Converting a binary tree to binary search tree by maintaining its original structure.

Akshay Gupta IIT2017505, Naman Deept IIT2017507, Snigdha Dobhal IIT2017506

Dept. of Information Technology
Indian Institute of Information Technology Allahabad

February 22, 2019

1 Abstract

This paper introduces an efficient algorithm for conversion of any given number of nodes in any type of binary tree into its binary search tree by maintaining its original structure.

Keywords: *Binary Tree*, *Binary Search Tree*, *Original Structure*, *Conversion*, *Recursion*

2 Introduction

A **Binary Tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. A recursive definition using just set theory notions is that a (non-empty) binary tree is a three tuple (L, S, R), where L and R are binary trees or the empty set and S is a singleton set.
Binary Search Trees (BST), sometimes called ordered or sorted binary trees, are a particular type of container: data structures that store "items" (such as numbers, names etc.) in memory. They allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its key.Binary search trees on average provide much better time complexity than the linear time required by the unsorted array but slower than the hash tables.

3 Proposed Method

Input : Given a binary tree with n number of nodes.

3.1 Procedure

1. At first a binary tree with n number of nodes was created of desired structure.

2. Inorder traversal of the created binary tree was performed and the resulted elements were stored in an array.

3. Array was first checked if it was in sorted otherwise array elements were sorted.

4. Once again inorder traversal was performed for the given binary tree and the corresponding nodes were updated with the array elements.

5. Updated binary tree is the desired binary search tree with original structure still intact .

3.2 Algorithm

3.2.1 Node creation:

Creating a node (initializing the values of its data and left and right links).

```
1: procedure NodeInitials(data)
2:     info ← data
3:     node left ← null
4:     node right ← null
5: end procedure
```

3.2.2 Inorder Traversal:

Inorder traversal of the elements in the binary tree and appending these directly onto a set.

```
1: procedure inorder(node root,Set set)
2:     if root = null
3:         EXIT
4:     end if
5:     inorder(root.left, Set)
6:     Set.add(root.data)
7:     inorder(root.right, Set)
8: end procedure
```

3.2.3 Binary tree to BST conversion:

The below algorithm converts binary tree to BST using iterator.

```
1: procedure convertBST(NodeRoot, Iterator < Integer > it)
2:     if root = null
3:         EXIT
4:     end if
5:     convertBST(root.left, it);
6:     root.data ← it.next()
7:     convertBST(root.right, it);
8: end procedure
```

3.2.4 Inserting elements:

The below algorithm inserts elements into the fully balanced binary tree which is then converted into Binary Search Tree without distorting its structure.

```
1: procedure insertion(Nodehead, key)
2:     Queue<Node>queue ← null
3:     queue asemptyLinkedList
4:     queue ← queue + [head]
5:     while queue = NULL
6:         head ← queue.peek()
7:         queue ← queue.remove()
8:         if head.left = null
9:             head.left ← key
10:            break
11:        end if
12:        if head.left ≠ null
13:            queue ← queue + [head.left]
14:        end if
15:        if head.right = null
16:            head.right ← key
17:            break
18:        end if
19:        if head.right ≠ null
20:            queue ← queue + [head.right]
21:        end if
22:    end while
23: end procedure
```

3.3 Time Complexity Analysis

Best Case-Given algorithms works at its best in the case when all the elements are already in sorted order (either ascending or descending) So that we only need the case of insertion of the elements into a fully balanced binary tree which takes time which is proportional to the size of the array,that is

$$t_\omega \propto (n)$$

Hence the best case complexity turns out to be $\Omega(n)$.
Worst Case- Given algorithm will show its worst behaviour when the array is non sorted, Hence time taken will be mainly due to 2 considerations ,Firstly the time taken for inserting elements into the full binary tree and secondly to sort the elements after insertion .So the time taken in this case will be:

$$t_o \propto nlog_2(n)$$

Where t_o is the worst case time ,Insertion time is proportional to n and sorting time for every element is proportional to $log_2(n)$.Hence the overall complexity in this case will turn out to be $O(nlog(n))$.

4 Experimental results

If array after storing elements is already sorted than the given algorithm will convert any binary tree into its binary search tree in linear time complexity as for such trees no sorting is needed .

If array after storing elements of binary tree is unsorted than the array elements must be sorted and inorder traversal of the binary search tree should be done to update the value of each node to generate the binary search tree by maintaining the original structure. Since in such case , the given procedure depends upon the method of sorting hence the time complexity depends upon the sorting mechanism being used which is evident from the graph.

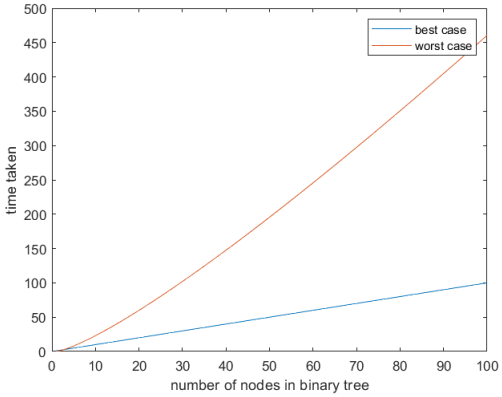

Figure 1: Figure: Graph showing the experimental analysis for the best case vs the worst case

5 Conclusion

Insertion , traversal and updating data of any particular node in binary tree takes time complexity proportional to that of O(n) . Since , converting a binary tree into binary search tree is done by updating all the node of the original binary tree to a sorted array found by inorder traversal of the original binary tree which takes time complexity proportional to that of O(n$log_2 n$) which is the bottleneck for the above algorithm hence the overall time complexity came out to be O(n$log_2 n$).

6 References

1. IDAA432C (Design and Analysis of Algorithm) class lecture
2. Binary Tree *https://en.wikipedia.org/wiki/Binary$_t$ree*
3. Binary Search Tree *https://en.wikipedia.org/wiki/Binary$_s$earch$_t$ree*
4. Introduction to Algorithms by Cormen