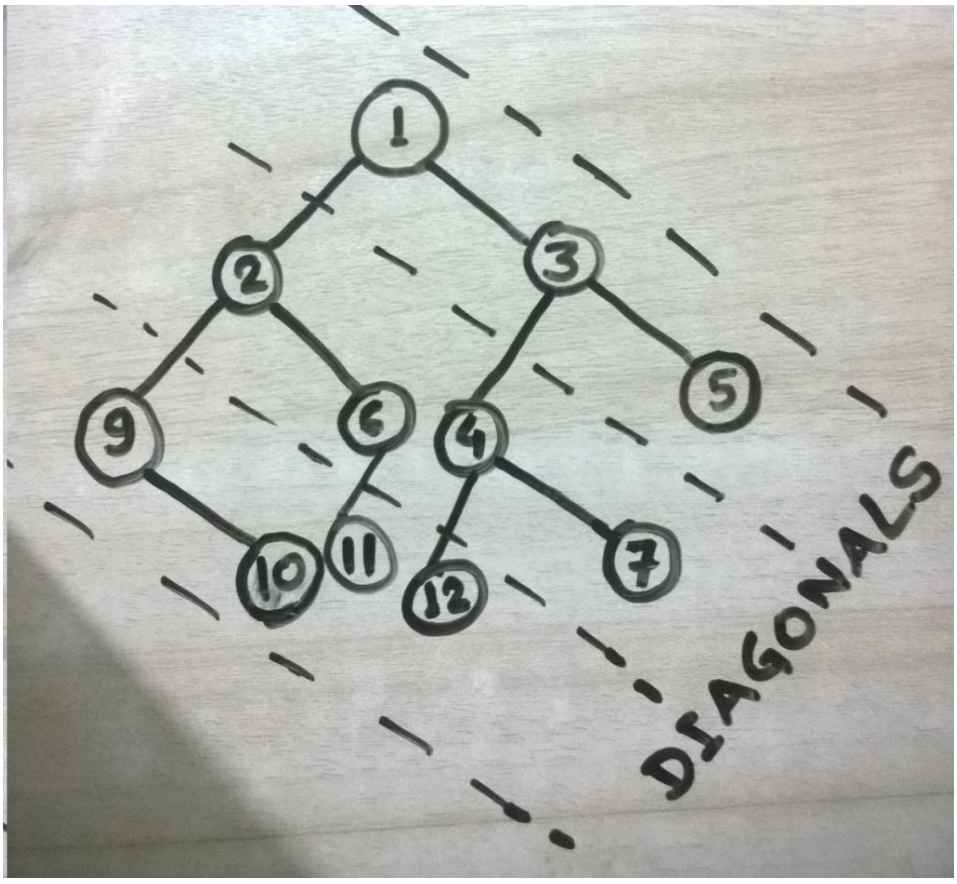
Diagonal Sum of a Binary Tree

Consider lines of slope -1 passing between nodes (dotted lines in below diagram). Diagonal sum in a binary tree is sum of all node's data lying between these lines. Given a Binary Tree, print all diagonal sums.

For the following input tree, output should be 9, 19, 42.
9 is sum of 1, 3 and 5.
19 is sum of 2, 6, 4 and 7.
42 is sum of 9, 10, 11 and 12.



We strongly recommend to minimize your browser and try this yourself first

Algorithm:

The idea is to keep track of vertical distance from top diagonal passing through root. We increment the vertical distance we go down to next diagonal.

1. Add root with vertical distance as 0 to the queue.
2. Process the sum of all right child and right of right child and so on.
3. Add left child current node into the queue for later processing. The vertical distance of left child is vertical distance of current node plus 1.
4. Keep doing 2nd, 3rd and 4th step till the queue is empty.

Following is Java implementation of above idea.

Java

```
// Java Program to find diagonal sum in a Binary Tree
import java.util.*;
import java.util.Map.Entry;

//Tree node
class TreeNode
{
    int data; //node data
    int vd; //vertical distance diagonally
    TreeNode left, right; //left and right child's reference

    // Tree node constructor
    public TreeNode(int data)
    {
        this.data = data;
        vd = Integer.MAX_VALUE;
        left = right = null;
    }
}

// Tree class
class Tree
{
    TreeNode root;//Tree root

    // Tree constructor
    public Tree(TreeNode root)  {  this.root = root;  }

    // Diagonal sum method
    public void diagonalSum()
    {
        // Queue which stores tree nodes
        Queue<TreeNode> queue = new LinkedList<TreeNode>();

        // Map to store sum of node's data lying diagonally
        Map<Integer, Integer> map = new TreeMap<>();

        // Assign the root's vertical distance as 0.
        root.vd = 0;

        // Add root node to the queue
        queue.add(root);

        // Loop while the queue is not empty
        while (!queue.isEmpty())
        {
            // Remove the front tree node from queue.
```

```
            TreeNode curr = queue.remove();

            // Get the vertical distance of the dequeued node.
            int vd = curr.vd;

            // Sum over this node's right-child, right-of-right-child
            // and so on
            while (curr != null)
            {
                int prevSum = (map.get(vd) == null)? 0: map.get(vd);
                map.put(vd, prevSum + curr.data);

                // If for any node the left child is not null add
                // it to the queue for future processing.
                if (curr.left != null)
                {
                    curr.left.vd = vd+1;
                    queue.add(curr.left);
                }

                // Move to the current node's right child.
                curr = curr.right;
            }
        }

        // Make an entry set from map.
        Set<Entry<Integer, Integer>> set = map.entrySet();

        // Make an iterator
        Iterator<Entry<Integer, Integer>> iterator = set.iterator();

        // Traverse the map elements using the iterator.
         System.out.print("Diagonal sum in a binary tree is - ");
        while (iterator.hasNext())
        {
            Map.Entry<Integer, Integer> me = iterator.next();

            System.out.print(me.getValue()+" ");
        }
    }
}

//Driver class
public class DiagonalSum
{
    public static void main(String[] args)
    {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(9);
        root.left.right = new TreeNode(6);
        root.right.left = new TreeNode(4);
        root.right.right = new TreeNode(5);
        root.right.left.left = new TreeNode(12);
        root.right.left.right = new TreeNode(7);
        root.left.right.left = new TreeNode(11);
        root.left.left.right = new TreeNode(10);
        Tree tree = new Tree(root);
        tree.diagonalSum();
    }
}
```

Run on IDE

C++

```
#include <iostream>
#include <stdlib.h>
#include <map>
using namespace std;

struct node
{
     int data;
     struct node* left;
     struct node* right;
};

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

     return node;
}

/* root - root of the binary tree
vd - vertical distance diagonally
diagonalSum - map to store Diagonal Sum(Passed by Reference) */

void diagonalSumUtil(struct node* root, int vd, map<int, int> &diagonalSum)
{
     if(!root)
          return;
     diagonalSum[vd] += root->data;

     // increase the vertical distance if left child
     diagonalSumUtil(root->left, vd + 1, diagonalSum);

     // vertical distance remains same for right child
     diagonalSumUtil(root->right, vd, diagonalSum);
}

//Function to calculate diagonal sum of given binary tree
void diagonalSum(struct node* root)
 {

     map<int, int> diagonalSum; // create a map to store Diagonal Sum
     diagonalSumUtil(root, 0, diagonalSum);

     map<int, int>::iterator it;
     cout << "Diagonal sum in a binary tree is - ";
     for(it = diagonalSum.begin(); it != diagonalSum.end(); ++it) {
          cout << it->second << " ";
 }
}

int main()
{
     struct node* root = newNode(1);
     root->left = newNode(2);
     root->right = newNode(3);
     root->left->left = newNode(9);
     root->left->right = newNode(6);
```

```
    root->right->left = newNode(4);
    root->right->right = newNode(5);
    root->right->left->right = newNode(7);
    root->right->left->left = newNode(12);
    root->left->right->left = newNode(11);
    root->left->left->right = newNode(10);

    diagonalSum(root);

    return 0;
}
//C++ implementation is contributed by Aditya Goel
```

Run on IDE

Output:

```
Diagonal sum in a binary tree is - 9 19 42
```

Exercise:

This problem was for diagonals from top to bottom and slope -1. Try the same problem for slope +1.

This article is contributed by **Kumar Gautam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

56 Comments Category: Trees

Related Posts:

- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree

- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?
- Closest leaf to a given node in Binary Tree

(Login to Rate and Mark)

2.6 Average Difficulty : **2.6/5.0**
Based on **6** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.