|  |
| --- |
| // C++ program to convert a left unbalanced BST to  // a balanced BST  #include <bits/stdc++.h>  using namespace std;    struct Node  {      int data;      Node\* left,  \*right;  };    /\* This function traverse the skewed binary tree and     stores its nodes pointers in vector nodes[] \*/  void storeBSTNodes(Node\* root, vector<Node\*> &nodes)  {      // Base case      if (root==NULL)          return;        // Store nodes in Inorder (which is sorted      // order for BST)      storeBSTNodes(root->left, nodes);      nodes.push\_back(root);      storeBSTNodes(root->right, nodes);  }    /\* Recursive function to construct binary tree \*/  Node\* buildTreeUtil(vector<Node\*> &nodes, int start,                     int end)  {      // base case      if (start > end)          return NULL;        /\* Get the middle element and make it root \*/      int mid = (start + end)/2;      Node \*root = nodes[mid];        /\* Using index in Inorder traversal, construct         left and right subtress \*/      root->left  = buildTreeUtil(nodes, start, mid-1);      root->right = buildTreeUtil(nodes, mid+1, end);        return root;  }    // This functions converts an unbalanced BST to  // a balanced BST  Node\* buildTree(Node\* root)  {      // Store nodes of given BST in sorted order      vector<Node \*> nodes;      storeBSTNodes(root, nodes);        // Constructs BST from nodes[]      int n = nodes.size();      return buildTreeUtil(nodes, 0, n-1);  }    // Utility function to create a new node  Node\* newNode(int data)  {      Node\* node = new Node;      node->data = data;      node->left = node->right = NULL;      return (node);  }    /\* Function to do preorder traversal of tree \*/  void preOrder(Node\* node)  {      if (node == NULL)          return;      printf("%d ", node->data);      preOrder(node->left);      preOrder(node->right);  }    // Driver program  int main()  {      /\* Constructed skewed binary tree is                  10                 /                8               /              7             /            6           /          5   \*/        Node\* root = newNode(10);      root->left = newNode(8);      root->left->left = newNode(7);      root->left->left->left = newNode(6);      root->left->left->left->left = newNode(5);        root = buildTree(root);        printf("Preorder traversal of balanced "              "BST is : \n");      preOrder(root);        return 0;  } |

Output :

Preorder traversal of balanced BST is :

7 5 6 8 10

 using System;

using System.Collections.Generic;

// C# program to convert a left unbalanced BST to a balanced BST

/\* A binary tree node has data, pointer to left child

and a pointer to right child \*/

public class Node

{

public int data;

public Node left, right;

public Node(int data)

{

this.data = data;

left = right = null;

}

}

public class BinaryTree

{

public Node root;

/\* This function traverse the skewed binary tree and

stores its nodes pointers in vector nodes[] \*/

public virtual void storeBSTNodes(Node root, List<Node> nodes)

{

// Base case

if (root == null)

{

return;

}

// Store nodes in Inorder (which is sorted

// order for BST)

storeBSTNodes(root.left, nodes);

nodes.Add(root);

storeBSTNodes(root.right, nodes);

}

/\* Recursive function to construct binary tree \*/

public virtual Node buildTreeUtil(List<Node> nodes, int start, int end)

{

// base case

if (start > end)

{

return null;

}

/\* Get the middle element and make it root \*/

int mid = (start + end) / 2;

Node node = nodes[mid];

/\* Using index in Inorder traversal, construct

left and right subtress \*/

node.left = buildTreeUtil(nodes, start, mid - 1);

node.right = buildTreeUtil(nodes, mid + 1, end);

return node;

}

// This functions converts an unbalanced BST to

// a balanced BST

public virtual Node buildTree(Node root)

{

// Store nodes of given BST in sorted order

List<Node> nodes = new List<Node>();

storeBSTNodes(root, nodes);

// Constructs BST from nodes[]

int n = nodes.Count;

return buildTreeUtil(nodes, 0, n - 1);

}

/\* Function to do preorder traversal of tree \*/

public virtual void preOrder(Node node)

{

if (node == null)

{

return;

}

Console.Write(node.data + " ");

preOrder(node.left);

preOrder(node.right);

}

// Driver program to test the above functions

public static void Main(string[] args)

{

/\* Constructed skewed binary tree is

10

/

8

/

7

/

6

/

5 \*/

BinaryTree tree = new BinaryTree();

tree.root = new Node(10);

tree.root.left = new Node(8);

tree.root.left.left = new Node(7);

tree.root.left.left.left = new Node(6);

tree.root.left.left.left.left = new Node(5);

tree.root = tree.buildTree(tree.root);

Console.WriteLine("Preorder traversal of balanced BST is :");

tree.preOrder(tree.root);

}

}

// This code is contributed by Shrikant13

Output :

Preorder traversal of balanced BST is :

7 5 6 8 10

/\* C++ program to check if a tree

is height-balanced or not \*/

#include <bits/stdc++.h>

using namespace std;

#define bool int

/\* 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;

};

/\* The function returns true if root is

balanced else false The second parameter

is to store the height of tree. Initially,

we need to pass a pointer to a location with

value as 0. We can also write a wrapper

over this function \*/

bool isBalanced(node\* root, int\* height)

{

/\* lh --> Height of left subtree

rh --> Height of right subtree \*/

int lh = 0, rh = 0;

/\* l will be true if left subtree is balanced

and r will be true if right subtree is balanced \*/

int l = 0, r = 0;

if (root == NULL) {

\*height = 0;

return 1;

}

/\* Get the heights of left and right subtrees in lh and rh

And store the returned values in l and r \*/

l = isBalanced(root->left, &lh);

r = isBalanced(root->right, &rh);

/\* Height of current node is max of heights of left and

right subtrees plus 1\*/

\*height = (lh > rh ? lh : rh) + 1;

/\* If difference between heights of left and right

subtrees is more than 2 then this node is not balanced

so return 0 \*/

if (abs(lh - rh) >= 2)

return 0;

/\* If this node is balanced and left and right subtrees

are balanced then return true \*/

else

return l && r;

}

/\* UTILITY FUNCTIONS TO TEST isBalanced() FUNCTION \*/

/\* Helper function that allocates a new node with the

given data and NULL left and right pointers. \*/

node\* newNode(int data)

{

node\* Node = new node();

Node->data = data;

Node->left = NULL;

Node->right = NULL;

return (Node);

}

// Driver code

int main()

{

int height = 0;

/\* Constructed binary tree is

1

/ \

2 3

/ \ /

4 5 6

/

7

\*/

node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

root->right->left = newNode(6);

root->left->left->left = newNode(7);

if (isBalanced(root, &height))

cout << "Tree is balanced";

else

cout << "Tree is not balanced";

return 0;

}

// This is code is contributed by rathbhupendra

Output

Tree is balanced