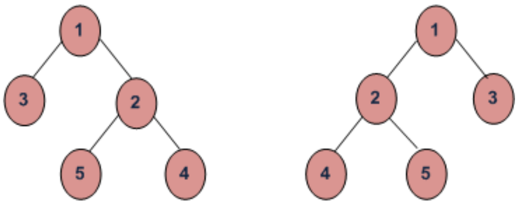
Write an Efficient Function to Convert a Binary Tree into its Mirror Tree

Mirror of a Tree: Mirror of a Binary Tree T is another Binary Tree M(T) with left and right children of all non-leaf nodes interchanged.



Mirror Trees

Trees in the above figure are mirror of each other

We strongly recommend that you click here and practice it, before moving on to the solution.

Algorithm – Mirror(tree):

```
(1)  Call Mirror for left-subtree    i.e., Mirror(left-subtree)
(2)  Call Mirror for right-subtree  i.e., Mirror(right-subtree)
(3)  Swap left and right subtrees.
          temp = left-subtree
          left-subtree = right-subtree
          right-subtree = temp
```

Program:

C

```
#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;
};

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)

{
  struct node* node = (struct node*)
                       malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;

  return(node);
}


/* Change a tree so that the roles of the  left and
    right pointers are swapped at every node.

 So the tree...
       4
      / \
     2   5
    / \
   1   3

 is changed to...
       4
      / \
      / \
     5   2
        / \
       3   1
*/
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}


/* Helper function to test mirror(). Given a binary
   search tree, print out its data elements in
   increasing sorted order.*/
void inOrder(struct node* node)
{
  if (node == NULL)
    return;

  inOrder(node->left);
  printf("%d ", node->data);

  inOrder(node->right);
}


/* Driver program to test mirror() */
int main()
{
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);

  /* Print inorder traversal of the input tree */
  printf("\n Inorder traversal of the constructed tree is \n");
  inOrder(root);

  /* Convert tree to its mirror */
  mirror(root);

  /* Print inorder traversal of the mirror tree */
  printf("\n Inorder traversal of the mirror tree is \n");
  inOrder(root);

  getchar();
  return 0;
}
```

Java

```
// Java program to convert binary tree into its mirror

/* Class containing left and right child of current
   node and key value*/
class Node
{
    int data;
    Node left, right;

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

```
class BinaryTree
{
    Node root;

    void mirror()
    {
        root = mirror(root);
    }

    Node mirror(Node node)
    {
        if (node == null)
            return node;

        /* do the subtrees */
        Node left = mirror(node.left);
        Node right = mirror(node.right);

        /* swap the left and right pointers */
        node.left = right;
        node.right = left;

        return node;
    }

    void inOrder()
    {
        inOrder(root);
    }

    /* Helper function to test mirror(). Given a binary
       search tree, print out its data elements in
       increasing sorted order.*/
    void inOrder(Node node)
    {
        if (node == null)
            return;

        inOrder(node.left);
        System.out.print(node.data + " ");

        inOrder(node.right);
    }

    /* testing for example nodes */
    public static void main(String args[])
    {
        /* creating a binary tree and entering the nodes */
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        /* print inorder traversal of the input tree */
        System.out.println("Inorder traversal of input tree is :");
        tree.inOrder();
        System.out.println("");

        /* convert tree to its mirror */
        tree.mirror();

        /* print inorder traversal of the minor tree */
        System.out.println("Inorder traversal of binary tree is : ");
        tree.inOrder();

    }
}
```

Time & Space Complexities: This program is similar to traversal of tree space and time complexities will be same as Tree traversal (Please see our Tree Traversal post for details)