257. Binary Tree Paths

Nov. 17, 2018 | 37.1K views

Average Rating: 4.39 (13 votes)

Given a binary tree, return all root-to-leaf paths.

Note: A leaf is a node with no children.

Example:

```
Input:

   1
 /   \
2     3
 \
  5

Output: ["1->2->5", "1->3"]

Explanation: All root-to-leaf paths are: 1->2->5, 1->3
```

Solution

Binary tree definition

First of all, here is the definition of the `TreeNode` which we would use in the following implementation.

Java | Python

Copy

```python
class TreeNode(object):
    """ Definition of a binary tree node."""
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

Approach 1: Recursion

The most intuitive way is to use a recursion here. One is going through the tree by considering at each step the node itself and its children. If node *is not* a leaf, one extends the current path by a node value and calls recursively the path construction for its children. If node *is* a leaf, one closes the current path and adds it into the list of paths.

Java | Python

Copy

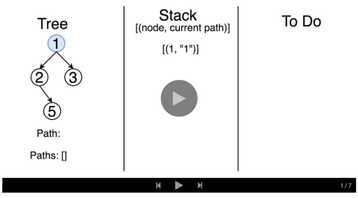
```python
class Solution:
    def binaryTreePaths(self, root):
        """
        :type root: TreeNode
        :rtype: List[str]
        """
        def construct_paths(root, path):
            if root:
                path += str(root.val)
                if not root.left and not root.right:  # if reach a leaf
                    paths.append(path)  # update paths
                else:
                    path += '->'  # extend the current path
                    construct_paths(root.left, path)
                    construct_paths(root.right, path)

        paths = []
        construct_paths(root, '')
        return paths
```

Complexity Analysis

- Time complexity : we visit each node exactly once, thus the time complexity is $\mathcal{O}(N)$, where N is the number of nodes.
- Space complexity : $\mathcal{O}(N)$. Here we use the space for a stack call and for a `paths` list to store the answer. `paths` contains as many elements as leafs in the tree and hence couldn't be larger than $\log N$ for the trees containing more than one element. Hence the space complexity is determined by a stack call. In the worst case, when the tree is completely unbalanced, *e.g.* each node has only one child node, the recursion call would occur N times (the height of the tree), therefore the storage to keep the call stack would be $\mathcal{O}(N)$. But in the best case (the tree is balanced), the height of the tree would be $\log(N)$. Therefore, the space complexity in this case would be $\mathcal{O}(\log(N))$.

Approach 2: Iterations

The approach above could be rewritten with the help of iterations. This way we initiate the stack by a root node and then at each step we pop out one node and its path. If the popped node *is* a leaf, one update the list of all paths. If not, one pushes its child nodes and corresponding paths into stack till all nodes are checked.



Java | Python

Copy

```python
class Solution:
    def binaryTreePaths(self, root):
        """
        :type root: TreeNode
        :rtype: List[str]
        """
        if not root:
            return []

        paths = []
        stack = [(root, str(root.val))]
        while stack:
            node, path = stack.pop()
            if not node.left and not node.right:
                paths.append(path)
            if node.left:
                stack.append((node.left, path + '->' + str(node.left.val)))
            if node.right:
                stack.append((node.right, path + '->' + str(node.right.val)))

        return paths
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ since each node is visited exactly once.
- Space complexity : $\mathcal{O}(N)$ as we could keep up to the entire tree.

Rate this article:

Previous Next

Comments: 12

Sort By

Type comment here... (Markdown is supported)

Preview Post

ankothari ★ 75 December 7, 2018 11:48 PM

Report

path += '->' is bad because this will create a lot of useless strings in the heap. Instead create append values in a buffer and join them with '->' at the end to the final result

30 Share Reply

SHOW 4 REPLIES

junhaowanggg ★ 547 July 5, 2019 10:54 AM

I don't quite understand the space complexity analysis in the recursion approach. I think it depends on how many leaves and the height.

- **Space:** `O(Lh)`, `L` is the number of leaves / paths.
  - Stack frame: `O(h)`

Read More

11 Share Reply

SHOW 2 REPLIES

sriharik ★ 168 May 8, 2020 9:15 AM

String Builder solution if anyone is interested. Its important to note that since StringBuilder is an object, once it reaches the end, it will not revert as we percolate up the recursive stack (It will if we used a String instead of StringBuilder). We must save the current iteration of the string during each recursive stack.

Read More

3 Share Reply

lengthOfUndefined ★ 52 April 8, 2019 3:35 AM

Report

cc: @liaison @andvary

"Here we use the space for a stack call and for a paths list to store the answer, paths contains as many elements as leafs in the tree and hence couldn't be larger than logN for the trees containing more than one element."

Read More

2 Share Reply

wzli ★ 21 December 21, 2019 9:12 AM

The space complexity consists of two sections, one is for the use of the stack, which is O(N) for the worst case.
The other section is for the use of the list, which is O(2 ^ h) for the worst case. we have 2^h leaves at most.
We can consider h, which is 2^0 + 2^1 + 2^2 + ... 2^h = N. Thus, h = log(N+1) -1. Therefore, the list

Read More

1 Share Reply

DirectionNZ ★ 11 June 25, 2019 1:15 AM

I think the space complexity should always be O(N), not matter whether the tree is balanced or not (considering the fact that author says complexity in best case would be O(log(N)) in approach 1). The `path` variable holds all the node values of the original tree no matter how the tree is shaped. So I think the space complexity is dominated by the `path` variable, not the runtime stack.

1 Share Reply

nag418 ★ 14 August 6, 2019 2:43 AM

Do I need to remove the last character from a string, using recursion way?

0 Share Reply

SHOW 1 REPLY

Mr-Programmer ★ 30 June 2, 2019 11:16 AM

In the recursion solution, dont we have to account for the number of strings we are creating during the entire process, when calculating the space complexity? If not, why?

0 Share Reply

SHOW 3 REPLIES

marazik ★ 0 November 18, 2018 3:54 AM

For the recursive solution in Java.
On line 10 (unlike line 9), you are not setting the return value to anything.

```
paths = construct_paths(root.left, path, paths);  // Line 9
paths = construct_paths(root.right, path, paths);  // Line 10
```

Read More

0 Share Reply

kris-codes ★ 1 February 15, 2020 6:47 AM

Report

Here is an iterative solution using BFS in Javascript:

```
var binaryTreePaths = function(root) {
    var paths = [];
    if (root) {
```

Read More

0 Share Reply