Construct BST from given preorder traversal | Set 1

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 root of following tree.

```
     10
   /   \
  5     40
 /  \      \
1    7      50
```

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 first element which is greater than root. Let the index be 'i'. The values between root and 'i' will be part of left subtree, and the values between 'i+1' and 'n-1' will be part of 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.

```
            10
          /    \
         /      \
  {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 <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;
```

```
    // The first node in preorder traversal is root. So take the node at
    // preIndex from pre[] and make it root, and increment preIndex
    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 program to test above functions
int main ()
{
    int pre[] = {10, 5, 1, 7, 40, 50};
    int size = sizeof( pre ) / sizeof( pre[0] );

    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;
        }

        // 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()
    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);
    }

    // Driver program to test above functions
    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);
        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) 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

# A recurseive function to construct Full from pre[].
# preIndex is used to keep track of index in pre[[].
def constructTreeUtil(pre, low, high, size):

    # Base Case
    if( getPreIndex() >= size or 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

    # Search for the first element greater than root
    for i in range(low, high+1):
        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, getPreIndex(),  i-1 , size)

    root.right = constructTreeUtil(pre, i, high, size)

    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
    return constructTreeUtil(pre, 0, size-1, size)
```

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


# Driver program to test above function
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)
```

Output:

```
Inorder traveral 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 from 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 root node. To construct the left subtree, set the range as {INT_MIN ...root->data}. If a values is in the range {INT_MIN .. root->data}, the values is part part of left subtree. To construct the right subtree, set the range as {root->data..max .. INT_MAX}.

C

```
/* A O(n) program for construction of BST from preorder traversal */
#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;
};

// 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
    if( key > min && key < max )
    {
        // Allocate memory for root of this subtree and increment *preIndex
        root = newNode ( key );
        *preIndex = *preIndex + 1;

        if (*preIndex < size)
        {
            // Contruct 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 );

            // 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 program to test above functions
int main ()
{
    int pre[] = {10, 5, 1, 7, 40, 50};
    int size = sizeof( pre ) / sizeof( pre[0] );

    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;
        }

        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) {

                // Contruct 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);

                // 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) {
        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) {
        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) 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

def incrementPreIndex():
    constructTreeUtil.preIndex += 1

# 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)

            # 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()
def constructTree(pre):
    constructTreeUtil.preIndex = 0
    size = len(pre)
    return constructTreeUtil(pre, pre[0], INT_MIN, INT_MAX, size)


# 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 program to test above function
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)
```

Output:

```
Inorder traversal of Binary Tree:
1 5 7 10 40 50
```

Time Complexity: O(n)

We will soon publish a O(n) iterative solution as a separate post.