MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 4

Assignment Title: Create an inorder threaded binary search tree and perform the traversals.

Aim: Implement Inorder Threaded Binary Tree and its inorder traversal

Prerequisite:

1. Basic concepts of Threaded Binary Tree (TBT) and Binary Search Tree (BST).

Objectives:

1. Understand and implement Threaded Binary Trees (TBT) and concepts.
2. To understand how to make inorder traversal faster

Outcomes:

Upon Completion of the assignment the students will be able to

1. create and implement TBT
2. understand and analyse various operation of the TBT
3. understand how to traverse a binary tree without using stack and without recursion

Theory:

A binary tree is a data structure where every node can have up to two children. There are different ways of traversing through the tree, one of them is in-order traversal. In inorder traversal, you visit the nodes “in order”, which means if the binary tree were a binary search tree, an inorder traversal would give us the elements in an increasing order. However, to perform this traversal, we need to make use of recursion. If not recursion we need to use a stack to mimic the recursion. A Threaded Binary Tree lets us perform inorder traversal without the use of recursion or stack

What Is a Threaded Binary Tree?

In a threaded binary tree, either of the NULL pointers of leaf nodes is made to point to their successor or predecessor nodes

What is the need for a Threaded Binary Tree?

In a regular tree, the space complexity for insertion/ deletion can range from logarithmic to linear time. We get log time in a balanced binary tree and linear time is the worst case in an unbalanced tree. In applications where we have a constraint of space, or even when there is a use case of storing large trees in memory, we might need to save up on the extra space invested in the recursion depth. A threaded binary tree is advantageous in such scenarios where space usage is critical

The node structure for a threaded binary tree varies a bit and it's like this --

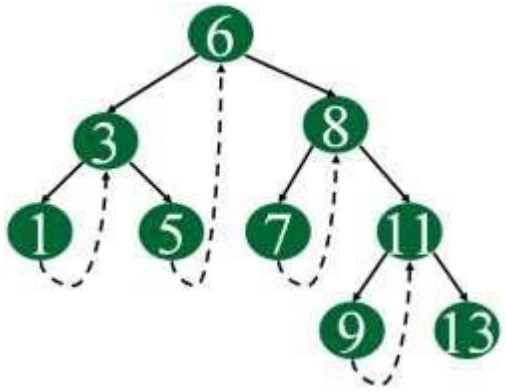
```
struct TBTNode{
        int lbit;
        TBTNode*lchild;
        char data;

        TBTNode*rchild;
        int rbit;
}
```

Node Structure

lbit (flag)	lchild (pointer)	Data	rchild (pointer)	rbit (flag)

There are two types of threaded binary trees.

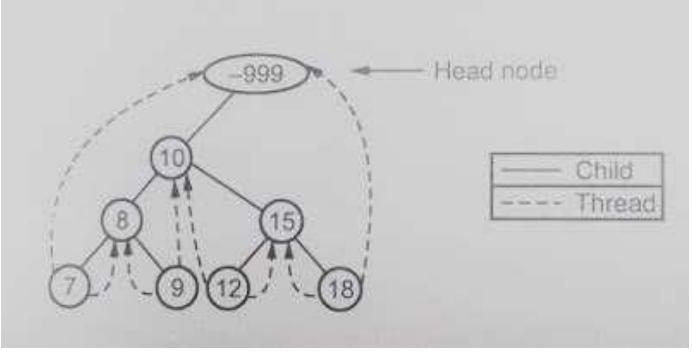
Single Threaded: Where a NULL right pointer is made to point to the inorder successor (if successor exists). Following diagram shows an example Single Threaded Binary Tree.



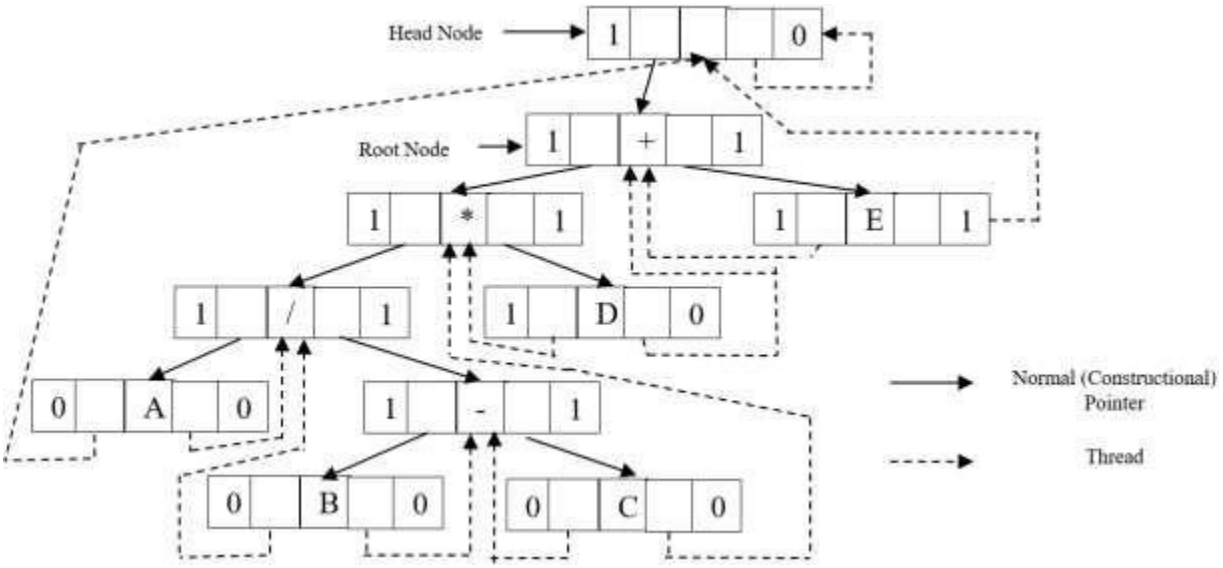
Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

From the Inorder traversal of TBT, the first nodes left pointer and last nodes right pointer is connected to the dummy node called as head node.

The threads are also useful for fast accessing ancestors of a node.The dotted lines represent threads.



Double Threaded TBT Node Representation in memory



Inorder Traversal – A/B-C*D+E
Preorder Traversal - +*?A-BCDE

Advantages of Threaded Binary Tree

- In this Tree it enables linear traversal of elements.
- It eliminates the use of stack as it performs linear traversal, so save memory.
- Enables to find parent node without explicit use of parent pointer
- Threaded tree give forward and backward traversal of nodes by in-order fashion
- Nodes contain pointers to in-order predecessor and successor
- For a given node, we can easily find inorder predecessor and successor. So, searching is much easier.
- In threaded binary tree there is no NULL pointer present. Hence memory wastage in occupying NULL links is avoided.
- The threads are pointing to successor and predecessor nodes. This makes us to obtain predecessor and successor node of any node quickly.
- There is no need of stack while traversing the tree, because using thread links we can reach to previously visited nodes.

Disadvantages of Threaded Binary Tree

- Every node in threaded binary tree need extra information (extra memory) to indicate whether its left or right node indicated its child nodes or its inorder predecessor or successor. So, the node consumes extra memory to implement.
- Insertion and deletion are way more complex and time consuming than the normal one since both threads and ordinary links need to be maintained.
- Implementing threads for every possible node is complicated.
- Increased complexity: Implementing a threaded binary tree requires more complex algorithms and data structures than a regular binary tree. This can make the code harder to read and debug.
- Extra memory usage: In some cases, the additional pointers used to thread the tree can use up more memory than a regular binary tree. This is especially true if the tree is not fully balanced, as threading a skewed tree can result in a large number of additional pointers.
- Limited flexibility: Threaded binary trees are specialized data structures that are optimized for specific types of traversal. While they can be more efficient than regular binary trees for these types of operations, they may not be as useful in other scenarios. For example, they cannot be easily modified (e.g. inserting or deleting nodes) without breaking the threading.
- Difficulty in parallelizing: It can be challenging to parallelize operations on a threaded binary tree, as the threading can introduce data dependencies that make it difficult to process nodes independently. This can limit the performance gains that can be achieved through parallelism.

Applications of threaded binary tree

- Expression evaluation: Threaded binary trees can be used to evaluate arithmetic expressions in a way that avoids recursion or a stack. The tree can be constructed from the inputexpression, and then traversed in-order or pre-order to perform the evaluation.
- Database indexing: In a database, threaded binary trees can be used to index data based on a specific field (e.g. last name). The tree can be constructed with the indexed values as keys, and then traversed in-order to retrieve the data in sorted order.
- Symbol table management: In a compiler or interpreter, threaded binary trees can be used to store and manage symbol tables for variables and functions. The tree can be constructed with the symbols as keys, and then traversed in-order or pre-order to perform various operations on the symbol table.
- Disk-based data structures: Threaded binary trees can be used in disk-based data structures (e.g. B-trees) to improve performance. By threading the tree, it can be traversed in a way that minimizes disk seeks and improves locality of reference.
- Navigation of hierarchical data: In certain applications, threaded binary trees can be used to navigate hierarchical data structures, such as file systems or web site directories. The tree can be constructed from the hierarchical data, and then traversed in-order or pre-order to efficiently access the data in a specific order.

Conclusion:

Threaded binary trees do not require a stack or recursion for their traversal.

Questions:

1. Create a Threaded binary trees using the following data values (Step wise)
   45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67
2. Write an algorithm for an Inorder Traversal of a Inorder threaded Binary Tree
3. What is the Time and Space Complexity of insertion, deletion, and searching operations in a threaded binary tree?

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

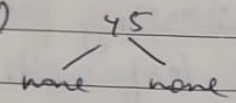
MIT-ADT UNIVERSITY
PUNE, INDIA

Name : Sourav Shailesh Tashildar.
Class : SY-1 (B)
Roll no. : 2213047
Subject : Advanced Data Structure Lab.

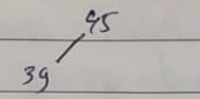
Assignment-4

1. Create a threaded binary tree using the following data values : 45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67

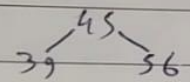
Ans (i)



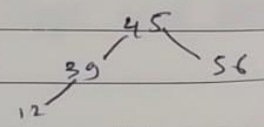
(ii)



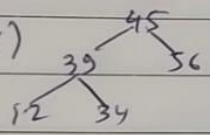
(iii)



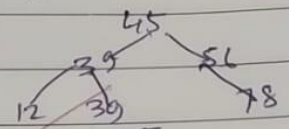
(iv)



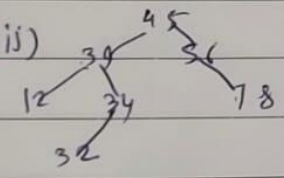
(v)



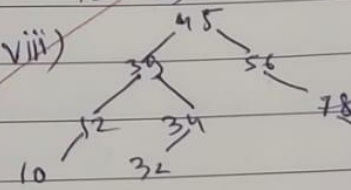
(vi)



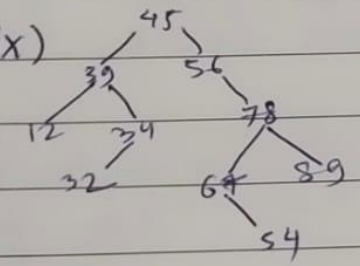
(vii)



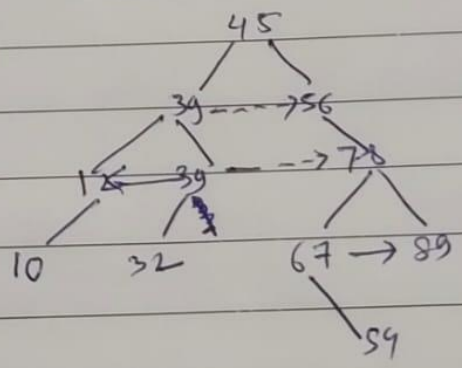
(viii)



(ix)



Threading



Inorder Traversal :

10, 12, 32, 34, 45, 54, 56, 67, 78, 89.

2) Write an algorithm for an Inorder Traversal of a Inorder TBT.

Ans ① Initialize a pointer curr to the leftmost node in the TBT
② While curr is not NULL:
a) Visit the current node.
b) If the right pointer of curr is NULL, curr to the node pointer to by the right threaded.
c) Otherwise, move to the leftmost node in the right subtree of curr.
③ stop.

3) What is the Time and Space complexity of Insertion, deletion and searching operation in TBT?

Ans ① Insertion:
Time complexity: O(log n)
Space complexity: O(1)
② Searching:
Time complexity: O(log n)
Space complexity: O(1)
③ Deletion:
Time complexity: O(log n)
Space complexity: O(1).

Code:

```
#include <iostream>
using namespace std;


class TBTNode
{
    public:
        int data, Lbit, Rbit;
        TBTNode *left, *right;
};


class TBTree
{
    public:
        TBTNode *curr, *temp, *head, *root;


        TBTree()
        {
            root = NULL;
        }


        void create();
        void inorder(TBTNode *);
};


void TBTree :: create()
{
    char s;
    int flag;
    TBTNode *node, *temp;


    head = new TBTNode;
```

```
head->left=head;
head->right=head;
head->Rbit=head -> Lbit =1;


root = new TBTNode;
cout<<endl<<"\n Enter data for root = ";
cin >> root -> data;
root->left=head;
root->right=head;


head->left = root;
root->Lbit=root->Rbit=1;


do
{
    node = new TBTNode;
    cout<<"Enter next data = ";
    cin>> node->data;
    node->Lbit=node->Rbit=1;
    temp = root;


    while(1)
    {
        if(node->data<temp->data)
        {
            if(temp->Lbit==1)
            {
                node->left=temp->left;
                node->right=temp;
                temp->Lbit=0;
                temp->left=node;
                break;
            }
            else
            temp=temp->left;
```

```
            }
            else{
                if(temp->Rbit==1)
                {
                    node->left=temp;
                    node->right=temp->right;
                    temp->right = node;
                    temp->Rbit = 0;
                    break;
                }
                else
                temp=temp->right;
            }
        }
        cout<<"\n\nDo you want to add more data ?[y/n]";
        cin>>s;
    }
    while(s=='y'|| s=='Y');
}


void TBTree::inorder(TBTNode *)
{
    TBTNode *temp;
    temp=root;
    int flag=0;
    if(root==NULL)
    {
        cout<<"Tree not present";
    }

    else
        {
            while(temp!=head)
            {
                if (temp->Lbit==0 && flag==0)
                {
                    temp=temp->left;
                }
```

```
                else
                {
                    cout<<temp->data<<" ";
                    if(temp->Rbit==0)
                    {
                        temp=temp->right;
                        flag=0;
                    }
                        else
                        {
                            temp=temp->right;
                            flag=1;
                        }
                }
            }
        }
}


int main()
{
    TBTree t;
    cout<<"\n\t**** Threaded Binary Tree ****";
    t.create();
    cout<<"\n\nInorder Traversal: ";
    t.inorder(t.root);
    return 0;
}
```

Output: