MIT World Peace University

Advanced Data Structures
Second Year B. Tech, Semester 4

Implementation of Threaded Binary Tree and Its Traversals

Assignment No. 4

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 26, 2023

Contents

1 Objectives

1. **To study the data Structure** : Threaded Binary Tree
2. To study the advantages of Threaded Binary Tree over Binary Tree

2 Problem Statement

Implement threaded binary tree and perform inorder traversal.

3 Theory

3.1 Threaded Binary Tree

3.1.1 What is Threaded Binary Tree?

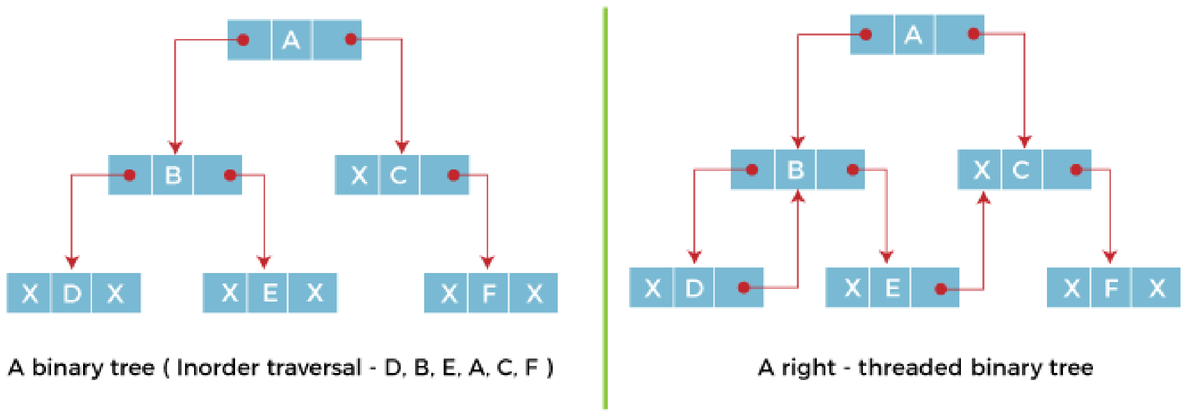
A
B
X C
X D X
X E X
X F X
A binary tree (Inorder traversal - D, B, E, A, C, F)
A
B
X C
X D
X E
X F X
A right - threaded binary tree

Figure 1: Threaded and a Normal Binary Tree

A Threaded Binary Tree is a binary tree where every node has an extra bit, and this bit is used to indicate whether the right pointer points to the right child or to the inorder successor of the node. In this way, we can traverse the tree without using the stack and without recursion. A Threaded Binary Tree is a binary tree where every node has an extra bit, and this bit is used to indicate whether the right pointer points to the right child or to the inorder successor of the node. In this way, we can traverse the tree without using the stack and without recursion.

3.1.2 Advantages of Threaded Binary Tree

1. Threaded Binary Tree is used to traverse the tree without using the stack and without recursion.
2. Inorder traversal of a binary tree is the most frequently used traversal. Inorder traversal of a threaded binary tree is done without using the stack and without recursion easier than the normal binary tree.

3.2 Space Utilization in Threaded Binary Tree

Space Utilized by a Normal Tree A Normal Binary Tree with N nodes each has data, left and right pointers. So, the space utilized by a normal binary tree is 3N.

Space Utilized by a Threaded Tree

A Threaded Binary Tree with N nodes each has data, left and right pointers and 2 bits. So, the space utilized by a threaded binary tree is 3N+2N = 5N.

But they avoid the usege of a stack during traversal. So, the space utilized by a threaded binary tree is less than the space utilized by a normal binary tree.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : g++ and gcc on linux for C++

5 Input

1. Input at least 10 nodes.
2. Display inorder traversal of binary tree with 10 nodes.

6 Output

1. The traversal of the Threaded binary tree in different ways.

7 Test Conditions

1. Input at least 10 nodes.
2. Display all traversals of binary tree with 10 nodes.(recursive and nonrecursive)

8 Pseudo Code

8.1 Create

```
// Allocate memory for root
root = new ThreadedBinaryTreeNode()

// Assign root->leftc and rightc to head
root->left = head
root->right = head

// Accept root data
print "Enter the root data: "
cin >> root->data

```

```
// Assign head lbit to 0
head->left = root
// Assign head->leftc to root
head->isLeftNodeAThread = false

ThreadedBinaryTreeNode *temp, *curr
temp = root
bool flag = true
int again = 0
int choice = 0

print "Do you want to enter another node? (1 or 0)" << endl
cin >> again
while (again != 0)
    temp = root
    flag = true
    print "Enter 1 for Entering a new node to the Left, and 2 for Entering it to
    the right" << endl
    cin >> choice
    while (flag)
        if (choice == 1)
            if (temp->isLeftNodeAThread == true)
                curr = new ThreadedBinaryTreeNode()
                print "Enter the Data: "
                cin >> curr->data
                curr->right = temp
                temp->left = curr
                temp->isLeftNodeAThread = false
                flag = false
            else
                temp = temp->left

        else if (choice == 2)
            if (temp->isRightNodeAThread == true)
                curr = new ThreadedBinaryTreeNode()
                print "Enter the Data: "
                cin >> curr->data
                curr->left = temp
                curr->right = temp->right
                temp->right = curr
                temp->isRightNodeAThread = false
                flag = false
            else
                temp = temp->right

    print "Do you want to enter another node? (1 or 0)" << endl
    cin >> again
```

8.2 Inorder Traversal

```
inorder_traversal()
    ThreadedBinaryTreeNode *temp
    temp = head
    while (true)
        temp = inorder_successor(temp)
        if (temp == head)
            break
        print temp->data << " "
```

```

ThreadedBinaryTreeNode *inorder_successor(ThreadedBinaryTreeNode *temp)
    ThreadedBinaryTreeNode *x = temp->right
    if (!temp->isRightNodeAThread)
        while (x->isLeftNodeAThread == false)
            x = x->left
    return x
```

8.3 PreOrder Traversal

```
void preorder_traversal()
    ThreadedBinaryTreeNode *temp = head->left
    while (temp != head)
        print temp->data << " "
        while (temp->isLeftNodeAThread == false)
            temp = temp->left
            print temp->data << " "
        while (temp->isRightNodeAThread == true)
            temp = temp->right
        temp = temp->right
```

9 Time Complexity

9.1 Creation of Threaded Binary Tree

- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

9.2 Inorder Traversal

- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

9.3 Preorder Traversal

- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

10 Code

10.1 Program

```
#include <iostream>
#include <queue>
#include <stack>
#include <string.h>
using namespace std;

class ThreadedBinaryTreeNode
{
    string data;
    ThreadedBinaryTreeNode *left;
    ThreadedBinaryTreeNode *right;
    bool isLeftNodeAThread;
    bool isRightNodeAThread;

public:
    ThreadedBinaryTreeNode()
    {
        left = NULL;
        right = NULL;
        isLeftNodeAThread = true;
        isRightNodeAThread = true;
    }
    friend class ThreadedBinaryTree;
};

class ThreadedBinaryTree
{
public:
    ThreadedBinaryTreeNode *head;
    ThreadedBinaryTreeNode *root;
    ThreadedBinaryTree()
    {
        head = new ThreadedBinaryTreeNode();
        head->left = head;
        head->right = head;
        head->isLeftNodeAThread = false;
        head->isRightNodeAThread = false;
        root = NULL;
    }

    void create()
    {
        // Allocate memory for root;
        root = new ThreadedBinaryTreeNode();

        // Assign root->leftc and rightc to head;
        root->left = head;
        root->right = head;

        // Accept root data;
        cout << "Enter the root data: ";
        cin >> root->data;

        // Assign head lbit to 0;
        head->left = root;
        // Assign head->leftc to root;
        head->isLeftNodeAThread = false;

        ThreadedBinaryTreeNode *temp, *curr;
        temp = root;
        bool flag = true;
        int again = 0;
        int choice = 0;

        cout << "Do you want to enter another node? (1 or 0)" << endl;
        cin >> again;
        while (again != 0)
        {
            temp = root;
            flag = true;

            cout << "Enter 1 for Entering a new node to the Left, and 2 for
    Entering it to the right" << endl;
            cin >> choice;
            while (flag)
            {
                if (choice == 1)
                {
                    if (temp->isLeftNodeAThread == true)
                    {
                        curr = new ThreadedBinaryTreeNode();
                        cout << "Enter the Data: ";
                        cin >> curr->data;
                        curr->right = temp;
                        temp->left = curr;
                        temp->isLeftNodeAThread = false;
                        flag = false;
                    }
                    else
                    {
                        temp = temp->left;
                    }
                }
                else if (choice == 2)
                {
                    if (temp->isRightNodeAThread == true)
                    {
                        curr = new ThreadedBinaryTreeNode();
                        cout << "Enter the Data: ";
                        cin >> curr->data;
                        curr->left = temp;
                        curr->right = temp->right;
                        temp->right = curr;
                        temp->isRightNodeAThread = false;
                        flag = false;
                    }
                    else
                    {
                        temp = temp->right;
                    }
                }
            }
            cout << "Do you want to enter another node? (1 or 0)" << endl;
            cin >> again;
        }
    }
    void inorder_traversal()
    {
        ThreadedBinaryTreeNode *temp;
        temp = head;
        while (true)
        {
            temp = inorder_successor(temp);
            if (temp == head)
                break;
            cout << temp->data << " ";
        }
    }
    ThreadedBinaryTreeNode *inorder_successor(ThreadedBinaryTreeNode *temp)
    {
        ThreadedBinaryTreeNode *x = temp->right;
        if (!temp->isRightNodeAThread)
        {
            while (x->isLeftNodeAThread == false)
                x = x->left;
        }
        return x;
    }

    void preorder_traversal()
    {
        ThreadedBinaryTreeNode *temp = head->left;
        while (temp != head)
        {
            cout << temp->data << " ";
            while (temp->isLeftNodeAThread == false)
            {
                temp = temp->left;
                cout << temp->data << " ";
            }
            while (temp->isRightNodeAThread == true)
            {
                temp = temp->right;
            }
            temp = temp->right;
        }
    }
};

int main()
{
    ThreadedBinaryTree tree;
    tree.create();
    cout<<"The inorder traversal is:" << endl;
    tree.inorder_traversal();
    cout << endl;
    cout << "The preorder traversal is:" << endl;
    tree.preorder_traversal();
}
```

10.2 Input and Output

```
Enter the root data: 1
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 3
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 5
Do you want to enter another node? (1 or 0)
1
```

```
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
2
Enter the Data: 69
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 544
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 245
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
2
Enter the Data: 36
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 41
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 255
Do you want to enter another node? (1 or 0)

1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
2
Enter the Data: 64
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 21
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
1
Enter the Data: 47
Do you want to enter another node? (1 or 0)
1
Enter 1 for Entering a new node to the Left, and 2 for Entering it to the right
2
Enter the Data: 65
Do you want to enter another node? (1 or 0)
0
The inorder traversal is:
47 21 255 41 245 544 5 3 1 69 36 64 65
The preorder traversal is:
1 3 5 544 245 41 255 21 47 69 36 64 65
```

11 Conclusion

Thus, implemented threaded binary tree with inorder traversal.

12 FAQ

1. **Why TBT can be traversed without stack?**

   A threaded binary tree is a binary tree that has additional links between nodes to allow for traversal without the use of a stack or recursion. These extra links, called threads, connect nodes that would otherwise be null pointers in a traditional binary tree.

   There **are two types of threaded binary trees**: inorder threaded binary trees and preorder threaded binary trees.

   (a) In an inorder threaded binary tree, the threads are used to connect nodes to their inorder predecessor and inorder successor. These threads allow for a traversal of the tree in inorder without using a stack or recursion. To traverse the tree in inorder without a stack, you start at the root node and follow the leftmost thread until you reach a node without a left child. Then, you visit that node and follow the thread to its inorder successor. You repeat this process until you have visited all nodes in the tree.

   (b) In a preorder threaded binary tree, the threads are used to connect nodes to their preorder successor. These threads allow for a traversal of the tree in preorder without using a stack or recursion. To traverse the tree in preorder without a stack, you start at the root node and visit each node in the tree in preorder. When you visit a node, you follow the thread to its preorder successor and visit that node next.

   (c) In both cases, the threads provide a way to traverse the tree without using a stack or recursion, which can be useful in situations where memory usage is a concern or where recursion or stack-based algorithms are not allowed. However, constructing the threads requires additional memory and processing time, so the benefits of threaded binary trees depend on the specific use case.

2. **What are the advantages and disadvantages of TBT?**

   Threaded binary trees have several advantages and disadvantages that should be considered when deciding whether to use them in a specific scenario. Here are some of the main advantages and disadvantages:

   Advantages:

   (a) **Traversal without stack or recursion**: As mentioned before, threaded binary trees allow for traversal without using a stack or recursion. This can be useful in situations where memory usage is a concern or where recursion or stack-based algorithms are not allowed.

   (b) **Efficient Inorder and Preorder Traversal**: The use of threads can significantly improve the performance of inorder and preorder traversal. In fact, threaded binary trees can perform these traversals in O(n) time complexity, which is faster than the O(nlogn) complexity of the recursive approach.

   (c) **Space Efficiency**: Compared to traditional binary trees, threaded binary trees require less space to store the threads. This can be useful in situations where memory usage is a concern.

Disadvantages:

(a) **Construction Overhead**: The process of constructing threads can be time-consuming and requires additional memory. In some cases, the overhead of constructing the threads can negate any performance benefits.

(b) **Complexity**: Threaded binary trees can be more complex than traditional binary trees due to the additional links. This can make them harder to understand and maintain.

(c) **Limited Applications**: Threaded binary trees are not suitable for all types of problems. They are typically used in scenarios where traversal performance is critical, but in other cases, traditional binary trees or other data structures may be more appropriate.

3. **Write application of TBT**

(a) **Expression Trees**: Threaded binary trees can be used to represent arithmetic expressions in a compact form. By using threads, the expression tree can be traversed efficiently without using a stack or recursion. This can be useful in compilers and other applications that deal with arithmetic expressions.

(b) **Database Indexing**: Threaded binary trees can be used to implement indexing in databases. By using threads, the binary tree can be traversed efficiently, which can improve the performance of database queries.

(c) **Text Editors**: Threaded binary trees can be used in text editors to efficiently search for words in a document. By using threads, the binary tree can be traversed efficiently, which can improve the speed of searches.

(d) **Spell Checking**: Threaded binary trees can be used in spell checking applications to efficiently search for misspelled words. By using threads, the binary tree can be traversed efficiently, which can improve the speed of spell checking.

(e) **Image Processing**: Threaded binary trees can be used in image processing applications to efficiently process pixels in an image. By using threads, the binary tree can be traversed efficiently, which can improve the speed of image processing algorithms.