MIT World Peace University
Advanced Data Structures

Assignment 4

Naman Soni Roll No. 10

Contents

1 Problem Statement

Implement threaded binary tree and perform inorder traversal.

2 Objective

- To study the data Structure : Threaded Binary Tree
- To study the advantages of Threaded Binary Tree over Binary Tree

3 Theory

3.1 *The data structure : Threaded Binary Tree*

A Threaded Binary Tree is a modified version of a binary tree where each node has a reference or pointer to its in-order predecessor or successor, called threaded links. These threaded links make traversal of the tree more efficient, as they eliminate the need for recursive function calls and stack space for storing nodes.

In a threaded binary tree, the null pointers of leaf nodes are replaced by pointers to their in-order predecessor and successor. For a node that has a left child, its predecessor link points to the maximum node in its left subtree. For a node that has a right child, its successor link points to the minimum node in its right subtree.

There are two types of threaded binary trees: singly threaded and doubly threaded. In a singly threaded tree, each node has either a predecessor or a successor thread, while in a doubly threaded tree, each node has both predecessor and successor threads.

Threaded binary trees are useful in applications where traversal of the tree is a common operation, such as in database indexing and searching, and they can save both memory and computational resources. However, they are more complicated to implement than standard binary trees, and the threaded links must be maintained when nodes are inserted or removed from the tree.

3.2 *Space Utilization in Threaded Binary Tree*

Space utilization in a threaded binary tree is a measure of how efficiently the memory is used to store the tree structure. Compared to a regular binary tree, a threaded binary tree can potentially save space by using the null pointers of leaf nodes to store threaded links to their in-order predecessors or successors.

In a singly threaded binary tree, each leaf node has one threaded link, which replaces one null pointer. In a doubly threaded binary tree, each leaf node has two threaded links, which replace two null pointers. This means that a threaded binary tree can potentially use up to 50 % less memory than a regular binary tree.

However, the actual space savings depend on the size and shape of the tree, as well as the implementation of the threaded links. In some cases, the overhead of maintaining the threaded links may outweigh the space savings.

Overall, space utilization is an important consideration when designing and implementing a threaded binary tree, and it is important to balance the benefits of space savings with the cost of additional complexity in the implementation.

4 Implementation

4.1 *Platform*

- 64-bit Mac OS
- Open Source C++ Programming tool like Visual Studio Code

4.2 *Test Conditions*

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

5 Conclusion

Thus, implemented threaded binary tree with inorder traversal.

6 FAQ's

6.1 *Why TBT can be traversed without stack?*

A Threaded Binary Tree (TBT) can be traversed without using a stack because of the presence of the threaded links between nodes. These links provide a way to navigate the tree without having to use a recursive function call or a stack to keep track of the nodes.

For example, in-order traversal of a TBT involves visiting the left subtree, the current node, and the right subtree in that order. To traverse a TBT in-order without a stack, we start at the root node and follow the left pointers until we reach a node with a null left pointer, which means we have reached the leftmost node. At this point, we visit the node, and then follow its successor link to the next node in the in-order sequence. We repeat this process until we reach the rightmost node, which has a null successor link.

By using the threaded links instead of a stack, we can traverse the tree without incurring the overhead of a recursive function call or the additional memory required to store nodes on the stack. This can make the traversal more efficient, especially for large or deep trees, where the stack space required for recursive traversal can be significant.

6.2 *What are the advantages and disadvantages of TBT?*

Advantages of Threaded Binary Trees (TBT's):

1. Improved traversal efficiency: TBTs allow for in-order traversal without using a stack, which can improve traversal efficiency by reducing memory usage and eliminating the need for recursive function calls.
2. Space optimization: TBTs can potentially use less memory than regular binary trees by using null pointers in leaf nodes to store threaded links to their in-order predecessors or successors.
3. Fast searching: TBTs can be used for searching and sorting, and their threaded links can make these operations faster by reducing the number of comparisons needed to find a specific node.

Disadvantages of Threaded Binary Trees (TBT's)

1. More complex implementation: Implementing TBTs requires additional logic to maintain the threaded links when nodes are inserted or removed from the tree. This can make the implementation more complex and error-prone than regular binary trees.

2. Increased overhead: Maintaining the threaded links can add overhead to the insertion and removal operations, which may reduce the overall performance of the tree.

3. Limited flexibility: TBTs are specifically designed for in-order traversal and may not be well-suited for other types of tree operations or data structures.

6.3 *Write application of TBT*

Threaded Binary Trees (TBTs) can be used in a variety of applications, including:

1. In-order traversal: TBTs were originally designed for efficient in-order traversal of binary trees. TBTs can be used to traverse the tree without using a stack, which can be useful for large or deep trees where stack space is a concern.

2. Searching and sorting: TBTs can be used for searching and sorting operations, where their threaded links can improve performance by reducing the number of comparisons needed to find a specific node.

3. Memory optimization: TBTs can potentially use less memory than regular binary trees by using null pointers in leaf nodes to store threaded links to their in-order predecessors or successors. This can be useful in memory-constrained environments, such as embedded systems or mobile devices.

4. Data compression: TBTs can be used in data compression algorithms, such as Huffman coding, where they can be used to efficiently represent the frequency of characters in a text string.

5. Expression evaluation: TBTs can be used in expression evaluation, where they can be used to represent and evaluate arithmetic expressions, such as postfix notation or expression trees.