Print Binary Tree

Print a binary tree in an m*n 2D string array following these rules:

1. The row number `m` should be equal to the height of the given binary tree.
2. The column number `n` should always be an odd number.
3. The root node's value (in string format) should be put in the exactly middle of the first row it can be put. The column and the row where the root node belongs will separate the rest space into two parts (**left-bottom part and right-bottom part**). You should print the left subtree in the left-bottom part and print the right subtree in the right-bottom part. The left-bottom part and the right-bottom part should have the same size. Even if one subtree is none while the other is not, you don't need to print anything for the none subtree but still need to leave the space as large as that for the other subtree. However, if two subtrees are none, then you don't need to leave space for both of them.
4. Each unused space should contain an empty string `""`.
5. Print the subtrees following the same rules.

Example 1:

Input:

```
     1
    /
   2
```

Output:

```
[["", "1", ""],
 ["2", "", ""]]
```

Example 2:

Input:

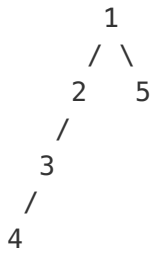
```
     1
    / \
   2   3
    \
     4
```

Output:

```
[["", "", "", "1", "", "", ""],
 ["", "2", "", "", "", "3", ""],
 ["", "", "4", "", "", "", ""]]
```

Example 3:

Input:



Output:

```
[["",  "",  "", "",  "", "", "", "1", "",  "",  "",  "",  "", "", ""]
 ["",  "",  "", "2", "", "", "", "",  "",  "",  "",  "5", "", "", ""]
 ["",  "3", "", "",  "", "", "", "",  "",  "",  "",  "",  "", "", ""]
 ["4", "",  "", "",  "", "", "", "",  "",  "",  "",  "",  "", "", ""]]
```

Note: The height of binary tree is in the range of [1, 10].

Solution 1

```
public List<List<String>> printTree(TreeNode root) {
    List<List<String>> res = new LinkedList<>();
    int height = root == null ? 1 : getHeight(root);
    int rows = height, columns = (int) (Math.pow(2, height) - 1);
    List<String> row = new ArrayList<>();
    for(int i = 0; i < columns; i++)  row.add("");
    for(int i = 0; i < rows; i++)  res.add(new ArrayList<>(row));
    populateRes(root, res, 0, rows, 0, columns - 1);
    return res;
}

public void populateRes(TreeNode root, List<List<String>> res, int row, int totalRow
s, int i, int j) {
    if (row == totalRows || root == null) return;
    res.get(row).set((i+j)/2, Integer.toString(root.val));
    populateRes(root.left, res, row+1, totalRows, i, (i+j)/2 - 1);
    populateRes(root.right, res, row+1, totalRows, (i+j)/2+1, j);
}

public int getHeight(TreeNode root) {
     if (root == null) return 0;
     return 1 + Math.max(getHeight(root.left), getHeight(root.right));
}
```

written by compton_scatter original link here

Solution 2

We can easily get the dimensions of the ans, which is height and width of the tree, by DFS. The remaining work is to recursively preorder or other traversal of the tree.

Note a big number "1234567" takes the same space as "1". I misunderstood the question and came into a bit more complex problem.

Except initializing the answer of m row and n column, the run time is O(N). Here m is the height of the tree, n = 2^m-1, and N is how many nodes of the tree. In the worst case, for example, a tree with only left child, we have m = N, so the initialization is O(N x 2^N). Fortunately, m <= 10, so initialization is O(10000). When m is a big number, this can be an issue and we will have memory error as well.

Here I assume initialization using empty string is very fast. I did a test by adding the code below to show initialization is not the limiting factor.

```
vector<vector<string>> ans;
int k = 1000;
for (int i = 0; i < k; i++)
       ans = vector<vector<string>>(h, vector<string>(w, ""));
```

k = 1, 3 ms;
k = 10, 3 ms;
k = 100, 9-12 ms;
k = 1000, ~70 ms;
So initialization seems huge, but only takes about 6-7% of total run time.

```cpp
class Solution {
public:
    vector<vector<string>> printTree(TreeNode* root) {
        int h = get_height(root), w = get_width(root);
        vector<vector<string>> ans(h, vector<string>(w, ""));
        helper(ans, root, 0, 0, w-1);
        return ans;
    }
private:
    int get_height(TreeNode* p) {
        if (!p) return 0;
        int left = get_height(p->left), right = get_height(p->right);
        return max(left, right)+1;
    }
    // width is the max(left, right)*2+1
    int get_width(TreeNode* p) {
        if (!p) return 0;
        int left = get_width(p->left), right = get_width(p->right);
        return max(left, right)*2+1;
    }
    // always put the value in the middle of the range.
    void helper(vector<vector<string>>& ans, TreeNode* p, int level, int l, int r) {
        if (!p) return;
        int mid = l+(r-l)/2;
        ans[level][mid] = to_string(p->val);
        helper(ans, p->left, level+1, l, mid-1);
        helper(ans, p->right, level+1, mid+1, r);
    }
};
```

written by zestypanda original link here

Solution 3

```
def printTree(self, root):
        if not root: return [""]

        def depth(root):
            if not root: return 0
            return max(depth(root.left), depth(root.right)) + 1

        d = depth(root)
        self.res = [[""] * (2**d - 1) for _ in xrange(d)]

        def helper(node, d, pos):
            self.res[-d - 1][pos] = str(node.val)
            if node.left: helper(node.left, d - 1, pos - 2**(d - 1))
            if node.right: helper(node.right, d - 1, pos + 2**(d - 1))

        helper(root, d - 1, 2**(d - 1) - 1)
        return self.res
```

written by lee215 original link here