Print a Binary Tree in Vertical Order | Set 1

Given a binary tree, print it vertically. The following example illustrates vertical order traversal.

```
          1
        /    \
       2      3
      / \    / \
     4   5  6   7
              \   \
               8   9


The output of print this tree vertically will be:
4
2
1 5 6
3 8
7
9
```

We strongly recommend to minimize the browser and try this yourself first.

The idea is to traverse the tree once and get the minimum and maximum horizontal distance with respect to root. For the tree shown above, minimum distance is -2 (for node with value 4) and maximum distance is 3 (For node with value 9).
Once we have maximum and minimum distances from root, we iterate for each vertical line at distance minimum to maximum from root, and for each vertical line traverse the tree and print the nodes which lie on that vertical line.

Algorithm:

```
// min --> Minimum horizontal distance from root
// max --> Maximum horizontal distance from root
// hd  --> Horizontal distance of current node from root
findMinMax(tree, min, max, hd)
     if tree is NULL then return;

     if hd is less than min then
           min = hd;
     else if hd is greater than max then
          *max = hd;

     findMinMax(tree->left, min, max, hd-1);
     findMinMax(tree->right, min, max, hd+1);


printVerticalLine(tree, line_no, hd)
     if tree is NULL then return;

     if hd is equal to line_no, then
           print(tree->data);
     printVerticalLine(tree->left, line_no, hd-1);
     printVerticalLine(tree->right, line_no, hd+1);
```

Implementation:
Following is the implementation of above algorithm.

C++

```
#include <iostream>
using namespace std;

// A node of binary tree
```

```
// A node of binary tree
struct Node
{
    int data;
    struct Node *left, *right;
};

// A utility function to create a new Binary Tree node
Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to find min and max distances with respect
// to root.
void findMinMax(Node *node, int *min, int *max, int hd)
{
    // Base case
    if (node == NULL) return;

    // Update min and max
    if (hd < *min)  *min = hd;
    else if (hd > *max) *max = hd;

    // Recur for left and right subtrees
    findMinMax(node->left, min, max, hd-1);
    findMinMax(node->right, min, max, hd+1);
}

// A utility function to print all nodes on a given line_no.
// hd is horizontal distance of current node with respect to root.
void printVerticalLine(Node *node, int line_no, int hd)
{
    // Base case
    if (node == NULL) return;

    // If this node is on the given line number
    if (hd == line_no)
        cout << node->data << " ";

    // Recur for left and right subtrees
    printVerticalLine(node->left, line_no, hd-1);
    printVerticalLine(node->right, line_no, hd+1);
}

// The main function that prints a given binary tree in
// vertical order
void verticalOrder(Node *root)
{
    // Find min and max distances with resepect to root
    int min = 0, max = 0;
    findMinMax(root, &min, &max, 0);

    // Iterate through all possible vertical lines starting
    // from the leftmost line and print nodes line by line
    for (int line_no = min; line_no <= max; line_no++)
    {
        printVerticalLine(root, line_no, 0);
        cout << endl;
    }
}

// Driver program to test above functions
int main()
{
    // Create binary tree shown in above figure
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
```

```
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);

    cout << "Vertical order traversal is \n";
    verticalOrder(root);

    return 0;
}
```

Java

```
// Java program to print binary tree in reverse order

// A binary tree node
class Node
{
    int data;
    Node left, right;

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

class Values
{
    int max, min;
}

class BinaryTree
{
    Node root;
    Values val = new Values();

    // A utility function to find min and max distances with respect
    // to root.
    void findMinMax(Node node, Values min, Values max, int hd)
    {
        // Base case
        if (node == null)
            return;

        // Update min and max
        if (hd < min.min)
            min.min = hd;
        else if (hd > max.max)
            max.max = hd;

        // Recur for left and right subtrees
        findMinMax(node.left, min, max, hd - 1);
        findMinMax(node.right, min, max, hd + 1);
    }

    // A utility function to print all nodes on a given line_no.
    // hd is horizontal distance of current node with respect to root.
    void printVerticalLine(Node node, int line_no, int hd)
    {
        // Base case
        if (node == null)
            return;

        // If this node is on the given line number
        if (hd == line_no)
            System.out.print(node.data + " ");

        // Recur for left and right subtrees
```

```
        // Recur for left and right subtrees
        printVerticalLine(node.left, line_no, hd - 1);
        printVerticalLine(node.right, line_no, hd + 1);
    }

    // The main function that prints a given binary tree in
    // vertical order
    void verticalOrder(Node node)
    {
        // Find min and max distances with resepect to root
        findMinMax(node, val, val, 0);

        // Iterate through all possible vertical lines starting
        // from the leftmost line and print nodes line by line
        for (int line_no = val.min; line_no <= val.max; line_no++)
        {
            printVerticalLine(node, line_no, 0);
            System.out.println("");
        }
    }

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

        /* Let us construct the tree shown in above diagram */
        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);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.right.left.right = new Node(8);
        tree.root.right.right.right = new Node(9);

        System.out.println("vertical order traversal is :");
        tree.verticalOrder(tree.root);
    }
}

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

Python

```
# Program to print binary tree in vertical order

# A binary tree
class Node:
    # Constructor to create a new node
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None

# A utility function to find min and max distances with
# respect to root
def findMinMax(node, minimum, maximum, hd):

    # Base Case
    if node is None:
        return

    # Update min and max
    if hd < minimum[0] :
        minimum[0] = hd
    elif hd > maximum[0]:
        maximum[0] = hd
```

```
    # Recur for left and right subtrees
    findMinMax(node.left, minimum, maximum, hd-1)
    findMinMax(node.right, minimum, maximum, hd+1)

# A utility function to print all nodes on a given line_no
# hd is horizontal distance of current node with respect to root
def printVerticalLine(node, line_no, hd):

    # Base Case
    if node is None:
        return

    # If this node is on the given line number
    if hd == line_no:
        print node.data,

    # Recur for left and right subtrees
    printVerticalLine(node.left, line_no, hd-1)
    printVerticalLine(node.right, line_no, hd+1)

def verticalOrder(root):

    # Find min and max distances with respect to root
    minimum = [0]
    maximum = [0]
    findMinMax(root, minimum, maximum, 0)

    # Iterate through all possible lines starting
    # from the leftmost line and print nodes line by line
    for line_no in range(minimum[0], maximum[0]+1):
        printVerticalLine(root, line_no, 0)
        print


# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
root.right.left.right = Node(8)
root.right.right.right = Node(9)

print "Vertical order traversal is"
verticalOrder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Vertical order traversal is
4
2
1 5 6
3 8
7
9
```

Time Complexity: Time complexity of above algorithm is O(w*n) where w is width of Binary Tree and n is number of nodes in Binary Tree. In worst case, the value of w can be O(n) (consider a complete tree for example) and time complexity can become $O(n^2)$.

This problem can be solved more efficiently using the technique discussed in this post. We will soon be discussing complete algorithm and implementation of more efficient method.