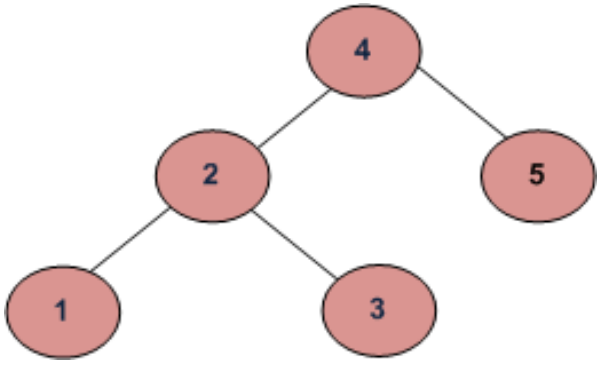
A program to check if a binary tree is BST or not

A binary search tree (BST) is a node based binary tree data structure which has the following properties.

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

From the above properties it naturally follows that:

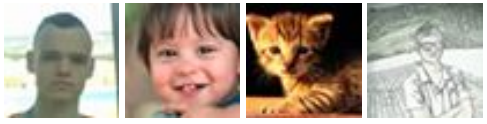
- Each node (item in the tree) has a distinct key.



METHOD 1 (Simple but Wrong)

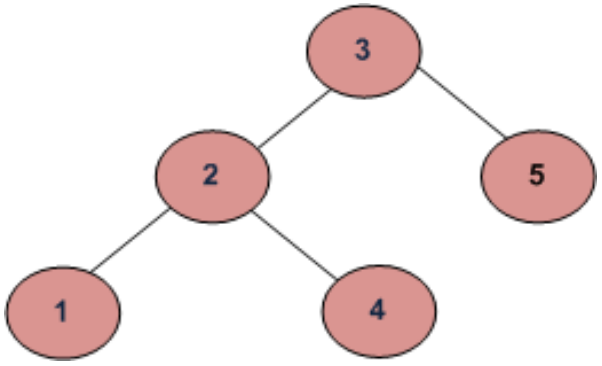
Following is a simple program. For each node, check if left node of it is smaller than the node and right node of it is greater than the node.

```
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;
```

```
  /* false if left is > than node */
  if (node->left != NULL && node->left->data > node->data)
    return 0;

  /* false if right is < than node */
  if (node->right != NULL && node->right->data < node->data)
    return 0;

  /* false if, recursively, the left or right is not a BST */
  if (!isBST(node->left) || !isBST(node->right))
    return 0;

  /* passing all that, it's a BST */
  return 1;
}
```

This approach is wrong as this will return true for below binary tree (and below tree is not a BST because 4 is in left subtree of 3)



METHOD 2 (Correct but not efficient)
For each node, check if max value in left subtree is smaller than the node and min value in right subtree greater than the node.

```
/* Returns true if a binary tree is a binary search tree */
int isBST(struct node* node)
{
  if (node == NULL)
    return(true);
```


Popular Posts

```
  /* false if the max of the left is > than us */
  if (node->left!=NULL && maxValue(node->left) > node->data)
    return(false);

  /* false if the min of the right is <= than us */
  if (node->right!=NULL && minValue(node->right) < node->data)
    return(false);

  /* false if, recursively, the left or right is not a BST */
  if (!isBST(node->left) || !isBST(node->right))
    return(false);

  /* passing all that, it's a BST */
  return(true);
}
```

It is assumed that you have helper functions minValue() and maxValue() that return the min or max int value from a non-empty tree

METHOD 3 (Correct and Efficient)

Method 2 above runs slowly since it traverses over some parts of the tree many times. A better solution looks at each node only once. The trick is to write a utility helper function isBSTUtil(struct node* node, int min, int max) that traverses down the tree keeping track of the narrowing min and max allowed values as it goes, looking at each node only once. The initial values for min and max should be INT_MIN and INT_MAX — they narrow from there.

```
/* Returns true if the given tree is a binary search tree
 (efficient version). */
int isBST(struct node* node)
{
  return(isBSTUtil(node, INT_MIN, INT_MAX));
}

/* Returns true if the given tree is a BST and its
 values are >= min and <= max. */
int isBSTUtil(struct node* node, int min, int max)
```

Implementation:

Implementation:

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

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

int isBSTUtil(struct node* node, int min, int max);

/* Returns true if the given tree is a binary search tree
 (efficient version). */
int isBST(struct node* node)
{
  return(isBSTUtil(node, INT_MIN, INT_MAX));
}

/* Returns true if the given tree is a BST and its
   values are >= min and <= max. */
int isBSTUtil(struct node* node, int min, int max)
{

  /* an empty tree is BST */
  if (node==NULL)
     return 1;

  /* false if this node violates the min/max constraint */
  if (node->data < min || node->data > max)
     return 0;

  /* otherwise check the subtrees recursively,
   tightening the min or max constraint */
  return
    isBSTUtil(node->left, min, node->data-1) &&  // Allow only distinc
    isBSTUtil(node->right, node->data+1, max);  // Allow only distinct
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
```


Recent Comments

karthik it should have been max_wrap= max_wrap -...

Maximum circular subarray sum · 1 minute ago

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 45 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 1 hour ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 1 hour ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 3 hours ago

```
{
  struct node* node = (struct node*)
                      malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;

  return(node);
}

/* Driver program to test above functions*/
int main()
{
  struct node *root = newNode(4);
  root->left        = newNode(2);
  root->right       = newNode(5);
  root->left->left  = newNode(1);
  root->left->right = newNode(3);

  if(isBST(root))
    printf("Is BST");
  else
    printf("Not a BST");

  getchar();
  return 0;
}
```

Time Complexity: O(n)
Auxiliary Space : O(1) if Function Call Stack size is not considered, otherwise O(n)

METHOD 4(Using In-Order Traversal)
Thanks to LJW489 for suggesting this method.
1) Do In-Order Traversal of the given tree and store the result in a temp array.
3) Check if the temp array is sorted in ascending order, if it is, then the tree is BST.

Time Complexity: O(n)

We can avoid the use of Auxiliary Array. While doing In-Order traversal, we can keep track of previously visited node. If the value of the currently visited node is less than the previous value, then tree is not BST. Thanks to ygos for this space optimization.

```
bool isBST(struct node* root)
{
```

```
    static struct node *prev = NULL;

    // traverse the tree in inorder fashion and keep track of prev nod
    if (root)
    {
        if (!isBST(root->left))
          return false;

        // Allows only distinct valued nodes
        if (prev != NULL && root->data <= prev->data)
          return false;

        prev = root;

        return isBST(root->right);
    }

    return true;
}
```

The use of static variable can also be avoided by using reference to prev node as a parameter (Similar to this post).

Sources:
http://en.wikipedia.org/wiki/Binary_search_tree
http://cslibrary.stanford.edu/110/BinaryTrees.html

Please write comments if you find any bug in the above programs/algorithms or other ways to solve the same problem.

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

Writing code in comment? Please use **ideone.com** and share the link here.

124 Comments GeeksforGeeks

Sort by Newest

Anjaneya Alluri • 25 days ago

I believe we can do either of the Traversals for this problem i.e Inorder, Preord

And while we are traversing and adding nodes to the stack , we can make con false if any of which breaks it.

Eg: , please find the below using Pre Order traversal.

```
public boolean BST_check(Node<t> root){
//check for null

if(root == null){
System.out.println("sent null , please recheck the BST");
return false;
}

Node<t> cur = root;

//check for right and left sub trees
boolean done = false;
Stack<node<t>> stk = new Stack<node<t>>();
```

see more

Reply • Share ›

Nitin ➦ Anjaneya Alluri • 10 days ago

preorder and postorder traversals of a BST are not guaranteed to be in

Reply • Share ›

Varun • 2 months ago

should it not be root->data > prev->data instead of root->data <= prev->data ir

Reply • Share ›

varun Varun • 2 months ago

my bad !! it is correct only...

• Reply • Share ›

Guest • 2 months ago

Could someone please point out the mistake in this code(if any):
assumption: null tree is not bst

```
bool is_bst(tree* tree 1)
{
if(tree1==NULL) return false;

if(tree1->left && tree1-> right)
return((tree1->left->value<tree1->value) && (tree1->right->value>tree1->value
is_bst(tree1->right));

else if(tree1->right)
return((tree1->right->value>tree1->value) && is_bst(tree1->right));

else if(tree1->left)
return((tree1->left->value<tree1->value) && is_bst(tree1->left));

return true;
}
```

• Reply • Share ›

dmr • 3 months ago

A doubt in method 4, if someone can explain. Why can't I replace below lines:

```
if (!isBST(root->left))
        return false;
```

WITH:

```
return isBST(root->left)
```

return isBST(root->left)

I get wrong answer if I does so.
But per my understanding, they are same..NO ?

• Reply • Share ›

gourav pathak ➚ dmr • 3 months ago

No they are different
When you write "if(!isBST(root->left) return false"
it means if left subtree is not a bst then return false(if left subtree is a b
further if right subtree is also a bst)

But when you write return isBST(root->left) it returns true if left subtree
to further check whether right subtree is a bst or not

• Reply • Share ›

dmr ➚ gourav pathak • 3 months ago

ok....i get your point. But when then we write "return isBST(root
Shouldn't it be handled like left subtree case ? like: if(!isBST(roo

• Reply • Share ›

gourav pathak ➚ dmr • 3 months ago

No it need not be..... We will reach there only if the left s
would return false).....Now since we know that left subtr
whether right subtree is a BST or not....if(isBST(root->ri

• Reply • Share ›

123 • 4 months ago

Won't the method 3 fail when one of the nodes have value either INT_MIN or IN
wrong

2 • Reply • Share ›

Nikhil Agrawal • 4 months ago

In Method 3:
There is no need to subtract and add 1 from root.data because in comparison itself will differentiate values resulting in unique values.
Correct me if I am wrong

^ | v • Reply • Share ›

Mukesh • 5 months ago

Method-4 is not good for above example. just replace 6 with 1. If last leave nod working.

^ | v • Reply • Share ›

RASHMIBS • 5 months ago

can anybody tell me here how to check for negative condition??? say suppose

^ | v • Reply • Share ›

Zeest ➚ RASHMIBS • 5 months ago

Common man isnt that really a trivial thing..

```
isNotBst(sturct node* root){
return 1^(isBst(root))
}
```

^ | v • Reply • Share ›

RASHMIBS ➚ Zeest • 5 months ago

ok thanks Zeest ,will check once in my code

1 ^ | v • Reply • Share ›

shruthi • 6 months ago

While implementing method 3

Shouldn't it be isBSTUtil(node->left, min, node->data) and not node->data -1 b

or equal to the node and not just lesser.

Reply • Share ›

Joao Brunet shruthi • 5 months ago

No. A binary search tree does not contains duplicate elements. The lef must be greater than root.

Reply • Share ›

João Brunet Joao Brunet • 5 months ago

*does not contain.

Reply • Share ›

Victor João Brunet • 2 months ago

That's actually not true. All nodes in the left subtree mus the root is less than all nodes in the right subtree

Reply • Share ›

Victor Victor • 2 months ago

must be*

Reply • Share ›

Mahendra • 6 months ago

I think here is a simple logic/solution to the given problem:

int isBST(struct node* root){

if(root){

if((root->left==NULL || root->left->data<root->data)&&(root->right==NULL || roc

if((root->left==0 || root->left->right==0 || root->left->right->data<root->data) && root->right->left->data>root->data))

```
return isBST(root->left)&&isBST(root->right);

else

return 0;

}

else

return 0;

}

return 1;

}
```

1 ^ | v • Reply • Share ›

GuestPost • 6 months ago

using in order traversal is good.. a little more optimization is possible
1) no need to have a temp array.
2) have two variables previous and present
3) each time check previous < present. Else break + stop in order traversal

will be a little more space and time optimized.

1 ^ | v • Reply • Share ›

Sameer • 7 months ago

http://codingrecipies.blogspot...
Very well explained binary search tree

^ | v • Reply • Share ›

pavansrinivas • 7 months ago

Code in Java using InOrder Traversal

```
boolean isBst(){
        Stack< Node> s = new Stack<>();
        Node temp = root;
        boolean isFirst = true;
        int cur=root.i,prev=-1;
        while(true){
                while(temp!=null){
                        s.push(temp);
                        temp = temp.left;
                }
                if(s.isEmpty()){
                        break;
                }
                temp = s.pop();\
                if(isFirst){
                        cur = temp.i;
```

see more

Reply • Share ›

Kuldeep Kumar • 7 months ago

```
#include<iostream>
#include<limits.h>
using namespace std;
int flag=1;
//int prev=INT_MIN;
int current=INT_MIN;
void checkBst(node *t){
        if(t){
                checkBst(t->left);
```

```
                if(t->data < current)
                        flag=0;
                current=t->data;
checkBst(t->right);
                        }
        }
int main(){
node *t=NULL,*root;
root = (node*)malloc(sizeof(node));
```

see more

1 ^ | v • Reply • Share ›

praveen kumar • 7 months ago

in method if given tree has root 100 and right child of root is also 100 then it nc binary tree ...its not a good method in that case.....am i correct??

2 ^ | v • Reply • Share ›

bakwasscoder ➦ praveen kumar • 5 months ago

Method is correct.....it's returning false: http://ideone.com/UDKcW9

^ | v • Reply • Share ›

kp ➦ praveen kumar • 7 months ago

which one method..??

1 ^ | v • Reply • Share ›

praveen ➦ kp • 7 months ago

method 4...using in-order traversal

5 ^ | v • Reply • Share ›

praveen kumar ➦ praveen • 7 months ago

yaah i also agree with yamini..its not a good method in t

viki • 8 months ago

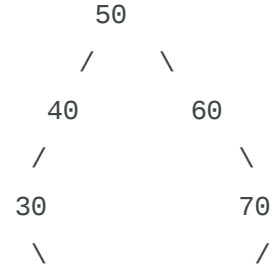
//Will this solution work ??

```
int isBST(node *root)

{

if(!root)return 1;

if(root->left && root->data < root->left->data)return 0;

if(root->right && root->data > root->right->data)return 0;

if(root->left &&root->left->right && root->data <root->left->right->data)return 0;

if(root->right &&root->right->left && root->data >root->right->left->data)return

if(isBST(root->left)&&isBST(root->right))return 1;

return 0;

}
```

1 • Reply • Share ›

Shiwakant Bharti ➦ viki • 7 months ago

The trick looks good to me at least on below test case. Is it inspired fro



```
                35          65
```

What about this? Looks like this logic also works on min and max bour

```
              280
                \
                 410
                 /
               290
```

see more

• Reply • Share ›

Anmol Kelkar • 9 months ago

Anukul Mohil This is a bst but not a balanced one!

• Reply • Share ›

Anukul Mohil • 9 months ago

Sandeep Jain what about this Input for Method 4

20(root)

20(right child)

this is not a bst but still method 4 flags it as one ?

• Reply • Share ›

Shiwakant Bharti ➦ Anukul Mohil • 7 months ago

@Anukul Mohil I checked on my machine and it returns false. Can you corrected it now!

• Reply • Share ›

rohit • 10 months ago

What will be the time complexity of method 2.Will it be O(NlogN)??

1 ^ | v • Reply • Share ›

Shiwakant Bharti ➦ rohit • 7 months ago

Only if the tree is balanced. Or else either of the minValue() or maxVal
worst case.

1 ^ | v • Reply • Share ›

Sandeep Jain • 10 months ago

O(Logn) doesn&#039t seem possible as we have to visit every node to find ou

2 ^ | v • Reply • Share ›

Munish Singla • 10 months ago

can any one tell me the implementation in o(log n).

1 ^ | v • Reply • Share ›

Shiwakant Bharti ➦ Munish Singla • 7 months ago

Is it even possible do so in worst case without even visiting all nodes C

1 ^ | v • Reply • Share ›

innosam • 11 months ago

http://innosamcodes.wordpress....

Check out this simple program, just introduced bool for checking if max/min e

1 ^ | v • Reply • Share ›

abhishek08aug • a year ago

C++ code:

```
 #include <iostream>
#include <stdlib.h>
using namespace std;
```

```
using namespace std;

class tree_node {
  private:
    int data;
    tree_node * left;
    tree_node * right;
  public:
    tree_node() {
      left=NULL;
      right=NULL;
    }
    void set_data(int data) {
      this->data=data;
```

see more

1 ^ | v • Reply • Share ›

Rudra ➦ abhishek08aug • 9 months ago

Great :D

```
/* Paste your code here (You may delete these lines if not wri
```

^ | v • Reply • Share ›

aman1234 ➦ abhishek08aug • 11 months ago

intelligent :D

1 ^ | v • Reply • Share ›

prity ➦ aman1234 • 9 months ago

Geek :)

1 ^ | v • Reply • Share ›

Pradeep • a year ago

Another solution (in C#) using Inorder using a stack

[sourcecode language="C#"]

```
/*
public bool IsBstInorder()
{
if (root == null) return true;
Stack<BinaryTreeNode<T>> stack = new Stack<BinaryTreeNode<T>>();
BinaryTreeNode<T> temp = root;
while (true)
{
for (; temp != null; temp = temp.Left)
{
if ((stack.IsEmpty()) ||((!stack.IsEmpty())&& (temp.item.CompareTo(stack.Top
in inorder, item going to push
stack.Push(temp);//should be greater than all the processed items
else
return false;
}

if (stack.IsEmpty()) return true;
BinaryTreeNode<T> node = stack.Pop();
temp = node.Right;
}
}*/
```

 • Reply • Share ›

Anish P • a year ago

```
public class Node {
private Node leftChild;
private int data;
```

```
 private Node rightChild;
}
private boolean isBST(Node node){ //Here node is the root when //this
 if (node != null) {
  isBST(node.lefChild);
  if (previous != null && previous.data >= node.data) {
   return false;
  }
  return isBST(node.rightChild);
 }
 return true;
}
```

Hi All, Please let me know if the above solution works.Have used the same me

Thanks
Anish P


Shiwakant Bharti Anish P • 7 months ago

The contribution of the bst checking of the left child is missing in the fin answer(s).


adsf • a year ago

void CheckBST(struct node* root)
{
int flag=1,min=-32768;
if(root==null)
{
printf("Empty Tree");

```
return;
}
IsBst(root,&min,&flag);
if(flag==1)
printf("Binary Search Tree");
else
printf("Not a Binary Search Tree");

}

void IsBst(struct node* node,int *min,int *flag)
{
if(node!=null)
```

see more

• Reply • Share ›

AdityaSraf • a year ago

Lots of typing errors in my last comment. Here it is again: Traverse all nodes s
node, temporarily store the value of its parent also(obviously except for root), t
Case1: current node < parent
2 checks:
1)left child <current node="" 2)right="" child="">current node && right child<pa
parent
2 checks:
1) left child <current node="" &&="" left="" child="">parent
2) right child>current node

• Reply • Share ›

Load more comments