## Vertical Sum in a given Binary Tree | Set 1

Given a Binary Tree, find vertical sum of the nodes that are in same vertical line. Print all sums through different vertical lines.

Examples:

```
      1
    /   \
  2      3
 / \    / \
4   5  6   7
```

The tree has 5 vertical lines

Vertical-Line-1 has only one node 4 => vertical sum is 4
Vertical-Line-2: has only one node 2=> vertical sum is 2
Vertical-Line-3: has three nodes: 1,5,6 => vertical sum is 1+5+6 = 12
Vertical-Line-4: has only one node 3 => vertical sum is 3
Vertical-Line-5: has only one node 7 => vertical sum is 7

So expected output is 4, 2, 12, 3 and 7

**Solution:**
We need to check the Horizontal Distances from root for all nodes. If two nodes have the same Horizontal Distance (HD), then they are on same vertical line. The idea of HD is simple. HD for root is 0, a right edge (edge connecting to right subtree) is considered as +1 horizontal distance and a left edge is considered as -1 horizontal distance. For example, in the above tree, HD for Node 4 is at -2, HD for Node 2 is -1, HD for 5 and 6 is 0 and HD for node 7 is +2.
We can do inorder traversal of the given Binary Tree. While traversing the tree, we can recursively calculate HDs. We initially pass the horizontal distance as 0 for root. For left subtree, we pass the Horizontal Distance as Horizontal distance of root minus 1. For right subtree, we pass the Horizontal Distance as Horizontal Distance of root plus 1.

Following is Java implementation for the same. HashMap is used to store the vertical sums for different horizontal distances. Thanks to Nages for suggesting this method.

```
import java.util.HashMap;

// Class for a tree node
class TreeNode {

    // data members
    private int key;
    private TreeNode left;
    private TreeNode right;

    // Accessor methods
    public int key()        { return key; }
    public TreeNode left()  { return left; }
    public TreeNode right() { return right; }

    // Constructor
    public TreeNode(int key) { this.key = key; left = null; right = null; }

    // Methods to set left and right subtrees
    public void setLeft(TreeNode left)   { this.left = left; }
    public void setRight(TreeNode right) { this.right = right; }
}

// Class for a Binary Tree
class Tree {

    private TreeNode root;

    // Constructors
    public Tree() { root = null; }
```

```
    public Tree(TreeNode n) { root = n; }

    // Method to be called by the consumer classes like Main class
    public void VerticalSumMain() { VerticalSum(root); }

    // A wrapper over VerticalSumUtil()
    private void VerticalSum(TreeNode root) {

        // base case
        if (root == null) { return; }

        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        // Calls the VerticalSumUtil() to store the vertical sum values in hM
        VerticalSumUtil(root, 0, hM);

        // Prints the values stored by VerticalSumUtil()
        if (hM != null) {
            System.out.println(hM.entrySet());
        }
    }

    // Traverses the tree in Inoorder form and builds a hashMap hM that
    // contains the vertical sum
    private void VerticalSumUtil(TreeNode root, int hD,
                                        HashMap<Integer, Integer> hM) {

        // base case
        if (root == null) {  return; }

        // Store the values in hM for left subtree
        VerticalSumUtil(root.left(), hD - 1, hM);

        // Update vertical sum for hD of this node
        int prevSum = (hM.get(hD) == null) ? 0 : hM.get(hD);
        hM.put(hD, prevSum + root.key());

        // Store the values in hM for right subtree
        VerticalSumUtil(root.right(), hD + 1, hM);
    }
}

// Driver class to test the verticalSum methods
public class Main {

    public static void main(String[] args) {
        /* Create following Binary Tree
              1
            /    \
          2        3
         / \      / \
        4   5    6   7

        */
        TreeNode root = new TreeNode(1);
        root.setLeft(new TreeNode(2));
        root.setRight(new TreeNode(3));
        root.left().setLeft(new TreeNode(4));
        root.left().setRight(new TreeNode(5));
        root.right().setLeft(new TreeNode(6));
        root.right().setRight(new TreeNode(7));
        Tree t = new Tree(root);

        System.out.println("Following are the values of vertical sums with "
                + "the positions of the columns with respect to root ");
        t.VerticalSumMain();
    }
}
```

See this for a sample run.

Vertical Sum in Binary Tree | Set 2 (Space Optimized)

Time Complexity: O(n)