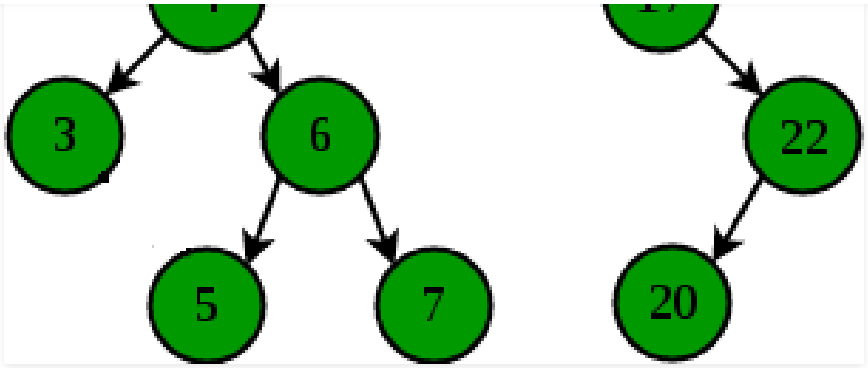
Find the closest element in Binary Search Tree | Space Efficient Method

Difficulty Level : Medium • Last Updated : 31 Mar, 2022

Given a binary search tree and a target node K. The task is to find the node with the minimum absolute difference with given target value K.

NOTE: The approach used should have constant extra space consumed O(1). No recursion or stack/queue like containers should be used.



Examples:

```
Input:  k = 4
Output:  4

Input:  k = 18
Output:  17
```

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Morris traversal is based on Threaded Binary trees which makes use of NU
to some successor or predecessor nodes. As in a binary tree with n nodes,
In the algorithm mentioned below we simply do inorder tree traversal and
Morris Traversal we check for differences between the node's data and the key and maintain two variables 'diff' and 'closest' which are updated when we find a closer node to the key. When we are done with the complete inorder tree traversal we have the closest node.

Algorithm :

```
1) Initialize Current as root.

2) Initialize a variable diff as INT_MAX.

3)initialize a variable closest(pointer to node) which
  will be returned.

4) While current is not NULL:

  4.1) If the current has no left child:
     a) If the absolute difference between current's data
        and the key is smaller than diff:
```

b)otherwise, Move to the right child of current.

4.2) Else, here we have 2 cases:

a) Find the inorder predecessor of the current node.
Inorder predecessor is the rightmost node
in the left subtree or left child itself.

b) If the right child of the inorder predecessor is NULL:
1) Set current as the right child of its inorder
predecessor(Making threads between nodes).
2) Move current node to its left child.

c) Else, if the threaded link between the current node
and it's inorder predecessor already exists :

1) Set right pointer of the inorder predecessor node as NULL.

2) If the absolute difference between current's data and
the key is smaller than diff:
a) Set diff variable as the absolute difference between
the current node and the key.

3)By the time we have traversed the whole tree, we have the closest node, so we simply return closest.

Below is the implementation of above approach:

C++

```
// CPP program to find closest value in
// a Binary Search Tree.
#include <iostream>
#include <limits.h>
using namespace std;

// Tree Node
struct Node {
    int data;
    Node *left, *right;
};

// Utility function to create a new Node
Node* newNode(int data)
{
    Node* temp = new Node();
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
```

```
{
    int diff = INT_MAX;
    Node* curr = root;
    Node* closest;

    while (curr) {
        if (curr->left == NULL) {

            // updating diff if the current diff is
            // smaller than prev difference
            if (diff > abs(curr->data - key)) {
                diff = abs(curr->data - key);
                closest = curr;
            }

            curr = curr->right;
        }

        else {

            // finding the inorder predecessor
            Node* pre = curr->left;
            while (pre->right != NULL &&
                   pre->right != curr)
                pre = pre->right;

            if (pre->right == NULL) {
                pre->right = curr;
                curr = curr->left;
            }
```

```
                // if a closer Node found, then update
                // the diff and set closest to current
                if (diff > abs(curr->data - key)) {
                    diff = abs(curr->data - key);
                    closest = curr;
                }

                // moving to the right child
                curr = curr->right;
            }
        }
    }

    return closest;
}

// Driver Code
int main()
{
    /* Constructed binary tree is
          5
        /   \
       3     9
     /  \   /  \
    1    2  8    12 */
    Node* root = newNode(5);
    root->left = newNode(3);
    root->right = newNode(9);
    root->left->left = newNode(1);
    root->left->right = newNode(2);
```

```
}
```

Java

```
// Java program to find closest value in
// a Binary Search Tree.
class GFG
{

// Tree Node
static class Node
{
    int data;
    Node left, right;
};

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

// Function to find the Node closest to the
// given key in BST using Morris Traversal
```

```
while (curr != null)
{
    if (curr.left == null)
    {

        // updating diff if the current diff is
        // smaller than prev difference
        if (diff > Math.abs(curr.data - key))
        {
            diff = Math.abs(curr.data - key);
            closest = curr;
        }

        curr = curr.right;
    }

    else
    {

        // finding the inorder predecessor
        Node pre = curr.left;
        while (pre.right != null &&
            pre.right != curr)
            pre = pre.right;

        if (pre.right == null)
        {
            pre.right = curr;
            curr = curr.left;
```

```
                pre.right = null;

                // if a closer Node found, then update
                // the diff and set closest to current
                if (diff > Math.abs(curr.data - key))
                {
                    diff = Math.abs(curr.data - key);
                    closest = curr;
                }

                // moving to the right child
                curr = curr.right;
            }
        }
    }

    return closest;
}

// Driver Code
public static void main(String[] args)
{
    /* Constructed binary tree is
        5
        / \
    3     9
    / \ / \
    1 2 8 12 */
    Node root = newNode(5);
    root.left = newNode(3);
```

```
        System.out.println(closestNodeUsingMorrisTraversal(root, 10
    }
    }

    // This code is contributed by Rajput-Ji
```

Python3

```
# Python program to find closest value in
# Binary search Tree

_MIN = -2147483648
_MAX = 2147483648

# Helper function that allocates a new
# node with the given data and None left
# and right pointers.
class newNode:

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

# Function to find the Node closest to the
# given key in BST using Morris Traversal
def closestNodeUsingMorrisTraversal(root,key):
```

```
    # updating diff if the current diff is
    # smaller than prev difference
    if (diff > abs(curr.data - key)) :
        diff = abs(curr.data - key)
        closest = curr

    curr = curr.right


else :

    # finding the inorder predecessor
    pre = curr.left
    while (pre.right != None and
            pre.right != curr):
        pre = pre.right

    if (pre.right == None):
        pre.right = curr
        curr = curr.left


    # threaded link between curr and
    # its predecessor already exists
    else :
        pre.right = None

        # if a closer Node found, then update
        # the diff and set closest to current
```

```
    return closest


# Driver Code
if __name__ == '__main__':
    """ /* Constructed binary tree is
        5
        / \
    3 9
    / \ / \
    1 2 8 12 */ """

    root = newNode(5)
    root.left = newNode(3)
    root.right = newNode(9)
    root.left.right = newNode(2)
    root.left.left = newNode(1)
    root.right.right = newNode(12)
    root.right.left = newNode(8)
    print(closestNodeUsingMorrisTraversal(root, 10).data)

# This code is contributed
# Shubham Singh(SHUBHAMSINGH10)
```

C#

```
// C# program to find closest value in
```

```
// Tree Node
public class Node
{
    public int data;
    public Node left, right;
};

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

// Function to find the Node closest to the
// given key in BST using Morris Traversal
static Node closestNodeUsingMorrisTraversal(Node root,
                                            int key)
{
    int diff = int.MaxValue;
    Node curr = root;
    Node closest = null;

    while (curr != null)
    {
        if (curr.left == null)
        {
```

```
            closest = curr;
        }

        curr = curr.right;
    }

    else
    {

        // finding the inorder predecessor
        Node pre = curr.left;
        while (pre.right != null &&
            pre.right != curr)
            pre = pre.right;

        if (pre.right == null)
        {
            pre.right = curr;
            curr = curr.left;
        }

        // threaded link between curr and
        // its predecessor already exists
        else
        {
            pre.right = null;

            // if a closer Node found, then update
            // the diff and set closest to current
            if (diff > Math.Abs(curr.data - key))
```

```
                curr = curr.right;
            }
        }
    }

    return closest;
}

// Driver Code
public static void Main(String[] args)
{
    /* Constructed binary tree is
        5
        / \
    3     9
    / \ / \
    1 2 8 12 */
    Node root = newNode(5);
    root.left = newNode(3);
    root.right = newNode(9);
    root.left.left = newNode(1);
    root.left.right = newNode(2);
    root.right.left = newNode(8);
    root.right.right = newNode(12);

    Console.WriteLine(closestNodeUsingMorrisTraversal(root, 10).data);
}
}

/* This code is contributed by PrinciRaj1992 */
```

```
// Javascript program to find closest value in
// a Binary Search Tree.

// Tree Node
class Node
{
    constructor()
    {
        this.data = 0;
        this.left = null;
        this.right = null;
    }
};

// Utility function to create a new Node
function newNode(data)
{
    var temp = new Node();
    temp.data = data;
    temp.left = temp.right = null;
    return temp;
}

// Function to find the Node closest to the
// given key in BST using Morris Traversal
function closestNodeUsingMorrisTraversal(root, key)
{
    var diff = 1000000000;
    var curr = root;
    var closest = null;
```

```
        // updating diff if the current diff is
        // smaller than prev difference
        if (diff > Math.abs(curr.data - key))
        {
            diff = Math.abs(curr.data - key);
            closest = curr;
        }

        curr = curr.right;
    }

    else
    {

        // finding the inorder predecessor
        var pre = curr.left;
        while (pre.right != null &&
            pre.right != curr)
            pre = pre.right;

        if (pre.right == null)
        {
            pre.right = curr;
            curr = curr.left;
        }

        // threaded link between curr and
        // its predecessor already exists
        else
        {
```

```
                    diff = Math.abs(curr.data - key);
                    closest = curr;
                }

                // moving to the right child
                curr = curr.right;
            }
        }
    }

    return closest;
}

// Driver Code
/* Constructed binary tree is
    5
    / \
3     9
/ \ / \
1 2 8 12 */
var root = newNode(5);
root.left = newNode(3);
root.right = newNode(9);
root.left.left = newNode(1);
root.left.right = newNode(2);
root.right.left = newNode(8);
root.right.right = newNode(12);
document.write(closestNodeUsingMorrisTraversal(root, 10).data);

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

9

Time Complexity: O(n)
Auxiliary Space : O(1)

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Article Tags : morris-traversal, tree-traversal, Binary Search Tree

Practice Tags : Binary Search Tree

Improve Article | Report Issue

GeeksforGeeks
5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 2(

feedback@geeksforgeeks.org