GeeksforGeeks

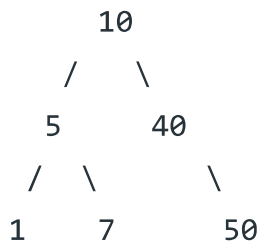
Related Articles

Construct BST from given preorder traversal | Set 1

Difficulty Level : Hard • Last Updated : 05 May, 2021

Given preorder traversal of a binary search tree, construct the BST.

For example, if the given traversal is {10, 5, 1, 7, 40, 50}, then the output should be the root of the following tree.



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

Method 1 (O(n^2) time complexity)

The first element of preorder traversal is always root. We first construct the root. Then we find the index of the first element which is greater than the root. Let the index be 'i'. The values between root and 'i' will be part of the left subtree, and the values between 'i+1' and 'n-1' will be part of the right subtree. Divide given pre[] at index "i" and recur for left and right sub-trees.

For example in {10, 5, 1, 7, 40, 50}, 10 is the first element, so we make it root. Now we look for the first element greater than 10, we find 40. So we know the structure of BST is as following.

```
            /      \
   {5, 1, 7}       {40, 50}
```

We recursively follow above steps for subarrays {5, 1, 7} and {40, 50}, and get the complete tree.

C++

```
/* A O(n^2) program for construction of BST from preorder
 * traversal */
#include <bits/stdc++.h>
using namespace std;

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

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[].
// preIndex is used to keep track of index in pre[].
node* constructTreeUtil(int pre[], int* preIndex, int low,
                        int high, int size)
{
    // Base case
    if (*preIndex >= size || low > high)
        return NULL;
```

```
    node* root = newNode(pre[*preIndex]);
    *preIndex = *preIndex + 1;

    // If the current subarry has only one element, no need
    // to recur
    if (low == high)
        return root;

    // Search for the first element greater than root
    int i;
    for (i = low; i <= high; ++i)
        if (pre[i] > root->data)
            break;

    // Use the index of element found in preorder to divide
    // preorder array in two parts. Left subtree and right
    // subtree
    root->left = constructTreeUtil(pre, preIndex, *preIndex,
                                   i - 1, size);
    root->right
        = constructTreeUtil(pre, preIndex, i, high, size);

    return root;
}

// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, 0, size - 1,
                             size);
}

// A utility function to print inorder traversal of a Binary
// Tree
void printInorder(node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}

// Driver code
int main()
```

```
    node* root = constructTree(pre, size);

    cout << "Inorder traversal of the constructed tree: \n";
    printInorder(root);

    return 0;
}

// This code is contributed by rathbhupendra
```

C

```
/* A O(n^2) program for construction of BST from preorder
 * traversal */
#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;
    struct node* right;
};

// A utility function to create a node
struct node* newNode(int data)
{
    struct node* temp
        = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[].
// preIndex is used to keep track of index in pre[].
struct node* constructTreeUtil(int pre[], int* preIndex,
                               int low, int high, int size)
{
    // Base case
    if (*preIndex >= size || low > high)
        return NULL;
```

```
    struct node* root = newNode(pre[*preIndex]);
    *preIndex = *preIndex + 1;

    // If the current subarry has only one element, no need
    // to recur
    if (low == high)
        return root;

    // Search for the first element greater than root
    int i;
    for (i = low; i <= high; ++i)
        if (pre[i] > root->data)
            break;

    // Use the index of element found in preorder to divide
    // preorder array in two parts. Left subtree and right
    // subtree
    root->left = constructTreeUtil(pre, preIndex, *preIndex,
                                    i - 1, size);
    root->right
        = constructTreeUtil(pre, preIndex, i, high, size);

    return root;
}

// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
struct node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, 0, size - 1,
                             size);
}

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

// Driver code
int main()
```

```
    struct node* root = constructTree(pre, size);

    printf("Inorder traversal of the constructed tree: \n");
    printInorder(root);

    return 0;
}
```

Java

```
// Java program to construct BST from given preorder
// traversal

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class Index {

    int index = 0;
}

class BinaryTree {

    Index index = new Index();

    // A recursive function to construct Full from pre[].
    // preIndex is used to keep track of index in pre[].
    Node constructTreeUtil(int pre[], Index preIndex,
                           int low, int high, int size)
    {

        // Base case
        if (preIndex.index >= size || low > high) {
            return null;
        }
```

```
        Node root = new Node(pre[preIndex.index]);
        preIndex.index = preIndex.index + 1;

        // If the current subarry has only one element, no
        // need to recur
        if (low == high) {
            return root;
        }

        // Search for the first element greater than root
        int i;
        for (i = low; i <= high; ++i) {
            if (pre[i] > root.data) {
                break;
            }
        }

        // Use the index of element found in preorder to
        // divide preorder array in two parts. Left subtree
        // and right subtree
        root.left = constructTreeUtil(
            pre, preIndex, preIndex.index, i - 1, size);
        root.right = constructTreeUtil(pre, preIndex, i,
                                       high, size);

        return root;
    }

    // The main function to construct BST from given
    // preorder traversal. This function mainly uses
    // constructTreeUtil()
    Node constructTree(int pre[], int size)
    {
        return constructTreeUtil(pre, index, 0, size - 1,
                                 size);
    }

    // A utility function to print inorder traversal of a
    // Binary Tree
    void printInorder(Node node)
    {
        if (node == null) {
            return;
        }
        printInorder(node.left);
        System.out.print(node.data + " ");
        printInorder(node.right);
```

```
    {
        BinaryTree tree = new BinaryTree();
        int pre[] = new int[] { 10, 5, 1, 7, 40, 50 };
        int size = pre.length;
        Node root = tree.constructTree(pre, size);
        System.out.println(
            "Inorder traversal of the constructed tree is ");
        tree.printInorder(root);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# A O(n^2) Python3 program for
# construction of BST from preorder traversal

# A binary tree node


class Node():

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


# constructTreeUtil.preIndex is a static variable of
# function constructTreeUtil

# Function to get the value of static variable
# constructTreeUtil.preIndex
def getPreIndex():
    return constructTreeUtil.preIndex

# Function to increment the value of static variable
# constructTreeUtil.preIndex


def incrementPreIndex():
    constructTreeUtil.preIndex += 1
```

```
def constructTreeUtil(pre, low, high):

        # Base Case
    if(low > high):
        return None

    # The first node in preorder traversal is root. So take
    # the node at preIndex from pre[] and make it root,
    # and increment preIndex
    root = Node(pre[getPreIndex()])
    incrementPreIndex()

    # If the current subarray has onlye one element,
    # no need to recur
    if low == high:
        return root

    r_root = -1

    # Search for the first element greater than root
    for i in range(low, high+1):
        if (pre[i] > root.data):
            r_root = i
            break

    # If no elements are greater than the current root,
    # all elements are left children
    # so assign root appropriately
    if r_root == -1:
        r_root = getPreIndex() + (high - low)

    # Use the index of element found in preorder to divide
    # preorder array in two parts. Left subtree and right
    # subtree
    root.left = constructTreeUtil(pre, getPreIndex(), r_root-1)

    root.right = constructTreeUtil(pre, r_root, high)

    return root

# The main function to construct BST from given preorder
# traversal. This function mailny uses constructTreeUtil()


def constructTree(pre):
    size = len(pre)
    constructTreeUtil.preIndex = 0
```

```
    if root is None:
        return
    printInorder(root.left)
    print root.data,
    printInorder(root.right)


# Driver code
pre = [10, 5, 1, 7, 40, 50]

root = constructTree(pre)
print "Inorder traversal of the constructed tree:"
printInorder(root)

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

C#

```
using System;

// C# program to construct BST from given preorder traversal
// A binary tree node
public class Node {

    public int data;
    public Node left, right;

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

public class Index {

    public int index = 0;
}

public class BinaryTree {

    public Index index = new Index();

    // A recursive function to construct Full from pre[].
    // preIndex is used to keep track of index in pre[].
```

```
{

    // Base case
    if (preIndex.index >= size || low > high) {
        return null;
    }

    // The first node in preorder traversal is root. So
    // take the node at preIndex from pre[] and make it
    // root, and increment preIndex
    Node root = new Node(pre[preIndex.index]);
    preIndex.index = preIndex.index + 1;

    // If the current subarry has only one element, no
    // need to recur
    if (low == high) {
        return root;
    }

    // Search for the first element greater than root
    int i;
    for (i = low; i <= high; ++i) {
        if (pre[i] > root.data) {
            break;
        }
    }

    // Use the index of element found in preorder to
    // divide preorder array in two parts. Left subtree
    // and right subtree
    root.left = constructTreeUtil(
        pre, preIndex, preIndex.index, i - 1, size);
    root.right = constructTreeUtil(pre, preIndex, i,
                                   high, size);

    return root;
}

// The main function to construct BST from given
// preorder traversal. This function mainly uses
// constructTreeUtil()
public virtual Node constructTree(int[] pre, int size)
{
    return constructTreeUtil(pre, index, 0, size - 1,
                             size);
}
```

```
        if (node == null) {
            return;
        }
        printInorder(node.left);
        Console.Write(node.data + " ");
        printInorder(node.right);
    }

    // Driver code
    public static void Main(string[] args)
    {
        BinaryTree tree = new BinaryTree();
        int[] pre = new int[] { 10, 5, 1, 7, 40, 50 };
        int size = pre.Length;
        Node root = tree.constructTree(pre, size);
        Console.WriteLine(
            "Inorder traversal of the constructed tree is ");
        tree.printInorder(root);
    }
}

// This code is contributed by Shrikant13
```

Output

```
Inorder traversal of the constructed tree:
1 5 7 10 40 50
```

Time Complexity: $O(n^2)$

Method 2 (O(n) time complexity)

The idea used here is inspired by method 3 of this post. The trick is to set a range {min .. max} for every node. Initialize the range as {INT_MIN .. INT_MAX}. The first node will definitely be in range, so create a root node. To construct the left subtree, set the range as {INT_MIN ...root->data}. If a value is in the range {INT_MIN .. root->data}, the values are part of the left subtree. To construct the right subtree, set the range as {root->data..max .. INT_MAX}.

Below is the implementation of the above idea:

C++

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

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

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct
// BST from pre[]. preIndex is used
// to keep track of index in pre[].
node* constructTreeUtil(int pre[], int* preIndex, int key,
                        int min, int max, int size)
{
    // Base case
    if (*preIndex >= size)
        return NULL;

    node* root = NULL;

    // If current element of pre[] is in range, then
    // only it is part of current subtree
    if (key > min && key < max) {
        // Allocate memory for root of this
        // subtree and increment *preIndex
        root = newNode(key);
        *preIndex = *preIndex + 1;

        if (*preIndex < size) {
            // Construct the subtree under root
            // All nodes which are in range
            // {min .. key} will go in left
```

```
                                            min, key, size);
        }
        if (*preIndex < size) {
            // All nodes which are in range
            // {key..max} will go in right
            // subtree, and first such node
            // will be root of right subtree.
            root->right = constructTreeUtil(pre, preIndex,
                                            pre[*preIndex],
                                            key, max, size);
        }
    }

    return root;
}

// The main function to construct BST
// from given preorder traversal.
// This function mainly uses constructTreeUtil()
node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, pre[0],
                             INT_MIN, INT_MAX, size);
}

// A utility function to print inorder
// traversal of a Binary Tree
void printInorder(node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}

// Driver code
int main()
{
    int pre[] = { 10, 5, 1, 7, 40, 50 };
    int size = sizeof(pre) / sizeof(pre[0]);

    // Function call
    node* root = constructTree(pre, size);

    cout << "Inorder traversal of the constructed tree: \n";
```

```
// This is code is contributed by rathbhupendra
```

C

```
/* A O(n) program for construction of BST from preorder
 * traversal */
#include <limits.h>
#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;
    struct node* right;
};

// A utility function to create a node
struct node* newNode(int data)
{
    struct node* temp
        = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct BST from pre[].
// preIndex is used to keep track of index in pre[].
struct node* constructTreeUtil(int pre[], int* preIndex,
                               int key, int min, int max,
                               int size)
{
    // Base case
    if (*preIndex >= size)
        return NULL;

    struct node* root = NULL;

    // If current element of pre[] is in range, then
    // only it is part of current subtree
```

```
        *preIndex = *preIndex + 1;

        if (*preIndex < size) {
            // Construct the subtree under root
            // All nodes which are in range {min .. key}
            // will go in left subtree, and first such node
            // will be root of left subtree.
            root->left = constructTreeUtil(pre, preIndex,
                                           pre[*preIndex],
                                           min, key, size);
        }
        if (*preIndex < size) {
            // All nodes which are in range {key..max} will
            // go in right subtree, and first such node will
            // be root of right subtree.
            root->right = constructTreeUtil(pre, preIndex,
                                            pre[*preIndex],
                                            key, max, size);
        }
    }

    return root;
}

// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
struct node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, pre[0],
                             INT_MIN, INT_MAX, size);
}

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

// Driver code
int main()
{
```

```
    struct node* root = constructTree(pre, size);

    printf("Inorder traversal of the constructed tree: \n");
    printInorder(root);

    return 0;
}
```

Java

```
// Java program to construct BST from given preorder
// traversal

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class Index {

    int index = 0;
}

class BinaryTree {

    Index index = new Index();

    // A recursive function to construct BST from pre[].
    // preIndex is used to keep track of index in pre[].
    Node constructTreeUtil(int pre[], Index preIndex,
                           int key, int min, int max,
                           int size)
    {

        // Base case
        if (preIndex.index >= size) {
            return null;
```

```
        // If current element of pre[] is in range, then
        // only it is part of current subtree
        if (key > min && key < max) {

            // Allocate memory for root of this
            // subtree and increment *preIndex
            root = new Node(key);
            preIndex.index = preIndex.index + 1;

            if (preIndex.index < size) {

                // Construct the subtree under root
                // All nodes which are in range {min .. key}
                // will go in left subtree, and first such
                // node will be root of left subtree.
                root.left = constructTreeUtil(
                    pre, preIndex, pre[preIndex.index], min,
                    key, size);
            }
            if (preIndex.index < size) {
                // All nodes which are in range {key..max}
                // will go in right subtree, and first such
                // node will be root of right subtree.
                root.right = constructTreeUtil(
                    pre, preIndex, pre[preIndex.index], key,
                    max, size);
            }
        }

        return root;
    }

    // The main function to construct BST from given
    // preorder traversal. This function mainly uses
    // constructTreeUtil()
    Node constructTree(int pre[], int size)
    {
        int preIndex = 0;
        return constructTreeUtil(pre, index, pre[0],
                                 Integer.MIN_VALUE,
                                 Integer.MAX_VALUE, size);
    }

    // A utility function to print inorder traversal of a
    // Binary Tree
    void printInorder(Node node)
    {
```

```
        System.out.print(node.data + " ");
        printInorder(node.right);
    }

    // Driver code
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        int pre[] = new int[] { 10, 5, 1, 7, 40, 50 };
        int size = pre.length;

        // Function call
        Node root = tree.constructTree(pre, size);
        System.out.println(
            "Inorder traversal of the constructed tree is ");
        tree.printInorder(root);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# A O(n) program for construction of BST from preorder traversal

INT_MIN = float("-infinity")
INT_MAX = float("infinity")

# A Binary tree node


class Node:

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

# Methods to get and set the value of static variable
# constructTreeUtil.preIndex for function construcTreeUtil()


def getPreIndex():
    return constructTreeUtil.preIndex
```

```
# A recursive function to construct BST from pre[].
# preIndex is used to keep track of index in pre[]


def constructTreeUtil(pre, key, mini, maxi, size):

    # Base Case
    if(getPreIndex() >= size):
        return None

    root = None

    # If current element of pre[] is in range, then
    # only it is part of current subtree
    if(key > mini and key < maxi):

        # Allocate memory for root of this subtree
        # and increment constructTreeUtil.preIndex
        root = Node(key)
        incrementPreIndex()

        if(getPreIndex() < size):

            # Construct the subtree under root
            # All nodes which are in range {min.. key} will
            # go in left subtree, and first such node will
            # be root of left subtree
            root.left = constructTreeUtil(pre,
                                          pre[getPreIndex()],
                                          mini, key, size)
        if(getPreindex() < size):

            # All nodes which are in range{key..max} will
            # go to right subtree, and first such node will
            # be root of right subtree
            root.right = constructTreeUtil(pre,
                                           pre[getPreIndex()],
                                           key, maxi, size)

    return root

# This is the main function to construct BST from given
# preorder traversal. This function mainly uses
# constructTreeUtil()
```

```
# A utility function to print inorder traversal of Binary Tree
def printInorder(node):

    if node is None:
        return
    printInorder(node.left)
    print node.data,
    printInorder(node.right)


# Driver code
pre = [10, 5, 1, 7, 40, 50]

# Function call
root = constructTree(pre)

print "Inorder traversal of the constructed tree: "
printInorder(root)

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

C#

```
// C# program to construct BST from given preorder traversal
using System;

// A binary tree node
public class Node {

    public int data;
    public Node left, right;

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

public class Index {
    public int index = 0;
}
```

```
// A recursive function to construct BST from pre[].
// preIndex is used to keep track of index in pre[].
public virtual Node constructTreeUtil(int[] pre,
                                      Index preIndex,
                                      int key, int min,
                                      int max, int size)
{

    // Base case
    if (preIndex.index >= size) {
        return null;
    }

    Node root = null;

    // If current element of pre[] is in range, then
    // only it is part of current subtree
    if (key > min && key < max) {

        // Allocate memory for root of this subtree
        // and increment *preIndex
        root = new Node(key);
        preIndex.index = preIndex.index + 1;

        if (preIndex.index < size) {

            // Construct the subtree under root
            // All nodes which are in range
            // {min .. key} will go in left
            // subtree, and first such node will
            // be root of left subtree.
            root.left = constructTreeUtil(
                pre, preIndex, pre[preIndex.index], min,
                key, size);
        }
        if (preIndex.index < size) {
            // All nodes which are in range
            // {key..max} will go in right
            // subtree, and first such node
            // will be root of right subtree.
            root.right = constructTreeUtil(
                pre, preIndex, pre[preIndex.index], key,
                max, size);
        }
    }

    return root;
```

```
    // constructTreeUtil()
    public virtual Node constructTree(int[] pre, int size)
    {

        return constructTreeUtil(pre, index, pre[0],
                                 int.MinValue, int.MaxValue,
                                 size);
    }

    // A utility function to print inorder traversal of a
    // Binary Tree
    public virtual 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)
    {
        BinaryTree tree = new BinaryTree();
        int[] pre = new int[] { 10, 5, 1, 7, 40, 50 };
        int size = pre.Length;

        // Function call
        Node root = tree.constructTree(pre, size);
        Console.WriteLine(
            "Inorder traversal of the constructed tree is ");
        tree.printInorder(root);
    }
}

// This code is contributed by Shrikant13
```

Output

```
Inorder traversal of the constructed tree:
1 5 7 10 40 50
```

Time Complexity: O(n)

information about the topic discussed above.

Method 3 (O(n^2) time complexity):

Simply do that just by using the recursion concept and iterating through the array of the given elements like below.

Java

```
/*Construct a BST from given pre-order traversal
for example if the given traversal is {10, 5, 1, 7, 40, 50},
then the output should be the root of the following tree.
     10
   /   \
  5     40
 /  \      \
1    7      50 */

class Node {
    int data;
    Node left, right;
    Node(int data)
    {
        this.data = data;
        this.left = this.right = null;
    }
}

class CreateBSTFromPreorder {
    private static Node node;

    // This will create the BST
    public static Node createNode(Node node, int data)
    {
        if (node == null)
            node = new Node(data);

        if (node.data > data)
            node.left = createNode(node.left, data);
        if (node.data < data)
            node.right = createNode(node.right, data);

        return node;
    }
```

```
    }
    // A function to print BST in inorder
    public static void inorderRec(Node root)
    {
        if (root != null) {
            inorderRec(root.left);
            System.out.println(root.data);
            inorderRec(root.right);
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        int[] nodeData = { 10, 5, 1, 7, 40, 50 };

        for (int i = 0; i < nodeData.length; i++) {
            create(nodeData[i]);
        }
        inorderRec(node);
    }
}
```

C#

```
/*Construct a BST from given pre-order traversal
for example if the given traversal is {10, 5, 1, 7, 40, 50},
then the output should be the root of the following tree.
     10
   /   \
  5     40
 /  \      \
1    7      50 */
using System;
public class Node {
    public int data;
    public Node left, right;
    public Node(int data)
    {
        this.data = data;
        this.left = this.right = null;
    }
}

public class CreateBSTFromPreorder {
```

```
    {
        if (node == null)
            node = new Node(data);

        if (node.data > data)
            node.left = createNode(node.left, data);
        if (node.data < data)
            node.right = createNode(node.right, data);

        return node;
    }

    // A wrapper function of createNode
    public static void create(int data)
    {
        node = createNode(node, data);
    }

    // A function to print BST in inorder
    public static void inorderRec(Node root)
    {
        if (root != null) {
            inorderRec(root.left);
            Console.WriteLine(root.data);
            inorderRec(root.right);
        }
    }

    // Driver Code
    public static void Main(String[] args)
    {
        int[] nodeData = { 10, 5, 1, 7, 40, 50 };
        for (int i = 0; i < nodeData.Length; i++) {
            create(nodeData[i]);
        }
        inorderRec(node);
    }
}

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

Output

```
1
5
```

```
40
50
```


Previous Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Construct BST from given preorder traversal | Set 2**
11, Oct 12

05 **Construct BST from its given level order traversal**
02, Oct 17

02 **Construct BST from given preorder**

04 **Construct a BST from given**

03 **Find postorder traversal of BST from preorder traversal**
22, Jun 18

04 **Number of elements smaller than root using preorder traversal of a BST**
31, Aug 18

07 **Check if a given array can represent Preorder Traversal of Binary Search Tree**
30, Oct 15

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

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : shrikanth13, rathbhupendra, Akanksha_Rai, SatvikNema, roysubham505, Rajput-Ji, duke0122, rhyscompton, zeekgeek, harmonpreet012

Article Tags : Binary Search Tree

Practice Tags : Binary Search Tree

Improve Article | Report Issue

Load Comments

GeeksforGeeks

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

feedback@geeksforgeeks.org