559. Maximum Depth of n-ary Tree

Oct. 2, 2018 | 20.6K views

Average Rating: 3.83 (6 votes)

Given a n-ary tree, find its maximum depth.

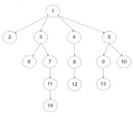
The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Example 1:

```
Input: root = [1,null,3,2,4,null,5,6]
Output: 3
```

Example 2:



```
Input: root = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null
Output: 5
```

Constraints:

- The depth of the n-ary tree is less than or equal to `1000`.
- The total number of nodes is between `[0, 10^4]`.

Solution

Tree definition

First of all, please refer to this article for the solution in case of binary tree. This article offers the same ideas with a bit of generalisation.

Here is the definition of the `TreeNode` which we would use.

Java | Python — Copy

```python
# Definition for a Node.
class Node(object):
    def __init__(self, val, children):
        self.val = val
        self.children = children
```

Approach 1: Recursion

Algorithm

The intuitive approach is to solve the problem by recursion. Here we demonstrate an example with the DFS (Depth First Search) strategy.

Java | Python — Copy

```python
class Solution(object):
    def maxDepth(self, root):
        """
        :type root: Node
        :rtype: int
        """
        if root is None:
            return 0
        elif root.children == []:
            return 1
        else:
            height = [self.maxDepth(c) for c in root.children]
            return max(height) + 1
```

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 : in the worst case, 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 completely 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: Iteration

We could also convert the above recursion into iteration, with the help of stack.

> The idea is to visit each node with the DFS strategy, while updating the max depth at each visit.

So we start from a stack which contains the root node and the corresponding depth which is `1`. Then we proceed to the iterations: pop the current node out of the stack and push the child nodes. The depth is updated at each step.

Java | Python — Copy

```python
class Solution(object):
    def maxDepth(self, root):
        """
        :type root: Node
        :rtype: int
        """
        stack = []
        if root is not None:
            stack.append((1, root))

        depth = 0
        while stack != []:
            current_depth, root = stack.pop()
            if root is not None:
                depth = max(depth, current_depth)
                for c in root.children:
                    stack.append((current_depth + 1, c))

        return depth
```

Complexity analysis

- Time complexity : $\mathcal{O}(N)$.
- Space complexity : $\mathcal{O}(N)$.

Rate this article:

Comments: 17

Sort By

Type comment here... (Markdown is supported)

Preview Post

SanD91 ★ 543 February 2, 2019 1:58 AM

Easy java solution without using collections. Beats 100%

```
class Solution {
    public int maxDepth(Node root) {
        if (root == null) return 0;
```

Read More

31 Share Reply

Heronalps ★ 151 October 27, 2018 2:00 AM

The second solution is basically BFS with a queue.

14 Share Reply

SHOW 1 REPLY

acidum ★ 28 October 6, 2018 12:10 AM

I don't think we need a null-check `if (root != null) {` inside the while of the iterative approach.

8 Share Reply

HY_huaiyu ★ 182 November 3, 2018 9:13 PM

recursion approach, I think this one is easier to understand :

```
class Solution {
    int max = 0;
    public int maxDepth(Node root) {
```

Read More

4 Share Reply

gnimwall ★ 3 November 6, 2018 7:34 PM

Second solution is BFS, the visiting order for the default test case is: 1, 3, 2, 4, 5, 6;

3 Share Reply

SHOW 3 REPLIES

sjw214 ★ 183 November 29, 2018 4:45 AM

Javascript solution:

```
class Node {
  constructor(val) {
    this.val = val;
```

Read More

1 Share Reply

SHOW 1 REPLY

try101 ★ 9 November 8, 2018 7:13 AM

how is space complexity is o(logn) in case of balanced tree .there is function call for each node so I would think o(n) . can some one explain this?

1 Share Reply

SHOW 3 REPLIES

kotaki86 ★ 37 July 8, 2019 5:54 AM

@liaison and @andvary for the second approach, Space complexity for the best case should log(n) correct ?
As we never store all the nodes in queue if tree is balanced

0 Share Reply

haoyangfan ★ 812 April 30, 2019 9:11 AM

Why the iterative solution is running so slow compared with the recursive solution?
Here is my implementation of iterative solution

```
/*
// Definition for a Node.
```

Read More

0 Share Reply

SHOW 1 REPLY

leetbunny ★ 53 January 7, 2019 6:51 PM

Is time complexity for DFS O(nlogn) given the sort/get max instead of O(n)?

0 Share Reply