is  $\max(\text{no. of rows, no. of columns})$

0	2	0	0	0	0
1	4	0	0	3	0
2	0	0	0	0	0
3	8	0	0	1	0
4	0	0	6	0	0



ii) Addition of two polynomials.

**Solution:**

```

typedef struct polyNode *polyPointer;
typedef struct {
    int coef;
    int expon;
    polyPointer link;
} polyNode;
polyPointer a,b;
..

```

We draw *polyNodes* as:

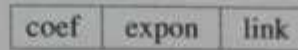
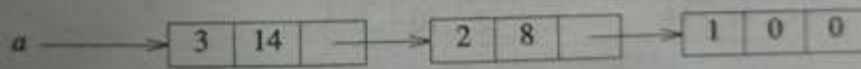


Figure 4.12 shows how we would store the polynomials

$$a = 3x^{14} + 2x^8 + 1$$

and

$$b = 8x^{14} - 3x^{10} + 10x^6$$



(a)



(b)

Figure 4.12: Representation of  $3x^{14} + 2x^8 + 1$  and  $8x^{14} - 3x^{10} + 10x^6$

