Admin AfterAcademy
13 Nov 2019

Check if a binary tree is BST or not

Check if a binary tree is BST or not- Interview Problem

AfterAcademy

Level: Easy

Asked in: Microsoft, Google, Amazon, Ebay

Understanding the Problem

Problem Description: Given the root of a binary tree, check whether it is a binary search tree or not.

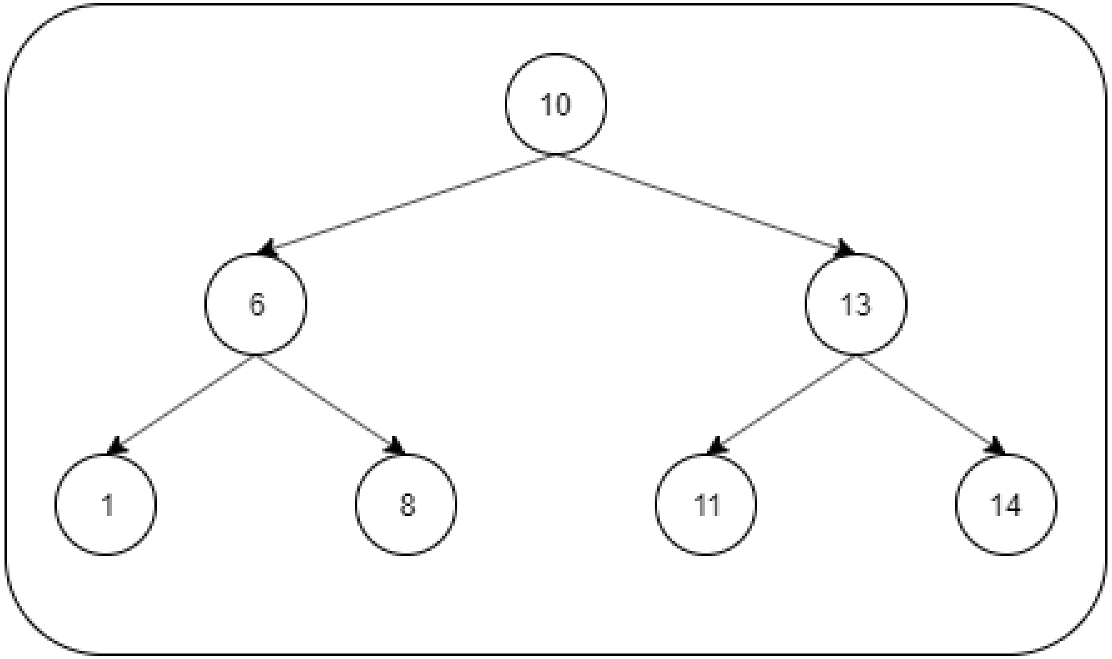
Binary Search Trees follow the following properties:-

- All nodes in the left subtree of a node have values less than the node's value
- All nodes in the right subtree of a node have values greater than the node's value
- Both left and right subtrees are also binary search trees

Problem Note: The inorder traversal of a binary search tree explores tree nodes in sorted order. (Getting an approach yet?)

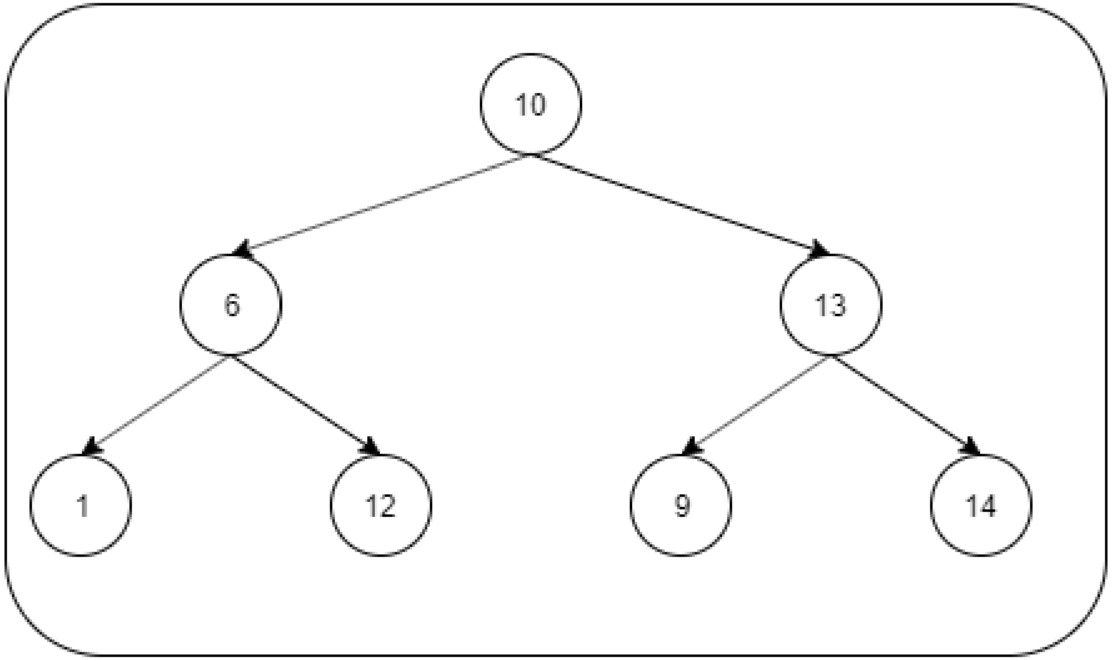
For example :

Input: Root 10 is given for this tree



Output: **True**

Input: Root 10 is given for this tree



Output: **False**, the right child of the left child of root is greater than the root

The node structure for the BST passed to your function will be

```
class BSTNode{
    int val
    BSTNode *left
    BSTNode *right
}
```

Common Mistake

A quite common naive solution for this problem involves just recursively checking if the left child's value is less than the current node's value and the right child's value is greater than the current node's value.

```
boolean isBST(BSTNode root)
{
    if (root == NULL)
        return True

    if (root.left and root.left.val > root.val)
        return False
    if (root.right and root.right.val < root.val)
        return False

    if (isBST(root.left) and isBST(root.right))
        return True
    return False
}
```

What's wrong with this particular implementation?

- This does not check if there is an element in the entire left subtree which is greater than the current node’s value.
- Or if there is an element in the right subtree which is smaller than the current node’s value.
- This implementation returns True for the second example above, which is obviously wrong.

Brute force and Efficient solutions

We will be discussing three possible solutions for this problem:-

1. **Brute Force Approach**: Get maximum value from left subtree and minimum value from the right subtree and check if root’s value falls within that range
2. **Optimized Brute Force**: Pass the allowed range for left and right subtrees as function arguments
3. **Using Inorder Traversal**: Store the inorder traversal of the tree and check if it's in sorted order

1. Brute Force Approach

We need to check is the left subtree of a node contains any element greater than the node’s value and whether the right subtree of a node contains any element smaller than the node’s value.

We shall design two helper functions **getMin(root)** and **getMax(root)** which will get the minimum and maximum values of the tree with a given root. Then we will check :

1. If the node’s value is greater than the maximum value in the left subtree or not
2. The node’s value is lesser than the minimum value in the right subtree or not. Basically, we will check if this expression holds true or not: **getMax(root.left) < root.val < getMin(root.right)**

Pseudo-Code

```
int getMin(root)
{
   BSTNode temp = root
```

```
    while(temp.left != NULL)
        temp = temp.left
    return temp.val
}


int getMax(root)
{
    BSTNode temp = root
    while(temp.right != NULL)
        temp = temp.right
    return temp.val
}


boolean isBST(BSTNode root)
{
    if (root == NULL)
        return True

    int max_left = getMax(root.left)
    int min_right = getMin(root.right)
```

```
    if (max_left > root.val || min_right < root.val)
        return False

    if (isBST(root.left) && isBST(root.right))
        return True
    return False
}
```

Complexity Analysis

We are traversing and calling **getMax(root.left)** and **getMin(root.right)** for each node.

Time complexity: n * (Time complexity of finding max in left subtree + Time complexity of finding min in right subtree) = n * O(n) **(Think!)**= $O(n^2)$

Space Complexity: O(height), stack space for recursion in worst case = O(n) (**Why?**)

Critical Ideas to think

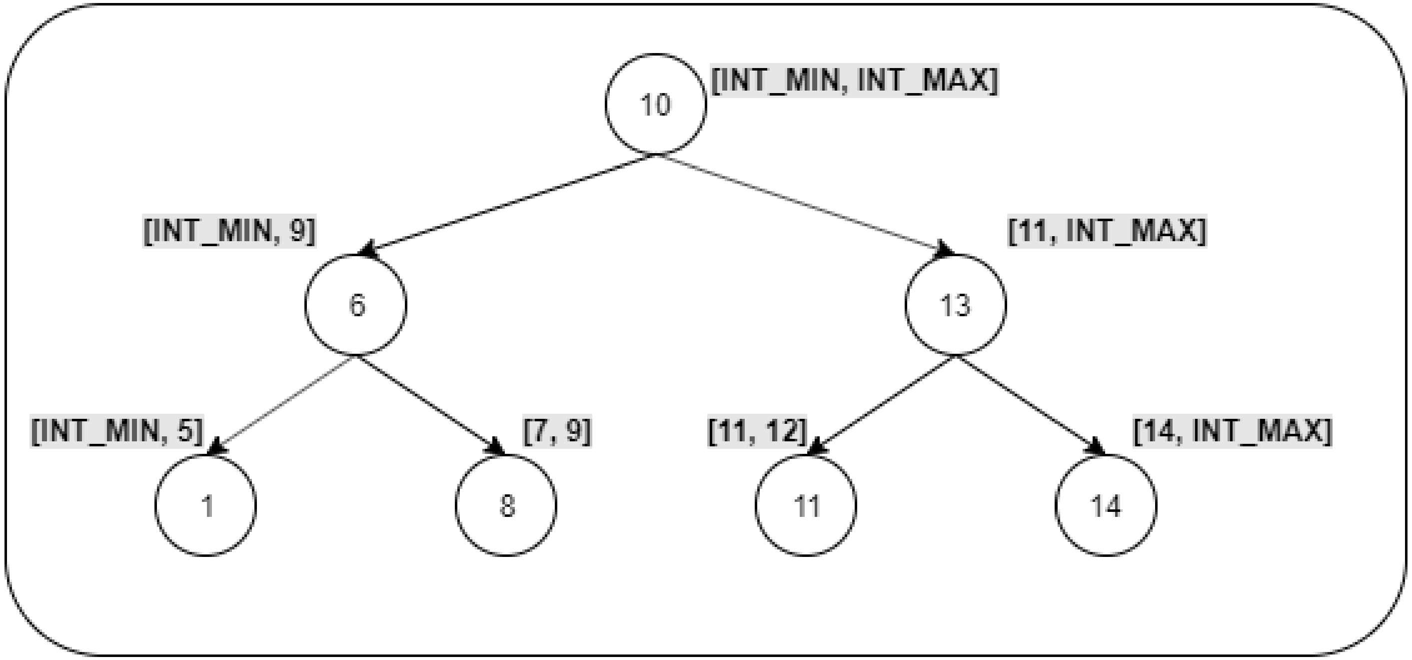
- Think about recursive implement of **getMin(root)** and **getMax(root)**. Also, compare the time and space complexity of both the implementation.
- Try to explore the best and worst-case scenario of time and space complexity.
- Can we design a better solution which checks each node only once?

2. Optimized Brute Force

In the previous approach, we are traversing some parts of the tree many times. Instead of calling getMin(root) and getMax(root) for each node, we traverse down the tree keeping track of min and max allowed value for each node.

To achieve this we pass the allowed range as a function argument while recursing for left and right subtree. Here we are looking at each node only once and initial values for min and max should be INT_MIN and INT_MAX.

For example, The allowed ranges are denoted in square brackets near the node



Pseudo-Code

```
boolean helper(BSTNode root, int range_min, int range_max)
{
    if(root == NULL)
        return True

    if (root.val < range_min || root.val > range_max)
        return False
```

```
    if (helper(root.left, range_min, root.val-1))
        if (helper(root.right, root.val+1, range_max))
            return True
    return False
}

boolean isBST(BSTNode root)
{
    if (helper(root, INT_MIN, INT_MAX))
        return True
    return False
}
```

Complexity Analysis

Here each node is visited only once.

Time Complexity: O(n)

Space Complexity: O(height), recursion stack space in worst case = O(n)

Critical Ideas to think

- How tracking the min and max node range help us to solve the problem? Try to explore the code via an example
- Why are we passing parameter range_max = **root.value-1** for the left subtree and range_min = **root.value + 1** for the right subtree?

3. Using Inorder Traversal

We know that the inorder traversal of a binary search tree gives a sorted order of its elements. We shall use this to our advantage and traverse the tree in inorder fashion and store the elements in an array. We shall then traverse the array in a linear manner and check if its in sorted order or not.

Pseudo-code

```
void storeInorder(BSTNode root, int arr[])
{
   if (root == NULL)
```

```
        return
    storeInorder(root.left, arr)
    arr.append(root.val)
    storeInorder(root.right, arr)
}
boolean isBST(BSTNode root)
{
    int arr[]   // Auxiliary array to store inorder
    storeInorder(root, arr) // Traverses the array in inorder fashion and stores the result in arr
    // Lets check if arr is in sorted order or not
    for(i = 1 to arr.length - 1)
        if (arr[i] < arr[i-1])
            return False
    return True
}
```

Complexity Analysis

Time Complexity: Inorder traversal of BST for storing elements in arr[] + Single loop to check arr[] is sorted or not = O(n) + O(n) = O(n)

Space Complexity: O(n) for storing elements in an array + O(height), for recursion stack space in worst case = O(n)

Critical Ideas to think

- Can we solve this problem without using Auxiliary Array?
- Explore problems in the BST which can be solved using the in-order traversal.

Comparison of different solutions

Approach	Time Complexity	Space Complexity
Brute Force	$O(n^2)$	O(n)
Optimized Brute Force	O(n)	O(n)
Using inorder property	O(n)	O(n)

Suggested problems to solve

- Check if the given binary tree is a full binary tree or not
- Check whether the given binary is perfect or not
- Check if a binary tree is a subtree of another binary subtree
- Check if a binary tree is a complete tree or not
- Check if given binary tree is Heap or not
- Check if given binary tree is SumTree or not

Please comment down below if you have alternative approaches or find an error/bug in the above approaches.

Happy Coding! Enjoy Algorithms!

VIDEO

System Design - Interview

This System Design series will help you in designing better software applications as well as in cracking the System Design Interview.

WATCH NOW

Share this blog and spread the knowledge

SHARE ON FACEBOOK

SHARE ON TWITTER

SHARE ON LINKEDIN

SHARE ON TELEGRAM

SHARE ON REDDIT

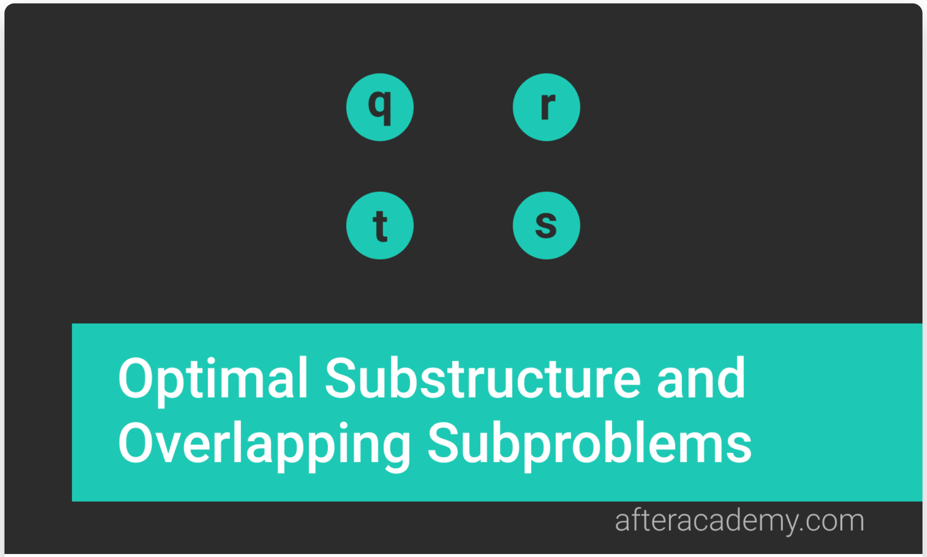
SHARE ON WHATSAPP

0 Comments Sort by Oldest

Add a comment...

Facebook Comments Plugin

Recommended for You



Binary Tree Zig Zag Level Order Traversal

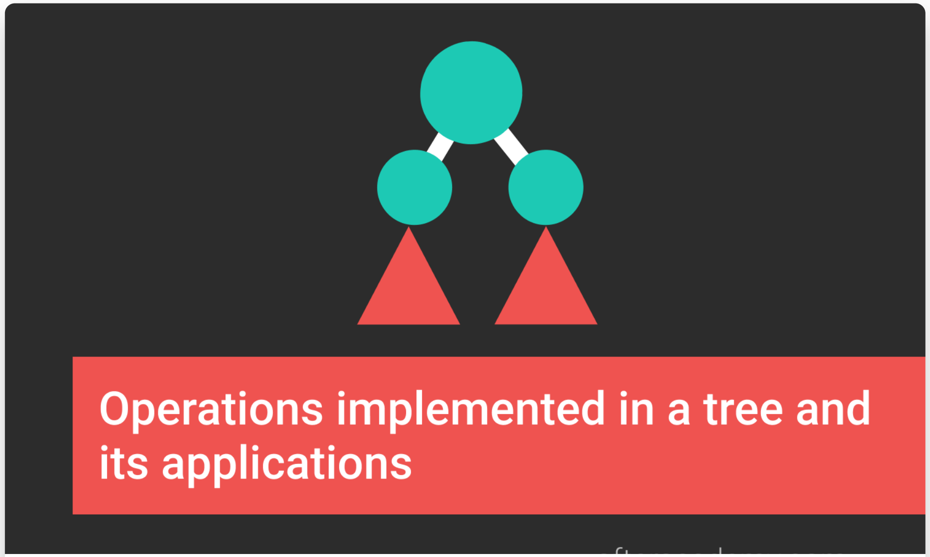
Given a binary tree, return the zigzag level order traversal of its nodes' values. (i.e, from left to right, then right to left for the next level and alternate between). The problem is a typical Interview problem based on Tree Traversal

Admin AfterAcademy
4 May 2020

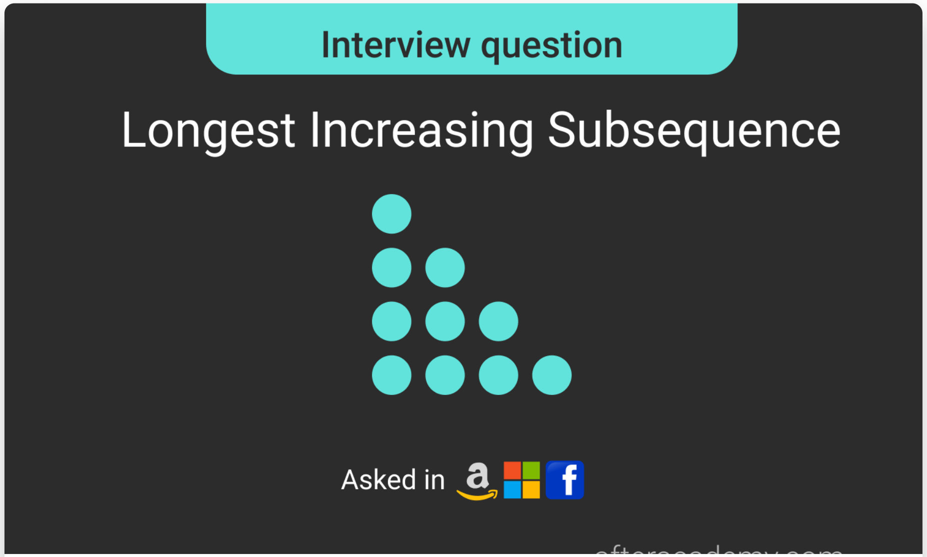
Optimal Substructure and Overlapping Subproblems

In this blog, we will understand the optimal substructure and overlapping subproblems property. We will also discuss how the problems having these two properties can be solved using Dynamic programming.

Admin AfterAcademy
1 May 2020



What are the operations implemented in tree and what are its



Longest Increasing Subsequence - Interview Problem

applications?

Tree Data Structure supports various operations such as insertion, deletion and searching of elements along with few others. This blogs discussed those operations along with explaining its applications.

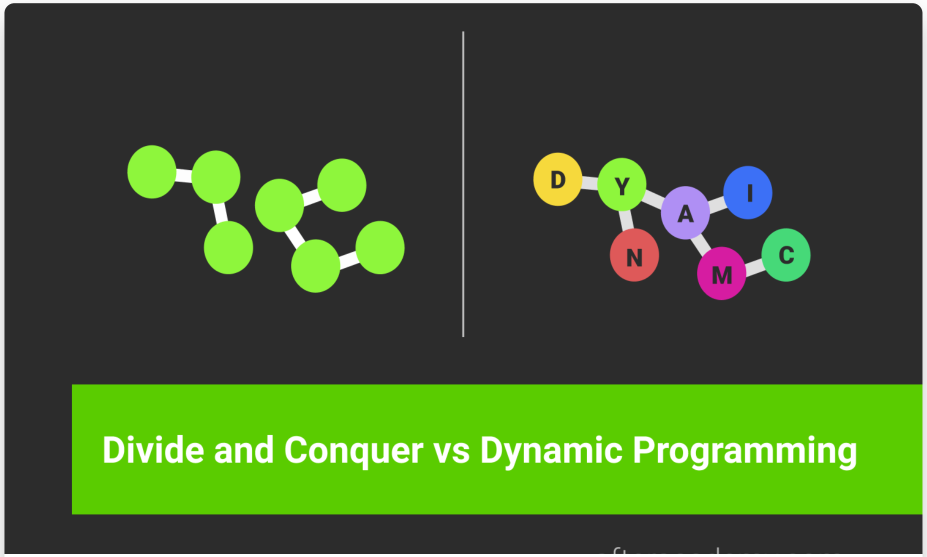
Admin AfterAcademy
11 Feb 2020

Greatest Common Divisor-Interview Problem

This is a mathematical problem asked in interviews of Google like companies. For solving such problems you need to have some math skills and

You are given an array A with N elements. You need to find the longest increasing subsequence in the array.

Admin AfterAcademy
7 Apr 2020



Divide & Conquer vs Dynamic Programming

Divide and Conquer and Dynamic Programming are two most used problem-solving skills in Data Structures. In this blog, we will see the similarities also some idea of algorithms like Euclid's GCD algorithm.

Admin AfterAcademy
10 Feb 2020

and difference between Dynamic Programming and Divide-and-Conquer approaches.

Admin AfterAcademy
10 Feb 2020

Our Learners Work At

AfterAcademy

Stay up to date. Follow us on

MindOrks Nextgen Private Limited
Gurgaon, Haryana, India
+91-8287460223

Quick Links

Contact Us
Privacy Policy
Terms And Conditions
Cookie Policy

About Us

MindOrks
Amit Shekhar
Janishar Ali

Free Resources

Publication
Medium
Video Lessons
Open Source