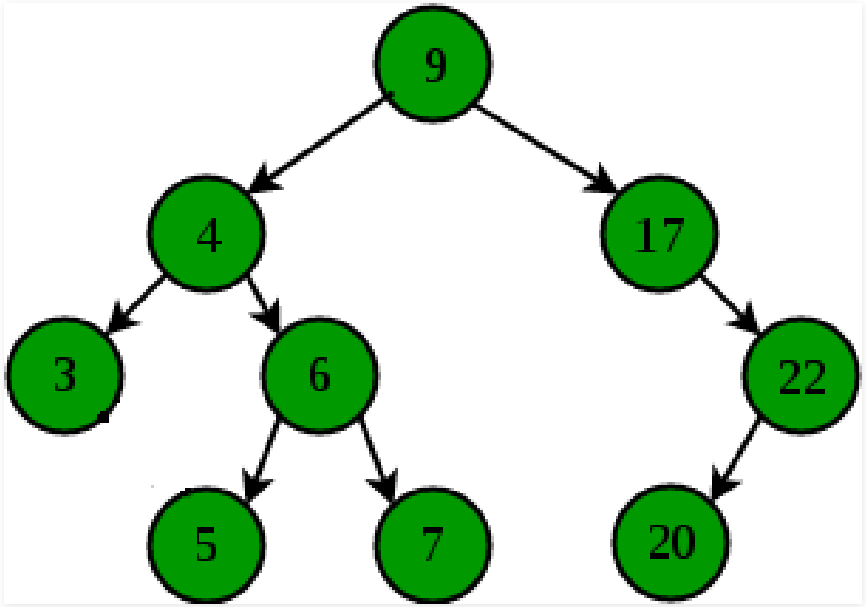
Find the closest element in Binary Search Tree

Difficulty Level : Medium ● Last Updated : 28 Jun, 2021

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



Examples:

```
Input  :  k = 18
Output :  17

Input  :  k = 12
Output :  9
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

A **simple solution** for this problem is to store Inorder traversal of given binary search tree in an auxiliary array and then by taking absolute difference of each element find the node having minimum absolute difference with given target value K in linear time.

An **efficient solution** for this problem is to take advantage of characteristics of BST. Here is the algorithm to solve this problem :

- If target value K is present in given BST, then it's the node having minimum absolute difference.
- If target value K is less than the value of current node then move to the left child.
- If target value K is greater than the value of current node then move to the right child.

C++ Java Python3 C#

```
#include<bits/stdc++.h>
using namespace std;

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

/* Utility that allocates a new node with the
  given key and NULL left and right pointers. */
struct Node* newnode(int key)
{
    struct Node* node = new (struct Node);
    node->key = key;
    node->left = node->right  = NULL;
    return (node);
}

// Function to find node with minimum absolute
// difference with given K
// min_diff   --> minimum difference till now
// min_diff_key  --> node having minimum absolute
//                   difference with K
void maxDiffUtil(struct Node *ptr, int k, int &min_diff,
                                          int &min_diff_key)
{
    if (ptr == NULL)
```

```
            return;
        }

        // update min_diff and min_diff_key by checking
        // current node value
        if (min_diff > abs(ptr->key - k))
        {
            min_diff = abs(ptr->key - k);
            min_diff_key = ptr->key;
        }

        // if k is less than ptr->key then move in
        // left subtree else in right subtree
        if (k < ptr->key)
            maxDiffUtil(ptr->left, k, min_diff, min_diff_key);
        else
            maxDiffUtil(ptr->right, k, min_diff, min_diff_key);
    }

    // Wrapper over maxDiffUtil()
    int maxDiff(Node *root, int k)
    {
        // Initialize minimum difference
        int min_diff = INT_MAX, min_diff_key = -1;

        // Find value of min_diff_key (Closest key
        // in tree with k)
        maxDiffUtil(root, k, min_diff, min_diff_key);

        return min_diff_key;
```

```
    root->left    = newnode(4);
    root->right   = newnode(17);
    root->left->left = newnode(3);
    root->left->right = newnode(6);
    root->left->right->left = newnode(5);
    root->left->right->right = newnode(7);
    root->right->right = newnode(22);
    root->right->right->left = newnode(20);
    int k = 18;
    cout << maxDiff(root, k);
    return 0;
}
```

Java

```
// Recursive Java program to find key closest to k
// in given Binary Search Tree.

 class solution
 {

     static int min_diff, min_diff_key;

/*  A binary tree node has key, pointer to left child
and a pointer to right child */
static class Node
{
    int key;
```

```
 static Node  newnode(int key)
{

     Node  node = new Node();
    node.key = key;
    node.left = node.right  = null;
    return (node);
}

// Function to find node with minimum absolute
// difference with given K
// min_diff   -. minimum difference till now
// min_diff_key  -. node having minimum absolute
//                  difference with K
static void maxDiffUtil(Node  ptr, int k)
{
    if (ptr == null)
        return ;

    // If k itself is present
    if (ptr.key == k)
    {
        min_diff_key = k;
        return;
    }

    // update min_diff and min_diff_key by checking
    // current node value
    if (min_diff > Math.abs(ptr.key - k))
```

```
    // left subtree else in right subtree
    if (k < ptr.key)
        maxDiffUtil(ptr.left, k);
    else
        maxDiffUtil(ptr.right, k);
}

// Wrapper over maxDiffUtil()
static int maxDiff(Node  root, int k)
{
    // Initialize minimum difference
    min_diff = 999999999; min_diff_key = -1;

    // Find value of min_diff_key (Closest key
    // in tree with k)
    maxDiffUtil(root, k);

    return min_diff_key;
}

// Driver program to run the case
public static void main(String args[])
{

     Node  root = newnode(9);
    root.left    = newnode(4);
    root.right   = newnode(17);
    root.left.left = newnode(3);
    root.left.right = newnode(6);
    root.left.right.left = newnode(5);
```

```
}
}
//contributed by Arnab Kundu
```

Python3

```
# Recursive Python program to find key
# closest to k in given Binary Search Tree.

# Utility that allocates a new node with the
# given key and NULL left and right pointers.
class newnode:

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

# Function to find node with minimum
# absolute difference with given K
# min_diff --> minimum difference till now
# min_diff_key --> node having minimum absolute
#                  difference with K
def maxDiffUtil(ptr, k, min_diff, min_diff_key):
    if ptr == None:
        return
```

```
    # checking current node value
    if min_diff > abs(ptr.key - k):
        min_diff = abs(ptr.key - k)
        min_diff_key[0] = ptr.key

    # if k is less than ptr->key then move
    # in left subtree else in right subtree
    if k < ptr.key:
        maxDiffUtil(ptr.left, k, min_diff,
                                 min_diff_key)
    else:
        maxDiffUtil(ptr.right, k, min_diff,
                                  min_diff_key)

# Wrapper over maxDiffUtil()
def maxDiff(root, k):

    # Initialize minimum difference
    min_diff, min_diff_key = 999999999999, [-1]

    # Find value of min_diff_key (Closest
    # key in tree with k)
    maxDiffUtil(root, k, min_diff, min_diff_key)

    return min_diff_key[0]

# Driver Code
if __name__ == '__main__':
    root = newnode(9)
    root.left = newnode(4)
```

```
    root.right.right.left = newnode(20)
    k = 18
    print(maxDiff(root, k))

# This code is contributed by PranchalK
```

C#

```
using System;

// Recursive C# program to find key closest to k
// in given Binary Search Tree.

 public class solution
 {

     public static int min_diff, min_diff_key;

/*  A binary tree node has key, pointer to left child
and a pointer to right child */
public class Node
{
    public int key;

     public Node left, right;
}

/*  Utility that allocates a new node with the
```

```
    node.key = key;
    node.left = node.right = null;
    return (node);
 }

// Function to find node with minimum absolute
// difference with given K
// min_diff   -. minimum difference till now
// min_diff_key  -. node having minimum absolute
//                  difference with K
public static void maxDiffUtil(Node ptr, int k)
{
    if (ptr == null)
    {
        return;
    }

    // If k itself is present
    if (ptr.key == k)
    {
        min_diff_key = k;
        return;
    }

    // update min_diff and min_diff_key by checking
    // current node value
    if (min_diff > Math.Abs(ptr.key - k))
    {
        min_diff = Math.Abs(ptr.key - k);
        min_diff_key = ptr.key;
```

```
            maxDiffUtil(ptr.left, k);
        }
        else
        {
            maxDiffUtil(ptr.right, k);
        }
    }

    // Wrapper over maxDiffUtil()
    public static int maxDiff(Node root, int k)
    {
        // Initialize minimum difference
        min_diff = 999999999;
        min_diff_key = -1;

        // Find value of min_diff_key (Closest key
        // in tree with k)
        maxDiffUtil(root, k);

        return min_diff_key;
    }

    // Driver program to run the case
    public static void Main(string[] args)
    {

         Node root = newnode(9);
        root.left = newnode(4);
        root.right = newnode(17);
        root.left.left = newnode(3);
```

```
        Console.WriteLine(maxDiff(root, k));

    }
     }

      // This code is contributed by Shrikant13
```

Output:

```
17
```

Time complexity : O(h) where h is height of given Binary Search Tree.

Reference :
http://stackoverflow.com/questions/6209325/how-to-find-the-closest-element-to-a-given-key-value-in-a-binary-search-tree

This article is contributed by Shashank Mishra (Gullu). If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Previous

Find pairs with given sum such that pair elements lie in different BSTs

Next

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

| Space Efficient Method
18, Jun 18

02 Complexity of different operations in Binary tree, Binary Search Tree and AVL tree
19, Jan 18

03 Find the closest leaf in a Binary Tree
21, Dec 14

04 Difference between Binary Tree and Binary Search Tree
31, Oct 19

using STL set
22, Mar 18

06 Binary Tree to Binary Search Tree Conversion
15, Jun 12

07 Minimum swap required to convert binary tree to binary search tree
28, Jan 17

08 Binary Search Tree | Set 1 (Search and Insertion)
30, Jan 14

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

GeeksforGeeks
5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org