Bottom View of a Binary Tree

Given a Binary Tree, we need to print the bottom view from left to right. A node x is there in output if x is the bottommost node at its horizontal distance. Horizontal distance of left child of a node x is equal to horizontal distance of x minus 1, and that of right child is horizontal distance of x plus 1.

Examples:

```
                      20
                    /    \
                  8       22
                /   \      \
              5      3      25
                    / \
                  10    14
```

For the above tree the output should be 5, 10, 3, 14, 25.

If there are multiple bottom-most nodes for a horizontal distance from root, then print the later one in level traversal. For example, in the below diagram, 3 and 4 are both the bottom-most nodes at horizontal distance 0, we need to print 4.

```
                      20
                    /    \
                  8       22
                /   \    /   \
              5      3 4     25
                    / \
                  10    14
```

For the above tree the output should be 5, 10, 4, 14, 25.

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

The following are steps to print Bottom View of Bianry Tree.
1. We put tree nodes in a queue for the level order traversal.
2. Start with the horizontal distance(hd) 0 of the root node, keep on adding left child to queue along with the horizontal distance as hd-1 and right child as hd+1.
3. Also, use a TreeMap which stores key value pair sorted on key.
4. Every time, we encounter a new horizontal distance or an existing horizontal distance put the node data for the horizontal distance as key. For the first time it will add to the map, next time it will replace the value. This will make sure that the bottom most element for that horizontal distance is present in the map and if you see the tree from beneath that you will see that element.

A Java based implementation is below :

C++

```
// C++ Program to print Bottom View of Binary Tree
#include<bits/stdc++.h>
using namespace std;

// Tree node class
struct Node
{
    int data; //data of the node
    int hd; //horizontal distance of the node
    Node *left, *right; //left and right references

    // Constructor of tree node
    Node(int key)
    {
        data = key;
        hd = INT_MAX;
        left = right = NULL;
    }
};

// Method that prints the bottom view.
void bottomView(Node *root)
{
    if (root == NULL)
        return;

    // Initialize a variable 'hd' with 0
    // for the root element.
    int hd = 0;

    // TreeMap which stores key value pair
    // sorted on key value
    map<int, int> m;

    // Queue to store tree nodes in level
    // order traversal
    queue<Node *> q;

    // Assign initialized horizontal distance
    // value to root node and add it to the queue.
    root->hd = hd;
    q.push(root);

    // Loop until the queue is empty (standard
    // level order loop)
    while (!q.empty())
    {
        Node *temp = q.front();
        q.pop();

        // Extract the horizontal distance value
        // from the dequeued tree node.
        hd = temp->hd;

        // Put the dequeued tree node to TreeMap
        // having key as horizontal distance. Every
        // time we find a node having same horizontal
        // distance we need to replace the data in
        // the map.
        m[hd] = temp->data;

        // If the dequeued node has a left child add
        // it to the queue with a horizontal distance hd-1.
        if (temp->left != NULL)
        {
            temp->left->hd = hd-1;
            q.push(temp->left);
        }

        // If the dequeued node has a left child add
        // it to the queue with a horizontal distance
        // hd+1.
        if (temp->right != NULL)
        {
            temp->right->hd = hd+1;
            q.push(temp->right);
        }
    }

    // Traverse the map elements using the iterator.
    for (auto i = m.begin(); i != m.end(); ++i)
        cout << i->second << " ";
}

// Driver Code
int main()
```

```
{
    Node *root = new Node(20);
    root->left = new Node(8);
    root->right = new Node(22);
    root->left->left = new Node(5);
    root->left->right = new Node(3);
    root->right->left = new Node(4);
    root->right->right = new Node(25);
    root->left->right->left = new Node(10);
    root->left->right->right = new Node(14);
    cout << "Bottom view of the given binary tree :\n"
    bottomView(root);
    return 0;
}
```

Java

```
// Java Program to print Bottom View of Binary Tree
import java.util.*;
import java.util.Map.Entry;

// Tree node class
class Node
{
    int data; //data of the node
    int hd; //horizontal distance of the node
    Node left, right; //left and right references

    // Constructor of tree node
    public Node(int key)
    {
        data = key;
        hd = Integer.MAX_VALUE;
        left = right = null;
    }
}

//Tree class
class Tree
{
    Node root; //root node of tree

    // Default constructor
    public Tree() {}

    // Parameterized tree constructor
    public Tree(Node node)
    {
        root = node;
    }

    // Method that prints the bottom view.
    public void bottomView()
    {
        if (root == null)
            return;

        // Initialize a variable 'hd' with 0 for the root element.
        int hd = 0;

        // TreeMap which stores key value pair sorted on key value
        Map<Integer, Integer> map = new TreeMap<>();

         // Queue to store tree nodes in level order traversal
        Queue<Node> queue = new LinkedList<Node>();

        // Assign initialized horizontal distance value to root
        // node and add it to the queue.
        root.hd = hd;
        queue.add(root);
        // Loop until the queue is empty (standard level order loop)
        while (!queue.isEmpty())
        {
            Node temp = queue.remove();

            // Extract the horizontal distance value from the
            // dequeued tree node.
            hd = temp.hd;

            // Put the dequeued tree node to TreeMap having key
            // as horizontal distance. Every time we find a node
            // having same horizontal distance we need to replace
            // the data in the map.
            map.put(hd, temp.data);

            // If the dequeued node has a left child add it to the
            // queue with a horizontal distance hd-1.
            if (temp.left != null)
            {
                temp.left.hd = hd-1;
                queue.add(temp.left);
            }
            // If the dequeued node has a left child add it to the
            // queue with a horizontal distance hd+1.
            if (temp.right != null)
            {
                temp.right.hd = hd+1;
                queue.add(temp.right);
            }
        }

        // Extract the entries of map into a set to traverse
        // an iterator over that.
        Set<Entry<Integer, Integer>> set = map.entrySet();

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

        // Traverse the map elements using the iterator.
        while (iterator.hasNext())
        {
            Map.Entry<Integer, Integer> me = iterator.next();
            System.out.print(me.getValue()+" ");
        }
    }
}

// Main driver class
public class BottomView
{
    public static void main(String[] args)
    {
        Node root = new Node(20);
        root.left = new Node(8);
        root.right = new Node(22);
        root.left.left = new Node(5);
        root.left.right = new Node(3);
        root.right.left = new Node(4);
        root.right.right = new Node(25);
        root.left.right.left = new Node(10);
        root.left.right.right = new Node(14);
        Tree tree = new Tree(root);
        System.out.println("Bottom view of the given binary tree:");
        tree.bottomView();
    }
}
```

Output:

```
Bottom view of the given binary tree:
5 10 4 14 25
```

Exercise: Extend the above solution to print all bottommost nodes at a horizontal distance if there are multiple bottommost nodes. For the above second example, the output should be 5 10 3 4 14 25.