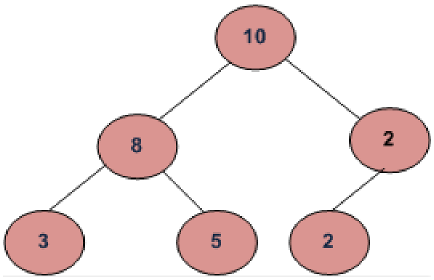
Root to leaf path sum equal to a given number

Given a binary tree and a number, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given number. Return false if no such path can be found.



For example, in the above tree root to leaf paths exist with following sums.

21 –> 10 – 8 – 3
23 –> 10 – 8 – 5
14 –> 10 – 2 – 2

So the returned value should be true only for numbers 21, 23 and 14. For any other number, returned value should be false.

Algorithm:
Recursively check if left or right child has path sum equal to (number – value at current node)

Implementation:

C

```
#include<stdio.h>
#include<stdlib.h>
#define bool int

/* 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;
};

/*
 Given a tree and a sum, return true if there is a path from the root
 down to a leaf, such that adding up all the values along the path
 equals the given sum.

 Strategy: subtract the node value from the sum when recurring down,
 and check to see if the sum is 0 when you run out of tree.
*/
bool hasPathSum(struct node* node, int sum)
{
  /* return true if we run out of tree and sum==0 */
  if (node == NULL)
  {
     return (sum == 0);
  }

  else
  {
    bool ans = 0;
    /* otherwise check both subtrees */
    int subSum = sum - node->data;

    /* If we reach a leaf node and sum becomes 0 then return true*/
    if ( subSum == 0 && node->left == NULL && node->right == NULL )
      return 1;

    if(node->left)
      ans = ans || hasPathSum(node->left, subSum);
    if(node->right)
      ans = ans || hasPathSum(node->right, subSum);

    return ans;
  }
}

/* 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 above functions*/
int main()
{

  int sum = 21;

  /* Constructed binary tree is
            10
          /   \
        8      2
      /  \    /
    3     5  2
  */
  struct node *root = newnode(10);
  root->left        = newnode(8);
  root->right       = newnode(2);
  root->left->left  = newnode(3);
  root->left->right = newnode(5);
  root->right->left = newnode(2);

  if(hasPathSum(root, sum))
    printf("There is a root-to-leaf path with sum %d", sum);
  else
    printf("There is no root-to-leaf path with sum %d", sum);

  getchar();
  return 0;
}
```

Java

```
// Java program to print to print root to leaf path sum equal to
// a given number

/* 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;

    /*
     Given a tree and a sum, return true if there is a path from the root
     down to a leaf, such that adding up all the values along the path
     equals the given sum.

     Strategy: subtract the node value from the sum when recurring down,
     and check to see if the sum is 0 when you run out of tree.
     */

    boolean haspathSum(Node node, int sum)
    {
        if (node == null)
            return (sum == 0);
        else
        {
            boolean ans = false;

            /* otherwise check both subtrees */
            int subsum = sum - node.data;
            if (subsum == 0 && node.left == null && node.right == null)
                return true;
            if (node.left != null)
                ans = ans || haspathSum(node.left, subsum);
            if (node.right != null)
                ans = ans || haspathSum(node.right, subsum);
            return ans;
        }
    }

    /* Driver program to test the above functions */
    public static void main(String args[])
    {
        int sum = 21;

        /* Constructed binary tree is
              10
             /  \
           8     2
          / \   /
         3   5 2
        */
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(2);

        if (tree.haspathSum(tree.root, sum))
            System.out.println("There is a root to leaf path with sum " + sum);
        else
            System.out.println("There is no root to leaf path with sum " + sum);
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Time Complexity: O(n)

References:

http://cslibrary.stanford.edu/110/BinaryTrees.html