GeeksforGeeks

A computer science portal for geeks

Placements Practice GATE CS IDE Q&A
GeeksQuiz

Check if a given array can represent Preorder Traversal of Binary Search Tree

Given an array of numbers, return true if given array can represent preorder traversal of a Binary Search Tree, else return false. Expected time complexity is O(n).

Examples:

```
Input:  pre[] = {2, 4, 3}
Output: true
Given array can represent preorder traversal
of below tree
    2
     \
      4
     /
    3

Input:  pre[] = {2, 4, 1}
Output: false
Given array cannot represent preorder traversal
of a Binary Search Tree.

Input:  pre[] = {40, 30, 35, 80, 100}
Output: true
Given array can represent preorder traversal
of below tree
     40
   /   \
 30    80
  \      \
  35     100


Input:  pre[] = {40, 30, 35, 20, 80, 100}
Output: false
```

```
Given array cannot represent preorder traversal
of a Binary Search Tree.
```

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is to do following for every node pre[i] starting from first one.

```
1) Find the first greater value on right side of current node.
   Let the index of this node be j. Return true if following
   conditions hold. Else return false
    (i)  All values after the above found greater value are
         greater than current node.
    (ii) Recursive calls for the subarrays pre[i+1..j-1] and
         pre[j+1..n-1] also return true.
```

Time Complexity of the above solution is O(n^2)

An **Efficient Solution** can solve this problem in O(n) time. The idea is to use a stack. This problem is similar to Next (or closest) Greater Element problem. Here we find next greater element and after finding next greater, if we find a smaller element, then return false.

```
1) Create an empty stack.
2) Initialize root as INT_MIN.
3) Do following for every element pre[i]
     a) If pre[i] is smaller than current root, return false.
     b) Keep removing elements from stack while pre[i] is greater
        then stack top. Make the last removed item as new root (to
        be compared next).
        At this point, pre[i] is greater than the removed root
        (That is why if we see a smaller element in step a), we
        return false)
     c) push pre[i] to stack (All elements in stack are in decreasing
        order)
```

Below is implementation of above idea.

C++

```cpp
// C++ program for an efficient solution to check if
// a given array can represent Preorder traversal of
// a Binary Search Tree
#include<bits/stdc++.h>
using namespace std;

bool canRepresentBST(int pre[], int n)
{
    // Create an empty stack
    stack<int> s;

    // Initialize current root as minimum possible
    // value
    int root = INT_MIN;

    // Traverse given array
    for (int i=0; i<n; i++)
    {
        // If we find a node who is on right side
        // and smaller than root, return false
        if (pre[i] < root)
            return false;

        // If pre[i] is in right subtree of stack top,
        // Keep removing items smaller than pre[i]
        // and make the last removed item as new
        // root.
        while (!s.empty() && s.top()<pre[i])
        {
            root = s.top();
            s.pop();
        }

        // At this point either stack is empty or
        // pre[i] is smaller than root, push pre[i]
        s.push(pre[i]);
    }
    return true;
}

// Driver program
int main()
{
    int pre1[] = {40, 30, 35, 80, 100};
    int n = sizeof(pre1)/sizeof(pre1[0]);
    canRepresentBST(pre1, n)? cout << "true\n":
                              cout << "false\n";

    int pre2[] = {40, 30, 35, 20, 80, 100};
    n = sizeof(pre2)/sizeof(pre2[0]);
    canRepresentBST(pre2, n)? cout << "true\n":
                              cout << "false\n";

    return 0;
}
```

Run on IDE

Java

```
// Java program for an efficient solution to check if
// a given array can represent Preorder traversal of
// a Binary Search Tree
import java.util.Stack;

class BinarySearchTree {

    boolean canRepresentBST(int pre[], int n) {
        // Create an empty stack
        Stack<Integer> s = new Stack<Integer>();

        // Initialize current root as minimum possible
        // value
        int root = Integer.MIN_VALUE;
        // Traverse given array
        for (int i = 0; i < n; i++) {
            // If we find a node who is on right side
            // and smaller than root, return false
            if (pre[i] < root) {
                return false;
            }

            // If pre[i] is in right subtree of stack top,
            // Keep removing items smaller than pre[i]
            // and make the last removed item as new
            // root.
            while (!s.empty() && s.peek() < pre[i]) {
                root = s.peek();
                s.pop();
            }

            // At this point either stack is empty or
            // pre[i] is smaller than root, push pre[i]
            s.push(pre[i]);
        }
        return true;
    }

    public static void main(String args[]) {
        BinarySearchTree bst = new BinarySearchTree();
        int[] pre1 = new int[]{40, 30, 35, 80, 100};
        int n = pre1.length;
        if (bst.canRepresentBST(pre1, n) == true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
        int[] pre2 = new int[]{40, 30, 35, 20, 80, 100};
        int n1 = pre2.length;
        if (bst.canRepresentBST(pre2, n) == true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
    }
}

//This code is contributed by Mayank Jaiswal
```

Run on IDE

Python

```
# Python program for an efficient solution to check if
# a given array can represent Preorder traversal of
# a Binary Search Tree

INT_MIN = -2**32

def canRepresentBST(pre):

    # Create an empty stack
    s = []

    # Initialize current root as minimum possible value
    root = INT_MIN
    # Traverse given array
    for value in pre:
        #NOTE:value is equal to pre[i] according to the
        #given algo

        # If we find a node who is on the right side
        # and smaller than root, return False
        if value < root :
            return False

        # If value(pre[i]) is in right subtree of stack top,
        # Keep removing items smaller than value
        # and make the last removed items as new root
        while(len(s) > 0 and s[-1] < value) :
            root = s.pop()

        # At this point either stack is empty or value
        # is smaller than root, push value
        s.append(value)

    return True

# Driver Program
pre1 = [40 , 30 , 35 , 80 , 100]
print "true" if canRepresentBST(pre1) == True else "false"
pre2 = [40 , 30 , 35 , 20 ,  80 , 100]
print "true" if canRepresentBST(pre2) == True else "false"

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

Run on IDE

Output:

```
true
false
```

This article is contributed by **Romil Punetha**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

20 Comments Category: Stack Trees

Related Posts:

- Minimum number of bracket reversals needed to make an expression balanced
- Iterative Depth First Traversal of Graph
- Sort a stack using recursion
- Length of the longest valid substring
- Find maximum of minimum for every window size in a given array
- Iterative Tower of Hanoi
- How to efficiently implement k stacks in a single array?
- Print ancestors of a given binary tree node without recursion

Like Share 14 people like this.

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

20 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

sai • 19 days ago

My solution

http://ideone.com/GXml4A

My logic is to just make bst insertion but while inserting a left check whether a right is there or not.
if right is available means return false
else true.

• Reply • Share ›

Anandan Arumugam • a month ago

Would the complexity still be O(N)? if I,

InsertBST() - O(N) from given array
PreOrderBST -O(N) and store it in another array.
Compare - O(N) (given array and new pre-order array)

• Reply • Share ›

Jayesh • 2 months ago

Easy to understand explanation of problem statement and solution in Java.

http://javabypatel.blogspot.in...

• Reply • Share ›

AlienOnEarth • 2 months ago

GeeksforGeeks The recursive solution is more intuitive than iterative.

```
bool isPreorder(int pre[], int *index, int n, int min, int max){

if(index == n-1){

if (pre[*index]<min ||="" pre[*index]="">max)

return false;

else

return true;

}

if(pre[*index]<min ||="" pre[*index]="">max)

return false;

int root = pre[*index];
```

see more

• Reply • Share ›

Yeshwanth Selvaraj • 2 months ago

the next grater element link is broken..

^ | ⌄ • Reply • Share ›

GeeksforGeeks Mod ➦ Yeshwanth Selvaraj • 2 months ago

Thanks for pointing this out. We have corrected the link.

^ | ⌄ • Reply • Share ›

Raj ➦ GeeksforGeeks • a month ago

@GeeksforGeeks:
For the simple solution posted above, it should be as following:

(i) All values after the above found greater value are
greater than current node.
(ii) Recursive calls for the subarrays pre[i+1..j-1] and
pre["j"..n-1] also return true.

The second subarray should start from j and not j+1

^ | ⌄ • Reply • Share ›

Ankit Aggarwal • 3 months ago

I am using idea of creating BST from preorder traversal. If there is something wrong please comment:

```
int canRepresentPreorderTraversal(int *pre, int size, int *preIndex, int key, int min, int
max) {
if(*preIndex >= size) {
return 1;
}

if(key <= min || key >= max) {
return 0;
}

*preIndex = *preIndex + 1;

if(*preIndex < size) {
return canRepresentPreorderTraversal(pre, size, preIndex, pre[*preIndex], min, key)
|| canRepresentPreorderTraversal(pre, size, preIndex, pre[*preIndex], key, max);
}
}
```

^ | ⌄ • Reply • Share ›

Haresh Chudgar • 3 months ago

@Romil Shouldn't the code return false for the following input:
{40, 30, 35, 80, 90, 100}

{40, 30, 35, 80, 90, 100}

80,90,100 violates the BST rule that value at left should be less than root.

⌃ | ⌄ • Reply • Share ›

Saurabh Verma • 3 months ago

A small correction:
In the line: The idea is to use a stack. This problem is similar toNext (or closest) Greater Element problem the link is pointing to wrong url. The url should be http://www.geeksforgeeks.org/n... instead of http://www.geeksforgeeks.org/e...

⌃ | ⌄ • Reply • Share ›

stack26 • 3 months ago

@GeeksforGeeks:Please explain the second method clearly. Its just quite unclear and misleading

1 ⌃ | ⌄ • Reply • Share ›

Anuj • 4 months ago

Can't we do it like

http://www.geeksforgeeks.org/a...

method-3
where we pass int_max and int_min and checking

⌃ | ⌄ • Reply • Share ›

Aditya Gaur • 5 months ago

"Recursive calls for the subarrays pre[i+1..j-1] and
pre[j+1..n-1] also return true"

Should this be pre[j...n-1] ?

⌃ | ⌄ • Reply • Share ›

drinetri hunt • 5 months ago

@geeksforgeeks@Romil Punetha for example {2,1,4} it is showing true but actual answer is false

⌃ | ⌄ • Reply • Share ›

vito ➦ drinetri hunt • 5 months ago

Kartik's answer is right.

⌃ | ⌄ • Reply • Share ›

Kartik ➦ drinetri hunt • 5 months ago

We can construct below tree wit {2, 1, 4}
2

```
/ \
1 4
```

2 • Reply • Share ›

Anonymous • 5 months ago

Solution in C without using extra O(n) space for stack. Let me know your comments.

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <limits.h>

int checkBFST(int *a, int n)
{
int i, min = INT_MIN, cur_max, prev, cur;
if (n <= 2)
return 1;

prev = a[0];
cur_max = prev;

for (i = 1; i < n; i++)
{
```

see more

• Reply • Share ›

Anonymous • 5 months ago

I think there is a typo -
In explanation it is written as -

a) If pre[i] is greater than current root, return false.

but in code it is written as -

```
if (pre[i] < root)
return false;
```

• Reply • Share ›

GeeksforGeeks Mod ➦ Anonymous • 5 months ago

Thanks for pointing this out. We have corrected the algorithm.

• Reply • Share ›

Koustav Chatterjee • 5 months ago

java recursive soln http://ideone.com/1Zbdxb

• Reply • Share ›