

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 15, 2023

Contents

1 Objectives	1
2 Problem Statement	1
3 Theory	1
3.1 Threaded Binary Tree	1
3.1.1 What is Threaded Binary Tree?	1
3.1.2 Advantages of Threaded Binary Tree	1
3.2 Space Utilization in Threaded Binary Tree	2
4 Platform	2
5 Input	2
6 Output	2
7 Test Conditions	2
8 Pseudo Code	2
8.1 Create	2
8.2 Inorder Traversal	2
8.3 PreOrder Traversal	2
9 Time Complexity	3
9.1 Creation of Threaded Binary Tree	3
9.2 Inorder Traversal	3
9.3 Preorder Traversal	3
10 Code	3
10.1 Program	3
10.2 Input and Output	6
11 Conclusion	9
12 FAQ	10

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?

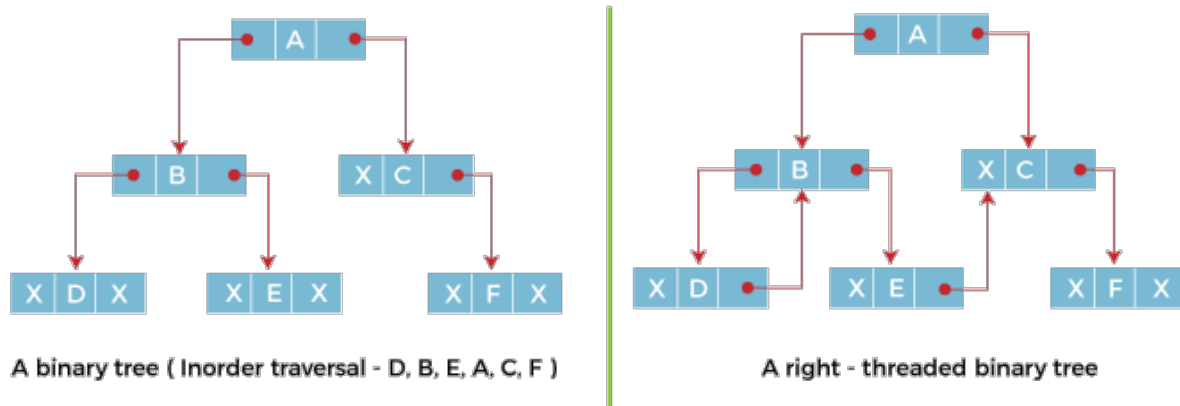


Figure 1: Threaded and a Normal Binary Tree

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 use 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

```
1 //Pseudo Code for Creation of Threaded Binary Tree
```

8.2 Inorder Traversal

8.3 PreOrder Traversal

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

```
1 #include <iostream>
2 #include <queue>
3 #include <stack>
4 #include <string.h>
5 using namespace std;
6
7 class ThreadedBinaryTreeNode
8 {
9     string data;
10    ThreadedBinaryTreeNode *left;
11    ThreadedBinaryTreeNode *right;
12    bool isLeftNodeAThread;
13    bool isRightNodeAThread;
14
15 public:
16    ThreadedBinaryTreeNode()
17    {
18        left = NULL;
19        right = NULL;
20        isLeftNodeAThread = true;
21        isRightNodeAThread = true;
22    }
23    friend class ThreadedBinaryTree;
24 };
25
26 class ThreadedBinaryTree
27 {
28 public:
29    ThreadedBinaryTreeNode *head;
30    ThreadedBinaryTreeNode *root;
31    ThreadedBinaryTree()
32    {
```

```
33     head = new ThreadedBinaryTreeNode();
34     head->left = head;
35     head->right = head;
36     head->isLeftNodeAThread = true;
37     head->isRightNodeAThread = true;
38     root = NULL;
39 }
40
41 void create()
42 {
43     // Allocate memory for root;
44     root = new ThreadedBinaryTreeNode();
45
46     // Assign root->leftc and rightc to head;
47     root->left = head;
48     root->right = head;
49
50     // Accept root data;
51     cout << "Enter the root data: ";
52     cin >> root->data;
53
54     // Assign head lbit to 0;
55     head->left = root;
56     // Assign head->leftc to root;
57     head->isLeftNodeAThread = false;
58
59     ThreadedBinaryTreeNode *temp, *curr;
60     temp = root;
61     bool flag = true;
62     int again = 0;
63     int choice = 0;
64
65     cout << "Do you want to enter another node? (1 or 0)" << endl;
66     cin >> again;
67     while (again != 0)
68     {
69         temp = root;
70         flag = true;
71
72         cout << "Enter 1 for Entering a new node to the Left, and 2 for
73 Entering it to the right" << endl;
74         cin >> choice;
75         while (flag)
76         {
77             if (choice == 1)
78             {
79                 if (temp->isLeftNodeAThread == true)
80                 {
81                     curr = new ThreadedBinaryTreeNode();
82                     cout << "Enter the Data: ";
83                     cin >> curr->data;
84                     curr->right = temp;
85                     temp->left = curr;
86                     temp->isLeftNodeAThread = false;
87                     flag = false;
88                 }
89                 else
90                 {
91                     temp = temp->left;
```

```
91         }
92     }
93     else if (choice == 2)
94     {
95         if (temp->isRightNodeAThread == true)
96         {
97             curr = new ThreadedBinaryTreeNode();
98             cout << "Enter the Data: ";
99             cin >> curr->data;
100            curr->left = temp;
101            curr->right = temp->right;
102            temp->right = curr;
103            temp->isRightNodeAThread = false;
104            flag = false;
105        }
106        else
107        {
108            temp = temp->right;
109        }
110    }
111    }
112    cout << "Do you want to enter another node? (1 or 0)" << endl;
113    cin >> again;
114 }
115 }
116 void inorder_traversal()
117 {
118     ThreadedBinaryTreeNode *temp;
119     temp = head;
120     while (true)
121     {
122         temp = inorder_successor(temp);
123         if (temp == head)
124             break;
125         cout << temp->data;
126     }
127 }
128 ThreadedBinaryTreeNode *inorder_successor(ThreadedBinaryTreeNode *temp)
129 {
130     ThreadedBinaryTreeNode *x = temp->right;
131     if (temp->isRightNodeAThread)
132     {
133         while (!x->isLeftNodeAThread)
134             x = x->left;
135     }
136     return x;
137 }
138
139 void preorder_traversal()
140 {
141     ThreadedBinaryTreeNode *temp = head;
142     while (temp != head)
143     {
144         cout << temp->data;
145         while (temp->isLeftNodeAThread)
146         {
147             temp = temp->left;
148             cout << temp->data;
149         }
```

```
150         while (temp->isRightNodeAThread)
151         {
152             temp = temp->right;
153         }
154         temp = temp->right;
155     }
156 }
157 };
158
159 int main()
160 {
161     ThreadedBinaryTree tree;
162     tree.create();
163     tree.inorder_traversal();
164     tree.preorder_traversal();
165     cout << "done";
166 }
```

10.2 Input and Output

```
1 What would like to do?
2
3
4 Welcome to ADS Assignment 2 - Binary Tree Traversals
5
6 What would you like to do?
7 1. Create a Binary Search Tree
8 2. Traverse the Tree Inorder Iteratively
9 3. Traverse the Tree PreOrder Iteratively
10 4. Traverse the Tree PostOrder Iteratively
11 5. Traverse it using BFS
12 6. Create a Copy of the tree Recursively
13 7. Create a Mirror of the Tree Recursively
14 8. Exit
15
16 1
17 Enter the Word: apple
18 Enter the definition of the word: fruit
19 Do you want to enter more Nodes? (0/1)
20
21 1
22 Enter the Word: banana
23 Enter the definition of the word: fruit
24 Do you want to enter another word? (1/0): 1
25 Enter the Word: keyboard
26 Enter the definition of the word: input
27 Do you want to enter another word? (1/0): 1
28 Enter the Word: pears
29 Enter the definition of the word: fruit
30 Do you want to enter another word? (1/0): 1
31 Enter the Word: bottle
32 Enter the definition of the word: water
33 Do you want to enter another word? (1/0): 1
34 Enter the Word: charger
35 Enter the definition of the word: charging
36 Do you want to enter another word? (1/0): 1
37 Enter the Word: monitor
38 Enter the definition of the word: see
39 Do you want to enter another word? (1/0): 1
```


Advanced Data Structures - Assignment 4

```
40 Enter the Word: paper
41 Enter the definition of the word: 1
42 Do you want to enter another word? (1/0): 1
43 Enter the Word: pen
44 Enter the definition of the word: writing
45 Do you want to enter another word? (1/0): 1
46 Enter the Word: phone
47 Enter the definition of the word: scrolling
48 Do you want to enter another word? (1/0): 0
49
50 What would like to do?
51
52
53 Welcome to ADS Assignment 2 - Binary Tree Traversals
54
55 What would you like to do?
56 1. Create a Binary Search Tree
57 2. Traverse the Tree Inorder Iteratively
58 3. Traverse the Tree PreOrder Iteratively
59 4. Traverse the Tree PostOrder Iteratively
60 5. Traverse it using BFS
61 6. Create a Copy of the tree Recursively
62 7. Create a Mirror of the Tree Recursively
63 8. Exit
64
65 2
66 Traversing through the Binary Tree Inorder Iteratively:
67 apple : fruit
68 banana : fruit
69 bottle : water
70 charger : charging
71 keyboard : input
72 monitor : see
73 paper : 1
74 pears : fruit
75 pen : writing
76 phone : scrolling
77
78 What would like to do?
79
80
81 Welcome to ADS Assignment 2 - Binary Tree Traversals
82
83 What would you like to do?
84 1. Create a Binary Search Tree
85 2. Traverse the Tree Inorder Iteratively
86 3. Traverse the Tree PreOrder Iteratively
87 4. Traverse the Tree PostOrder Iteratively
88 5. Traverse it using BFS
89 6. Create a Copy of the tree Recursively
90 7. Create a Mirror of the Tree Recursively
91 8. Exit
92
93 5
94 Traversing through the Binary Tree using BFS:
95 apple : fruit
96 banana : fruit
97 keyboard : input
98 bottle : water
```

Advanced Data Structures - Assignment 4

```
99 pears : fruit
100 charger : charging
101 monitor : see
102 pen : writing
103 paper : 1
104 phone : scrolling
105
106 What would like to do?
107
108
109 Welcome to ADS Assignment 2 - Binary Tree Traversals
110
111 What would you like to do?
112 1. Create a Binary Search Tree
113 2. Traverse the Tree Inorder Iteratively
114 3. Traverse the Tree PreOrder Iteratively
115 4. Traverse the Tree PostOrder Iteratively
116 5. Traverse it using BFS
117 6. Create a Copy of the tree Recursively
118 7. Create a Mirror of the Tree Recursively
119 8. Exit
120
121 6
122 Creating a copy of the tree
123 Traversing through the Binary Tree Inorder Iteratively:
124 apple : fruit
125 banana : fruit
126 bottle : water
127 charger : charging
128 keyboard : input
129 monitor : see
130 paper : 1
131 pears : fruit
132 pen : writing
133 phone : scrolling
134 Traversing via Breadth First Search:
135 apple : fruit
136 banana : fruit
137 keyboard : input
138 bottle : water
139 pears : fruit
140 charger : charging
141 monitor : see
142 pen : writing
143 paper : 1
144 phone : scrolling
145
146 What would like to do?
147
148
149 Welcome to ADS Assignment 2 - Binary Tree Traversals
150
151 What would you like to do?
152 1. Create a Binary Search Tree
153 2. Traverse the Tree Inorder Iteratively
154 3. Traverse the Tree PreOrder Iteratively
155 4. Traverse the Tree PostOrder Iteratively
156 5. Traverse it using BFS
157 6. Create a Copy of the tree Recursively
```

```
158 7. Create a Mirror of the Tree Recursively
159 8. Exit
160
161 7
162 Creating a mirror of the tree
163 Traversing through the Binary Tree Inorder Iteratively:
164 phone : scrolling
165 pen : writing
166 pears : fruit
167 paper : 1
168 monitor : see
169 keyboard : input
170 charger : charging
171 bottle : water
172 banana : fruit
173 apple : fruit
174 Traversing via Breadth First Search:
175 apple : fruit
176 banana : fruit
177 keyboard : input
178 pears : fruit
179 bottle : water
180 pen : writing
181 monitor : see
182 charger : charging
183 phone : scrolling
184 paper : 1
185
186 What would like to do?
187
188
189 Welcome to ADS Assignment 2 - Binary Tree Traversals
190
191 What would you like to do?
192 1. Create a Binary Search Tree
193 2. Traverse the Tree Inorder Iteratively
194 3. Traverse the Tree PreOrder Iteratively
195 4. Traverse the Tree PostOrder Iteratively
196 5. Traverse it using BFS
197 6. Create a Copy of the tree Recursively
198 7. Create a Mirror of the Tree Recursively
199 8. Exit
200
201 8
202 Exiting the program
```

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 $O(n \log n)$ 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.