Two nodes of a BST are swapped, correct the BST

Difficulty Level : Hard • Last Updated : 15 Jul, 2022

Two of the nodes of a Binary Search Tree (BST) are swapped. Fix (or correct) the BST.

```
Input Tree:
         10
        /  \
       5    8
      / \
     2   20

In the above tree, nodes 20 and 8 must be swapped to fix the tree.
Following is the output tree
         10
        /  \
       5    20
      / \
     2   8
```

The inorder traversal of a BST produces a sorted array. So a **simple method** is to store inorder traversal of the input tree in an auxiliary array. Sort the auxiliary array. Finally, insert the auxiliary array elements back to the BST, keeping the structure of the BST same. The time complexity of this method is O(nLogn) and the auxiliary space needed is O(n).

We can solve this in O(n) time and with a single traversal of the given BST. Since inorder traversal of BST is always a sorted array, the problem can be

reduced to a problem where two elements of a sorted array are swapped. There are two cases that we need to handle:

1. The swapped nodes are not adjacent in the inorder traversal of the BST.

```
For example, Nodes 5 and 25 are swapped in {3 5 7 8 10 15 20 25}.
The inorder traversal of the given tree is 3 25 7 8 10 15 20 5
```

If we observe carefully, during inorder traversal, we find node 7 is smaller than the previous visited node 25. Here save the context of node 25 (previous node). Again, we find that node 5 is smaller than the previous node 20. This time, we save the context of node 5 (the current node). Finally, swap the two node's values.

2. The swapped nodes are adjacent in the inorder traversal of BST.

```
For example, Nodes 7 and 8 are swapped in {3 5 7 8 10 15 20 25}.
The inorder traversal of the given tree is 3 5 8 7 10 15 20 25
```

Unlike case #1, here only one point exists where a node value is smaller than the previous node value. e.g. node 7 is smaller than node 8.

How to Solve? *We will maintain three-pointers, first, middle, and last. When we find the first point where the current node value is smaller than the previous node value, we update the first with the previous node & the middle with the current node. When we find the second point where the current node value is smaller than the previous node value, we update the last with the current node. In the case of #2, we will never find the second point. So, the last pointer will not*

be updated. After processing, if the last node value is null, then two swapped nodes of BST are adjacent.

Following is the implementation of the given code.

C++

```
// Two nodes in the BST's swapped, correct the BST.
#include <bits/stdc++.h>
using namespace std;

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

// A utility function to swap two integers
void swap( int* a, int* b )
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node *)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return(node);
}

// This function does inorder traversal to find out the two swapped nodes.
// It sets three pointers, first, middle and last.  If the swapped nodes are
// adjacent to each other, then first and middle contain the resultant nodes
// Else, first and last contain the resultant nodes
void correctBSTUtil( struct node* root, struct node** first,
                     struct node** middle, struct node** last,
                     struct node** prev )
{
    if( root )
    {
```

```
        // Recur for the left subtree
        correctBSTUtil( root->left, first, middle, last, prev );

        // If this node is smaller than the previous node, it's violating
        // the BST rule.
        if (*prev && root->data < (*prev)->data)
        {

            // If this is first violation, mark these two nodes as
            // 'first' and 'middle'
            if ( !*first )
            {
                *first = *prev;
                *middle = root;
            }

            // If this is second violation, mark this node as last
            else
                *last = root;
        }

        // Mark this node as previous
        *prev = root;

        // Recur for the right subtree
        correctBSTUtil( root->right, first, middle, last, prev );
    }
}

// A function to fix a given BST where two nodes are swapped.  This
// function uses correctBSTUtil() to find out two nodes and swaps the
// nodes to fix the BST
void correctBST( struct node* root )
{

    // Initialize pointers needed for correctBSTUtil()
    struct node *first, *middle, *last, *prev;
    first = middle = last = prev = NULL;

    // Set the pointers to find out two nodes
    correctBSTUtil( root, &first, &middle, &last, &prev );

    // Fix (or correct) the tree
    if( first && last )
        swap( &(first->data), &(last->data) );
    else if( first && middle ) // Adjacent nodes swapped
        swap( &(first->data), &(middle->data) );

    // else nodes have not been swapped, passed tree is really BST.
}
```

```
/* A utility function to print Inorder traversal */
void printInorder(struct node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    cout <<"  "<< node->data;
    printInorder(node->right);
}

/* Driver program to test above functions*/
int main()
{
    /*   6
        /  \
       10    2
      / \   / \
     1   3 7  12
     10 and 2 are swapped
    */

    struct node *root = newNode(6);
    root->left        = newNode(10);
    root->right       = newNode(2);
    root->left->left  = newNode(1);
    root->left->right = newNode(3);
    root->right->right = newNode(12);
    root->right->left = newNode(7);

    cout <<"Inorder Traversal of the original tree \n";
    printInorder(root);

    correctBST(root);

    cout <<"\nInorder Traversal of the fixed tree \n";
    printInorder(root);

    return 0;
}

// This code is contributed by shivanisinghss2110
```

C

```
// Two nodes in the BST's swapped, correct the BST.
#include <stdio.h>
#include <stdlib.h>
```

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

// A utility function to swap two integers
void swap( int* a, int* b )
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node *)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return(node);
}

// This function does inorder traversal to find out the two swapped nodes.
// It sets three pointers, first, middle and last.  If the swapped nodes are
// adjacent to each other, then first and middle contain the resultant nodes
// Else, first and last contain the resultant nodes
void correctBSTUtil( struct node* root, struct node** first,
                     struct node** middle, struct node** last,
                     struct node** prev )
{
    if( root )
    {
        // Recur for the left subtree
        correctBSTUtil( root->left, first, middle, last, prev );

        // If this node is smaller than the previous node, it's violating
        // the BST rule.
        if (*prev && root->data < (*prev)->data)
        {
            // If this is first violation, mark these two nodes as
            // 'first' and 'middle'
            if ( !*first )
            {
                *first = *prev;
```

```
                *middle = root;
            }

            // If this is second violation, mark this node as last
            else
                *last = root;
        }

        // Mark this node as previous
        *prev = root;

        // Recur for the right subtree
        correctBSTUtil( root->right, first, middle, last, prev );
    }
}

// A function to fix a given BST where two nodes are swapped.  This
// function uses correctBSTUtil() to find out two nodes and swaps the
// nodes to fix the BST
void correctBST( struct node* root )
{
    // Initialize pointers needed for correctBSTUtil()
    struct node *first, *middle, *last, *prev;
    first = middle = last = prev = NULL;

    // Set the pointers to find out two nodes
    correctBSTUtil( root, &first, &middle, &last, &prev );

    // Fix (or correct) the tree
    if( first && last )
        swap( &(first->data), &(last->data) );
    else if( first && middle ) // Adjacent nodes swapped
        swap( &(first->data), &(middle->data) );

    // else nodes have not been swapped, passed tree is really BST.
}

/* A utility function to print Inorder traversal */
void printInorder(struct node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);
}

/* Driver program to test above functions*/
int main()
{
```

```
    /*   6
        /  \
       10    2
      / \   / \
     1   3 7  12
     10 and 2 are swapped
    */

    struct node *root = newNode(6);
    root->left        = newNode(10);
    root->right       = newNode(2);
    root->left->left  = newNode(1);
    root->left->right = newNode(3);
    root->right->right = newNode(12);
    root->right->left = newNode(7);

    printf("Inorder Traversal of the original tree \n");
    printInorder(root);

    correctBST(root);

    printf("\nInorder Traversal of the fixed tree \n");
    printInorder(root);

    return 0;
}
```

Java

```
// Java program to correct the BST
// if two nodes are swapped
import java.util.*;
import java.lang.*;
import java.io.*;

class Node {

    int data;
    Node left, right;

    Node(int d) {
        data = d;
        left = right = null;
    }
}

class BinaryTree
{
    Node first, middle, last, prev;
```

```
// This function does inorder traversal
// to find out the two swapped nodes.
// It sets three pointers, first, middle
// and last. If the swapped nodes are
// adjacent to each other, then first
// and middle contain the resultant nodes
// Else, first and last contain the
// resultant nodes
void correctBSTUtil( Node root)
{
    if( root != null )
    {
        // Recur for the left subtree
        correctBSTUtil( root.left);

        // If this node is smaller than
        // the previous node, it's
        // violating the BST rule.
        if (prev != null && root.data <
                            prev.data)
        {
            // If this is first violation,
            // mark these two nodes as
            // 'first' and 'middle'
            if (first == null)
            {
                first = prev;
                middle = root;
            }

            // If this is second violation,
            // mark this node as last
            else
                last = root;
        }

        // Mark this node as previous
        prev = root;

        // Recur for the right subtree
        correctBSTUtil( root.right);
    }
}

// A function to fix a given BST where
// two nodes are swapped. This function
// uses correctBSTUtil() to find out
// two nodes and swaps the nodes to
// fix the BST
```

```
void correctBST( Node root )
{
    // Initialize pointers needed
    // for correctBSTUtil()
    first = middle = last = prev = null;

    // Set the pointers to find out
    // two nodes
    correctBSTUtil( root );

    // Fix (or correct) the tree
    if( first != null && last != null )
    {
        int temp = first.data;
        first.data = last.data;
        last.data = temp;
    }
    // Adjacent nodes swapped
    else if( first != null && middle !=
                                null )
    {
        int temp = first.data;
        first.data = middle.data;
        middle.data = temp;
    }

    // else nodes have not been swapped,
    // passed tree is really BST.
}

/* A utility function to print
 Inorder traversal */
void printInorder(Node node)
{
    if (node == null)
        return;
    printInorder(node.left);
    System.out.print(" " + node.data);
    printInorder(node.right);
}


// Driver program to test above functions
public static void main (String[] args)
{
    /*   6
        / \
       10  2
      / \ / \
     1  3 7 12
```

```
        10 and 2 are swapped
        */

        Node root = new Node(6);
        root.left = new Node(10);
        root.right = new Node(2);
        root.left.left = new Node(1);
        root.left.right = new Node(3);
        root.right.right = new Node(12);
        root.right.left = new Node(7);

        System.out.println("Inorder Traversal"+
                        " of the original tree");
        BinaryTree tree = new BinaryTree();
        tree.printInorder(root);

        tree.correctBST(root);

        System.out.println("\nInorder Traversal"+
                          " of the fixed tree");
        tree.printInorder(root);
    }
}
// This code is contributed by Chhavi
```

Python3

```
# Python3 program to correct the BST
# if two nodes are swapped
class Node:

    # Constructor to create a new node
    def __init__(self, data):

        self.key = data
        self.left = None
        self.right = None

# Utility function to track the nodes
# that we have to swap
def correctBstUtil(root, first, middle,
                   last, prev):

    if(root):

        # Recur for the left sub tree
        correctBstUtil(root.left, first,
```

```
                        middle, last, prev)

            # If this is the first violation, mark these
            # two nodes as 'first and 'middle'
            if(prev[0] and root.key < prev[0].key):
                if(not first[0]):
                    first[0] = prev[0]
                    middle[0] = root
                else:

                    # If this is the second violation,
                    # mark this node as last
                    last[0] = root

            prev[0] = root

            # Recur for the right subtree
            correctBstUtil(root.right, first,
                           middle, last, prev)

    # A function to fix a given BST where
    # two nodes are swapped. This function
    # uses correctBSTUtil() to find out two
    # nodes and swaps the nodes to fix the BST
    def correctBst(root):

        # Followed four lines just for forming
        # an array with only index 0 filled
        # with None and we will update it accordingly.
        # we made it null so that we can fill
        # node data in them.
        first = [None]
        middle = [None]
        last = [None]
        prev = [None]

        # Setting arrays (having zero index only)
        # for capturing the required node
        correctBstUtil(root, first, middle,
                       last, prev)

        # Fixing the two nodes
        if(first[0] and last[0]):

            # Swapping for first and last key values
            first[0].key, last[0].key = (last[0].key,
                                         first[0].key)

        elif(first[0] and middle[0]):
```

```
            # Swapping for first and middle key values
            first[0].key, middle[0].key = (middle[0].key,
                                           first[0].key)

        # else tree will be fine

    # Function to print inorder
    # traversal of tree
    def PrintInorder(root):

        if(root):
            PrintInorder(root.left)
            print(root.key, end = " ")
            PrintInorder(root.right)

        else:
            return

    # Driver code

    #      6
    #     /   \
    #   10    2
    #  / \   / \
    # 1   3 7   12

    # Following 7 lines are for tree formation
    root = Node(6)
    root.left = Node(10)
    root.right = Node(2)
    root.left.left = Node(1)
    root.left.right = Node(3)
    root.right.left = Node(7)
    root.right.right = Node(12)

    # Printing inorder traversal of normal tree
    print("inorder traversal of normal tree")
    PrintInorder(root)
    print("")

    # Function call to do the task
    correctBst(root)

    # Printing inorder for corrected Bst tree
    print("")
    print("inorder for corrected BST")

    PrintInorder(root)

    # This code is contributed by rajutkarshai
```

```
// C# program to correct the BST
// if two nodes are swapped
using System;
class Node{

public int data;
public Node left, right;
public Node(int d)
{
  data = d;
  left = right = null;
}
}

class BinaryTree
{
  Node first, middle,
       last, prev;

// This function does inorder traversal
// to find out the two swapped nodes.
// It sets three pointers, first, middle
// and last. If the swapped nodes are
// adjacent to each other, then first
// and middle contain the resultant nodes
// Else, first and last contain the
// resultant nodes
void correctBSTUtil( Node root)
{
  if( root != null )
  {
    // Recur for the
    // left subtree
    correctBSTUtil(root.left);

    // If this node is smaller than
    // the previous node, it's
    // violating the BST rule.
    if (prev != null && root.data <
        prev.data)
    {
      // If this is first violation,
      // mark these two nodes as
      // 'first' and 'middle'
      if (first == null)
```

```
      {
        first = prev;
        middle = root;
      }

      // If this is second violation,
      // mark this node as last
      else
        last = root;
    }

    // Mark this node
    // as previous
    prev = root;

    // Recur for the
    // right subtree
    correctBSTUtil(root.right);
  }
}

// A function to fix a given BST where
// two nodes are swapped. This function
// uses correctBSTUtil() to find out
// two nodes and swaps the nodes to
// fix the BST
void correctBST( Node root )
{
  // Initialize pointers needed
  // for correctBSTUtil()
  first = middle = last =
          prev = null;

  // Set the pointers to
  // find out two nodes
  correctBSTUtil(root);

  // Fix (or correct)
  // the tree
  if(first != null &&
     last != null)
  {
    int temp = first.data;
    first.data = last.data;
    last.data = temp;
  }

  // Adjacent nodes swapped
  else if(first != null &&
          middle != null)
```

```
    {
      int temp = first.data;
      first.data = middle.data;
      middle.data = temp;
    }

    // else nodes have not been
    // swapped, passed tree is
    // really BST.
  }

  // A utility function to print
  // Inorder traversal
  void printInorder(Node node)
  {
    if (node == null)
      return;
    printInorder(node.left);
    Console.Write(" " + node.data);
    printInorder(node.right);
  }

  // Driver code
  public static void Main(String[] args)
  {
    /*         6
              / \
            10   2
           / \ / \
          1  3 7 12

         10 and 2 are swapped
         */

    Node root = new Node(6);
    root.left = new Node(10);
    root.right = new Node(2);
    root.left.left = new Node(1);
    root.left.right = new Node(3);
    root.right.right = new Node(12);
    root.right.left = new Node(7);

    Console.WriteLine("Inorder Traversal" +
                      " of the original tree");
    BinaryTree tree = new BinaryTree();
    tree.printInorder(root);
    tree.correctBST(root);
    Console.WriteLine("\nInorder Traversal" +
                      " of the fixed tree");
    tree.printInorder(root);
```

```
}
}

// This code is contributed by gauravrajput1
```

Javascript

```
<script>

// JavaScript program to correct the BST
// if two nodes are swapped

class Node {
    constructor(val) {
        this.data = val;
        this.left = null;
        this.right = null;
    }
}


    var first, middle, last, prev;

    // This function does inorder traversal
    // to find out the two swapped nodes.
    // It sets three pointers, first, middle
    // and last. If the swapped nodes are
    // adjacent to each other, then first
    // and middle contain the resultant nodes
    // Else, first and last contain the
    // resultant nodes
    function correctBSTUtil(root) {
        if (root != null) {
            // Recur for the left subtree
            correctBSTUtil(root.left);

            // If this node is smaller than
            // the previous node, it's
            // violating the BST rule.
            if (prev != null && root.data < prev.data) {
                // If this is first violation,
                // mark these two nodes as
                // 'first' and 'middle'
                if (first == null) {
                    first = prev;
                    middle = root;
                }

                // If this is second violation,
```

```
            // mark this node as last
            else
                last = root;
        }

        // Mark this node as previous
        prev = root;

        // Recur for the right subtree
        correctBSTUtil(root.right);
    }
}

// A function to fix a given BST where
// two nodes are swapped. This function
// uses correctBSTUtil() to find out
// two nodes and swaps the nodes to
// fix the BST
function correctBST(root) {
    // Initialize pointers needed
    // for correctBSTUtil()
    first = middle = last = prev = null;

    // Set the pointers to find out
    // two nodes
    correctBSTUtil(root);

    // Fix (or correct) the tree
    if (first != null && last != null) {
        var temp = first.data;
        first.data = last.data;
        last.data = temp;
    }
    // Adjacent nodes swapped
    else if (first != null && middle != null) {
        var temp = first.data;
        first.data = middle.data;
        middle.data = temp;
    }

    // else nodes have not been swapped,
    // passed tree is really BST.
}

/*
 * A utility function to print Inorder traversal
 */
function printInorder(node) {
    if (node == null)
        return;
```

```
        printInorder(node.left);
        document.write(" " + node.data);
        printInorder(node.right);
    }

    // Driver program to test above functions

        /*
         * 6 / \ 10 2 / \ / \ 1 3 7 12
         *
         * 10 and 2 are swapped
         */

var root = new Node(6);
        root.left = new Node(10);
        root.right = new Node(2);
        root.left.left = new Node(1);
        root.left.right = new Node(3);
        root.right.right = new Node(12);
        root.right.left = new Node(7);

        document.write("Inorder Traversal" +
        " of the original tree<br/>");
        printInorder(root);

        correctBST(root);

        document.write("<br/>Inorder Traversal" +
        " of the fixed tree<br/>");
        printInorder(root);

// This code contributed by aashish1995

</script>
```

Output

```
 Inorder Traversal of the original tree
   1   10   3   6   7   2   12
 Inorder Traversal of the fixed tree
   1   2   3   6   7   10   12
```

Time Complexity: O(n)

Auxiliary Space: O(n) for call stack

Recommended Problem

Fixing Two nodes of a BST

Binary Search Tree Amazon BankBazaar +3 more

Solve Problem

Submission count: 34K

Like 81

Previous

Print Common Nodes in Two Binary Search Trees

Next

Count inversions in an array | Set 2 (Using Self-Balancing BST)

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Two nodes of a BST are swapped, correct the BST | Set-2**
12, Jun 19

02 **Maximum element between two nodes of BST**
22, Jan 17

03 **Shortest distance between two nodes in BST**

05 **K'th Largest Element in BST when modification to BST is not allowed**
19, Mar 15

06 **Find k-th smallest element in BST (Order Statistics in BST)**
15, Feb 11

07 **Sum of all nodes with smaller values at a distance K from a given node in a BST**

03, Oct 17

04 **Convert a normal BST to Balanced BST**
01, May 16

10, Aug 21

08 **Implementing a BST where every node stores the maximum number of nodes in the path till any leaf**
14, Aug 19

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : rajutkarshai, GauravRajput1, aashish1995, gabaa406, sumitgumber28, shivanisinghss2110, adnanirshad158, saurabh1990aror, arorakashish0911, polymatir3j, hardikkoriintern

Article Tags : Amazon, BankBazaar, FactSet, Microsoft, Binary Search Tree

Practice Tags : Amazon, BankBazaar, FactSet, Microsoft, Binary Search Tree

Improve Article | Report Issue

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

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Learn

Algorithms

Data Structures

SDE Cheat Sheet

Machine learning

CS Subjects

Video Tutorials

Courses

News

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Knowledge

Languages

Python

Java

CPP

Golang

C#

SQL

Kotlin

Web Development

Web Tutorials

Django Tutorial

HTML

JavaScript

Bootstrap

ReactJS

NodeJS

Contribute

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship

@geeksforgeeks , Some rights reserved

Do Not Sell My Personal Information