GeeksforGeeks

Related Articles

Binary Tree to Binary Search Tree Conversion

Difficulty Level : Medium • Last Updated : 24 May, 2021

Given a Binary Tree, convert it to a Binary Search Tree. The conversion must be done in such a way that keeps the original structure of Binary Tree.

Examples

```
Example 1
Input:
          10
         /  \
        2    7
       / \
      8   4
Output:
          8
         /  \
        4    10
       / \
      2   7


Example 2
Input:
          10
         /  \
        30   15
       /      \
      20       5
Output:
```

```
          15
        /   \
      10     20
     /       \
    5         30
```

Solution

Following is a 3 step solution for converting Binary tree to Binary Search Tree.

1) Create a temp array arr[] that stores inorder traversal of the tree. This step takes O(n) time.

2) Sort the temp array arr[]. Time complexity of this step depends upon the sorting algorithm. In the following implementation, Quick Sort is used which takes (n^2) time. This can be done in O(nLogn) time using Heap Sort or Merge Sort.

3) Again do inorder traversal of tree and copy array elements to tree nodes one by one. This step takes O(n) time.

Following is C implementation of the above approach. The main function to convert is highlighted in the following code.

C

```c
/* A program to convert Binary Tree to Binary Search Tree */
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node structure */
```

```
struct node {
    int data;
    struct node* left;
    struct node* right;
};

/* A helper function that stores inorder traversal of a tree rooted
with node */
void storeInorder(struct node* node, int inorder[], int* index_ptr)
{
    // Base Case
    if (node == NULL)
        return;

    /* first store the left subtree */
    storeInorder(node->left, inorder, index_ptr);

    /* Copy the root's data */
    inorder[*index_ptr] = node->data;
    (*index_ptr)++; // increase index for next entry

    /* finally store the right subtree */
    storeInorder(node->right, inorder, index_ptr);
}

/* A helper function to count nodes in a Binary Tree */
int countNodes(struct node* root)
{
    if (root == NULL)
        return 0;
    return countNodes(root->left) + countNodes(root->right) + 1;
}

// Following function is needed for library function qsort()
int compare(const void* a, const void* b)
{
    return (*(int*)a - *(int*)b);
}

/* A helper function that copies contents of arr[] to Binary Tree.
This function basically does Inorder traversal of Binary Tree and
one by one copy arr[] elements to Binary Tree nodes */
void arrayToBST(int* arr, struct node* root, int* index_ptr)
{
    // Base Case
    if (root == NULL)
        return;

    /* first update the left subtree */
    arrayToBST(arr, root->left, index_ptr);

    /* Now update root's data and increment index */
```

```
    root->data = arr[*index_ptr];
    (*index_ptr)++;

    /* finally update the right subtree */
    arrayToBST(arr, root->right, index_ptr);
}

// This function converts a given Binary Tree to BST
void binaryTreeToBST(struct node* root)
{
    // base case: tree is empty
    if (root == NULL)
        return;

    /* Count the number of nodes in Binary Tree so that
    we know the size of temporary array to be created */
    int n = countNodes(root);

    // Create a temp array arr[] and store inorder traversal of tree in arr[]
    int* arr = new int[n];
    int i = 0;
    storeInorder(root, arr, &i);

    // Sort the array using library function for quick sort
    qsort(arr, n, sizeof(arr[0]), compare);

    // Copy array elements back to Binary Tree
    i = 0;
    arrayToBST(arr, root, &i);

    // delete dynamically allocated memory to avoid memory leak
    delete[] arr;
}

/* Utility function to create a new Binary Tree node */
struct node* newNode(int data)
{
    struct node* temp = new struct node;
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

/* Utility function to print inorder traversal of Binary Tree */
void printInorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);
```

```
    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}

/* Driver function to test above functions */
int main()
{
    struct node* root = NULL;

    /* Constructing tree given in the above figure
        10
        / \
        30 15
    /     \
    20     5 */
    root = newNode(10);
    root->left = newNode(30);
    root->right = newNode(15);
    root->left->left = newNode(20);
    root->right->right = newNode(5);

    // convert Binary Tree to BST
    binaryTreeToBST(root);

    printf("Following is Inorder Traversal of the converted BST: \n");
    printInorder(root);

    return 0;
}
```

Java

```
/* A program to convert Binary Tree to Binary Search Tree */
import java.util.*;

public class GFG{

    /* A binary tree node structure */
    static class Node {
        int data;
        Node left;
        Node right;
    };

    // index pointer to pointer to the array index
    static int index;
```

```
/* A helper function that stores inorder traversal of a tree rooted
with node */
static void storeInorder(Node node, int inorder[])
{
    // Base Case
    if (node == null)
        return;

    /* first store the left subtree */
    storeInorder(node.left, inorder);

    /* Copy the root's data */
    inorder[index] = node.data;
    index++; // increase index for next entry

    /* finally store the right subtree */
    storeInorder(node.right, inorder);
}

/* A helper function to count nodes in a Binary Tree */
static int countNodes(Node root)
{
    if (root == null)
        return 0;
    return countNodes(root.left) + countNodes(root.right) + 1;
}

/* A helper function that copies contents of arr[] to Binary Tree.
This function basically does Inorder traversal of Binary Tree and
one by one copy arr[] elements to Binary Tree nodes */
static void arrayToBST(int[] arr, Node root)
{
    // Base Case
    if (root == null)
        return;

    /* first update the left subtree */
    arrayToBST(arr, root.left);

    /* Now update root's data and increment index */
    root.data = arr[index];
    index++;

    /* finally update the right subtree */
    arrayToBST(arr, root.right);
}

// This function converts a given Binary Tree to BST
static void binaryTreeToBST(Node root)
{
```

```
        // base case: tree is empty
        if (root == null)
            return;

        /* Count the number of nodes in Binary Tree so that
        we know the size of temporary array to be created */
        int n = countNodes(root);

        // Create a temp array arr[] and store inorder traversal of tree in arr[]
        int arr[] = new int[n];

        storeInorder(root, arr);

        // Sort the array using library function for quick sort
        Arrays.sort(arr);


        // Copy array elements back to Binary Tree
        index = 0;
        arrayToBST(arr, root);
    }

    /* Utility function to create a new Binary Tree node */
    static Node newNode(int data)
    {
        Node temp = new Node();
        temp.data = data;
        temp.left = null;
        temp.right = null;
        return temp;
    }

    /* Utility function to print inorder traversal of Binary Tree */
    static void printInorder(Node node)
    {
        if (node == null)
            return;

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.data + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    /* Driver function to test above functions */
    public static void main(String args[])
    {
        Node root = null;
```

```
        /* Constructing tree given in the above figure
             10
             / \
             30 15
        /     \
        20     5 */
        root = newNode(10);
        root.left = newNode(30);
        root.right = newNode(15);
        root.left.left = newNode(20);
        root.right.right = newNode(5);

        // convert Binary Tree to BST
        binaryTreeToBST(root);

        System.out.println("Following is Inorder Traversal of the converted BST: ");
        printInorder(root);

    }
}

// This code is contributed by adityapande88.
```

Python

```
# Program to convert binary tree to BST

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Helper function to store the inorder traversal of a tree
def storeInorder(root, inorder):

    # Base Case
    if root is None:
        return

    # First store the left subtree
    storeInorder(root.left, inorder)

    # Copy the root's data
    inorder.append(root.data)
```

```
        # Finally store the right subtree
        storeInorder(root.right, inorder)

    # A helper funtion to count nodes in a binary tree
    def countNodes(root):
        if root is None:
            return 0

        return countNodes(root.left) + countNodes(root.right) + 1

    # Helper function that copies contents of sorted array
    # to Binary tree
    def arrayToBST(arr, root):

        # Base Case
        if root is None:
            return

        # First update the left subtree
        arrayToBST(arr, root.left)

        # now update root's data delete the value from array
        root.data = arr[0]
        arr.pop(0)

        # Finally update the right subtree
        arrayToBST(arr, root.right)

    # This function converts a given binary tree to BST
    def binaryTreeToBST(root):

        # Base Case: Tree is empty
        if root is None:
            return

        # Count the number of nodes in Binary Tree so that
        # we know the size of temporary array to be created
        n = countNodes(root)

        # Create the temp array and store the inorder traversal
        # of tree
        arr = []
        storeInorder(root, arr)

        # Sort the array
        arr.sort()

        # copy array elements back to binary tree
        arrayToBST(arr, root)

    # Print the inorder traversal of the tree
```

```
def printInorder(root):
    if root is None:
        return
    printInorder(root.left)
    print root.data,
    printInorder(root.right)

# Driver program to test above function
root = Node(10)
root.left = Node(30)
root.right = Node(15)
root.left.left = Node(20)
root.right.right = Node(5)

# Convert binary tree to BST
binaryTreeToBST(root)

print "Following is the inorder traversal of the converted BST"
printInorder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output

```
Following is the inorder traversal of the converted BST
5 10 15 20 30
```

Complexity Analysis:

- **Time Complexity:** O(nlogn). This is the complexity of the sorting algorithm which we are using after first in-order traversal, rest of the operations take place in linear time.
- **Auxiliary Space:** O(n). Use of data structure 'array' to store in-order traversal.

We will be covering another method for this problem which converts the tree using O(height of the tree) extra space.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

In case you wish to attend live classes with industry experts, please refer **DSA Live Classes**

Previous Next

RECOMMENDED ARTICLES

Page : 1 2 3

30, Jan 14

19, Feb 19

04 **Minimum swap required to convert binary tree to binary search tree**
28, Jan 17

08 **Anagram Substring Search (Or Search for all permutations)**
19, Jul 14

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : bidibaaz123, Akanksha_Rai, simmytarika5, adityapande88

Article Tags : Amazon, Binary Search Tree, Tree

Practice Tags : Amazon, Binary Search Tree, Tree

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org