Foldable Binary Trees

Question: Given a binary tree, find out if the tree can be folded or not.

A tree can be folded if left and right subtrees of the tree are structure wise mirror image of each other. An empty tree is considered as foldable.

```
Consider the below trees:
(a) and (b) can be folded.
(c) and (d) cannot be folded.

(a)
       10
     /    \
    7      15
     \    /
      9  11

(b)
        10
       /  \
      7    15
     /      \
    9       11

(c)
        10
       /  \
      7   15
     /    /
    5   11

(d)

         10
       /   \
      7     15
    /  \    /
   9   10  12
```

Method 1 (Change Left subtree to its Mirror and compare it with Right subtree)

Algorithm: isFoldable(root)

```
1) If tree is empty, then return true.
2) Convert the left subtree to its mirror image
    mirror(root->left); /* See this post */
3) Check if the structure of left subtree and right subtree is same
   and store the result.
    res = isStructSame(root->left, root->right); /*isStructSame()
        recursively compares structures of two subtrees and returns
        true if structures are same */
4) Revert the changes made in step (2) to get the original tree.
    mirror(root->left);
5) Return result res stored in step 2.
```

Thanks to ajaym for suggesting this approach.

C

```
#include<stdio.h>
#include<stdlib.h>
```

```
/* You would want to remove below 3 lines if your compiler
   supports bool, true and false */
#define bool int
#define true 1
#define false 0

/* 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;
};

/* converts a tree to its mrror image */
void mirror(struct node* node);

/* returns true if structure of two trees a and b is same
   Only structure is considered for comparison, not data! */
bool isStructSame(struct node *a, struct node *b);

/* Returns true if the given tree is foldable */
bool isFoldable(struct node *root)
{
  bool res;

  /* base case */
  if(root == NULL)
    return true;

  /* convert left subtree to its mirror */
  mirror(root->left);

  /* Compare the structures of the right subtree and mirrored
    left subtree */
  res = isStructSame(root->left, root->right);

  /* Get the originial tree back */
  mirror(root->left);

  return res;
}


bool isStructSame(struct node *a, struct node *b)
{
  if (a == NULL && b == NULL)
  {  return true; }
  if ( a != NULL && b != NULL &&
       isStructSame(a->left, b->left) &&
       isStructSame(a->right, b->right)
     )
  {  return true; }

  return false;
}


/* UTILITY FUNCTIONS */
/* Change a tree so that the roles of the  left and
    right pointers are swapped at every node.
    See http://geeksforgeeks.org/?p=662 for details */
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

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

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

/* 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);
}

/* Driver program to test mirror() */
int main(void)
{
  /* The constructed binary tree is
         1
       /   \
      2     3
      \    /
       4  5
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->right->left  = newNode(4);
  root->left->right = newNode(5);

  if(isFoldable(root) == 1)
  { printf("\n tree is foldable"); }
  else
  { printf("\n tree is not foldable"); }

  getchar();
  return 0;
}
```

Java

```
// Java program to check foldable binary tree

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

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

class BinaryTree
{
    Node root;

    /* Returns true if the given tree is foldable */
```

```
boolean isFoldable(Node node)
{
    boolean res;

    /* base case */
    if (node == null)
        return true;

    /* convert left subtree to its mirror */
    mirror(node.left);

    /* Compare the structures of the right subtree and mirrored
     left subtree */
    res = isStructSame(node.left, node.right);

    /* Get the originial tree back */
    mirror(node.left);

    return res;
}

boolean isStructSame(Node a, Node b)
{
    if (a == null && b == null)
        return true;
    if (a != null && b != null
            && isStructSame(a.left, b.left)
            && isStructSame(a.right, b.right))
        return true;

    return false;
}

/* UTILITY FUNCTIONS */

/* Change a tree so that the roles of the  left and
   right pointers are swapped at every node.
   See http://geeksforgeeks.org/?p=662 for details */
void mirror(Node node)
{
    if (node == null)
        return;
    else
    {
        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;
    }
}

/* Driver program to test above functions */
public static void main(String args[])
{
    BinaryTree tree = new BinaryTree();

    /* The constructed binary tree is
         1
       /   \
      2     3
       \    /
        4  5
    */
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.right.left = new Node(4);
```

```
        tree.root.right.left = new Node(4);
        tree.root.left.right = new Node(5);

        if (tree.isFoldable(tree.root))
            System.out.println("tree is foldable");
        else
            System.out.println("Tree is not foldable");
    }

}

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

Time complexity: O(n)

Method 2 (Check if Left and Right subtrees are Mirror)
There are mainly two functions:

// Checks if tree can be folded or not

```
IsFoldable(root)
1) If tree is empty then return true
2) Else check if left and right subtrees are structure wise mirrors of
    each other. Use utility function IsFoldableUtil(root->left,
    root->right) for this.
```

// Checks if n1 and n2 are mirror of each other.

```
IsFoldableUtil(n1, n2)
1) If both trees are empty then return true.
2) If one of them is empty and other is not then return false.
3) Return true if following conditions are met
   a) n1->left is mirror of n2->right
   b) n1->right is mirror of n2->left
```

C

```
#include<stdio.h>
#include<stdlib.h>

/* You would want to remove below 3 lines if your compiler
   supports bool, true and false */
#define bool int
#define true 1
#define false 0

/* 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 that checks if trees with roots as n1 and n2
  are mirror of each other */
bool IsFoldableUtil(struct node *n1, struct node *n2);

/* Returns true if the given tree can be folded */
bool IsFoldable(struct node *root)
{
     if (root == NULL)
     { return true;  }

     return IsFoldableUtil(root->left, root->right);
}

/* A utility function that checks if trees with roots as n1 and n2
```

```
  are mirror of each other */
bool IsFoldableUtil(struct node *n1, struct node *n2)
{
    /* If both left and right subtrees are NULL,
      then return true */
    if (n1 == NULL && n2 == NULL)
    {  return true;  }

    /* If one of the trees is NULL and other is not,
      then return false */
 if (n1 == NULL || n2 == NULL)
    {  return false; }

    /* Otherwise check if left and right subtrees are mirrors of
       their counterparts */
 return IsFoldableUtil(n1->left, n2->right) &&
           IsFoldableUtil(n1->right, n2->left);
}

/*UTILITY FUNCTIONS */
/* 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);
}

/* Driver program to test mirror() */
int main(void)
{
  /* The constructed binary tree is
         1
       /   \
      2     3
      \    /
       4  5
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->right  = newNode(4);
  root->right->left = newNode(5);

  if(IsFoldable(root) == true)
  { printf("\n tree is foldable"); }
  else
  { printf("\n tree is not foldable"); }

  getchar();
  return 0;
}
```

Java

```
// Java program to check foldable binary tree

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

    Node(int item)
```

```java
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
     Node root;

    /* Returns true if the given tree can be folded */
    boolean IsFoldable(Node node)
    {
        if (node == null)
            return true;

        return IsFoldableUtil(node.left, node.right);
    }

    /* A utility function that checks if trees with roots as n1 and n2
     are mirror of each other */
    boolean IsFoldableUtil(Node n1, Node n2)
    {

        /* If both left and right subtrees are NULL,
         then return true */
        if (n1 == null && n2 == null)
            return true;

        /* If one of the trees is NULL and other is not,
         then return false */
        if (n1 == null || n2 == null)
            return false;

        /* Otherwise check if left and right subtrees are mirrors of
         their counterparts */
        return IsFoldableUtil(n1.left, n2.right)
                && IsFoldableUtil(n1.right, n2.left);
    }

    /* Driver program to test above functions */
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();

        /* The constructed binary tree is
             1
           /   \
          2     3
           \    /
            4  5
        */
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.right.left = new Node(4);
        tree.root.left.right = new Node(5);

        if (tree.IsFoldable(tree.root))
            System.out.println("tree is foldable");
        else
            System.out.println("Tree is not foldable");

    }

}

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