

[Previous \(/articles/two-sum-less-than-k/\)](/articles/two-sum-less-than-k/) [Next \(/articles/excel-sheet-column-number/\)](/articles/excel-sheet-column-number/)

662. Maximum Width of Binary Tree [\(/problems/maximum-width-of-binary-tree/\)](/problems/maximum-width-of-binary-tree/)

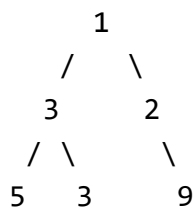
June 12, 2020 | 10.3K views

Average Rating: 4.42 (26 votes)


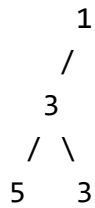
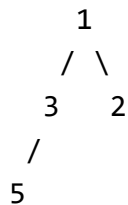
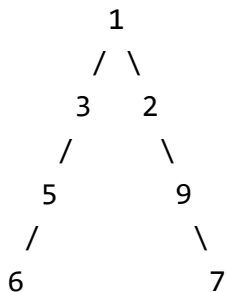
Given a binary tree, write a function to get the maximum width of the given tree. The width of a tree is the maximum width among all levels. The binary tree has the same structure as a **full binary tree**, but some nodes are null.

The width of one level is defined as the length between the end-nodes (the leftmost and right most non-null nodes in the level, where the `null` nodes between the end-nodes are also counted into the length calculation.

Example 1:

Input:**Output:** 4**Explanation:** The maximum width existing in the third level with the length 4 (5,

Example 2:

Input: Articles > 662. Maximum Width of Binary Tree**Output:** 2**Explanation:** The maximum width existing in the third level with the length 2 (5,**Example 3:****Input:****Output:** 2**Explanation:** The maximum width existing in the second level with the length 2 (3**Example 4:****Input:****Output:** 8**Explanation:** The maximum width existing in the fourth level with the length 8 (6,

Note: Answer will in the range of 32-bit signed integer.

Articles > 662. Maximum Width of Binary Tree

Solution

Overview

The problem defines the concept of **width** for the binary tree. In essence, it is about binary tree traversal, since we need to traverse the tree in order to measure its width.

As one would probably know, the common strategies to traverse a binary tree are Breadth-First Search (*a.k.a.* BFS) and Depth-First Search (*a.k.a.* DFS). Furthermore, the DFS strategy can be distinguished as *preorder* DFS, *inorder* DFS and *postorder* DFS, depending on the relative order of visit among the node itself and its child nodes.

If one is not familiar with the concepts of BFS and DFS, we have an Explore card called Queue & Stack (<https://leetcode.com/explore/learn/card/queue-stack/>) where we cover the BFS traversal (<https://leetcode.com/explore/learn/card/queue-stack/231/practical-application-queue/>) as well as the DFS traversal (<https://leetcode.com/explore/learn/card/queue-stack/232/practical-application-stack/>). Hence, in this article, we won't repeat ourselves on these concepts.

Intuition

The key to solve the problem though lie on how we **index** the nodes that are on the same level.

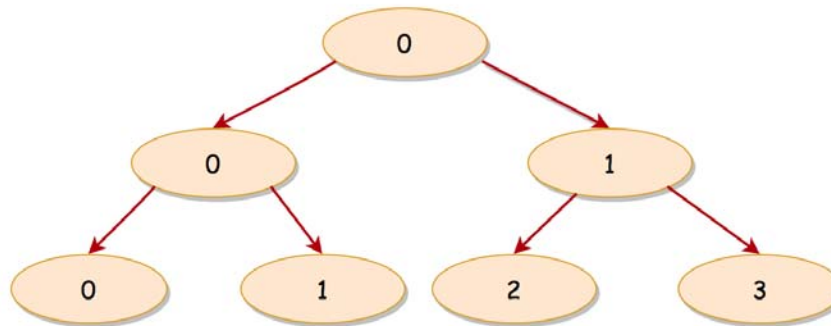
Suppose that the indices for the first and the last nodes of one particular level are C_1 and C_n respectively, we could then calculate the *width* of this level as $C_n - C_1 + 1$.

Now, let us try to come up with a schema to index the nodes, so that the problem can be solved easily with the above formula.

As we know, for a *full* binary tree, the number of nodes double at each level, since each parent node has two child nodes. Naturally, the range of our node index would double as well.

If the index of a parent node is C_i , accordingly we can define the index of its **left** child node as $2 \cdot C_i$ and the index of its **right** child node as $2 \cdot C_i + 1$.

In the following graph, we show an example of how the index works for a full binary tree, where on each node we label its index rather than its value.



With the above indexing schema, we manage to assign a unique index for each node on the same level, and in addition there is no *gap* among all the indices if it is a full binary tree.

For a non-full binary tree, the relationship between the indices of a parent and its child node still holds.

Now that we have an indexing schema, all we need to do is to assign an index for each node in the tree. Once it is done, we can calculate the *width* for each level, and finally we could return the maximal value among them as the solution.

Voila. This is the key insight to solve the problem. With this hint, we believe that one could definitely come up with some solutions.

As a spoiler alert, we will cover how to implement different solutions with BFS and DFS traversal strategies in the remaining sections.

Approach 1: BFS Traversal

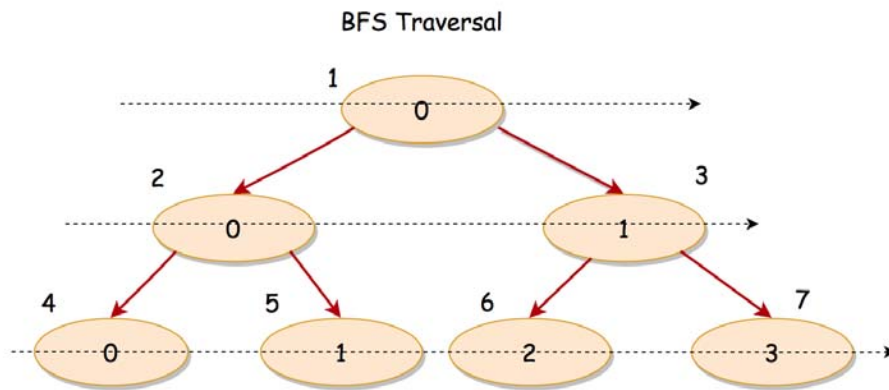
Intuition

Naturally, one might resort to the BFS traversal. After all, the width is measured among the nodes on the same level. So let us get down to the BFS traversal first.

There are several ways to implement the BFS traversal. Almost all of them share a common point, *i.e.* using the `queue` data structure to maintain the order of visits.

In brief, we push the nodes into the queue level by level. As a result, the priorities of visiting would roll

out from top to down and from left to right, due to the FIFO (First-In First-Out) principle of the queue data structure, *i.e.* the element that enters the queue first would exit first as well.

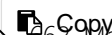


In the above graph, we show an example of BFS traversal on a full binary tree where we indicate the *global* order of visiting along with each node.

Algorithm

Here are a few steps to implement a solution with the BFS traversal.

- First of all, we create a `queue` data structure, which would be used to hold elements of tuple as `(node, col_index)`, where the `node` is the tree node and the `col_index` is the corresponding index that is assigned to the node based on our indexing schema. Also, we define a global variable called `max_width` which holds the maximal width that we've seen so far.
- Then we append the root node along with its index 0, to kick off the BFS traversal.
- The BFS traversal is basically an iteration over the elements of queue. We visit the nodes *level by level* until there are no more elements in the queue.
 - At the end of each level, we use the indices of the first and the last elements on the same level, in order to obtain the width of the level.
- At the end of BFS traversal, we then return the maximal width that we've seen over all levels.



Java

Python3

```

1  /**
2   * Definition for a binary tree node.
3   * public class TreeNode {
4   *     int val;
5   *     TreeNode left;
6   *     TreeNode right;
7   *     TreeNode(int x) { val = x; }
8   * }
9   */
10 class Solution {
11     public int widthOfBinaryTree(TreeNode root) {
12         if (root == null)
13             return 0;
14
15         // queue of elements [(node, col_index)]
16         LinkedList<Pair<TreeNode, Integer>> queue = new LinkedList<>();
17         Integer maxWidth = 0;
18
19         queue.addLast(new Pair<>(root, 0));
20         while (queue.size() > 0) {
21             Pair<TreeNode, Integer> head = queue.getFirst();
22
23             // iterate through the current level
24             Integer currLevelSize = queue.size();
25             Pair<TreeNode, Integer> elem = null;
26             for (int i = 0; i < currLevelSize; ++i) {
27                 elem = queue.removeFirst();
28                 TreeNode node = elem.getKey();
29                 if (node.left != null)
30                     queue.addLast(new Pair<>(node.left, 2 * elem.getValue()));
31                 if (node.right != null)
32                     queue.addLast(new Pair<>(node.right, 2 * elem.getValue() + 1));
33             }
34
35             // calculate the length of the current level,
36             // by comparing the first and last col_index.
37             maxWidth = Math.max(maxWidth, elem.getValue() - head.getValue() + 1);
38         }
39
40         return maxWidth;
41     }
42 }

```

Note: in the above implementation, we use the `size` of the queue as a delimiter to determine the boundary between each levels.

One could also use a specific dummy element as a marker to separate nodes of different levels in the queue.

Complexity Analysis

Let N be the total number of nodes in the input tree.

- Time Complexity: $\mathcal{O}(N)$

Articles > 662. Maximum Width of Binary Tree

- We visit each node once and only once. And at each visit, it takes a constant time to process.
- Space Complexity: $\mathcal{O}(N)$
 - We used a queue to maintain the nodes along with its indices, which is the main memory consumption of the algorithm.
 - Due to the nature of BFS, at any given moment, the queue holds no more than *two levels* of nodes. In the worst case, a level in a full binary tree contains at most half of the total nodes (i.e. $\frac{N}{2}$), i.e. this is also the level where the leaf nodes reside.
 - Hence, the overall space complexity of the algorithm is $\mathcal{O}(N)$.

Approach 2: DFS Traversal

Intuition

Although it is definitely more intuitive to implement a solution with BFS traversal, it is not impossible to do it with DFS.

It might sound twisted, but we don't need to visit the nodes strictly in the order of BFS. All we need is to compare the indices between the first and the last elements of the same level.

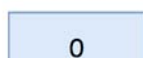
We could build a table that records the indices of nodes grouped by level. Then we could scan the indices level by level to obtain the **maximal** difference among them, which is also the width of the level.

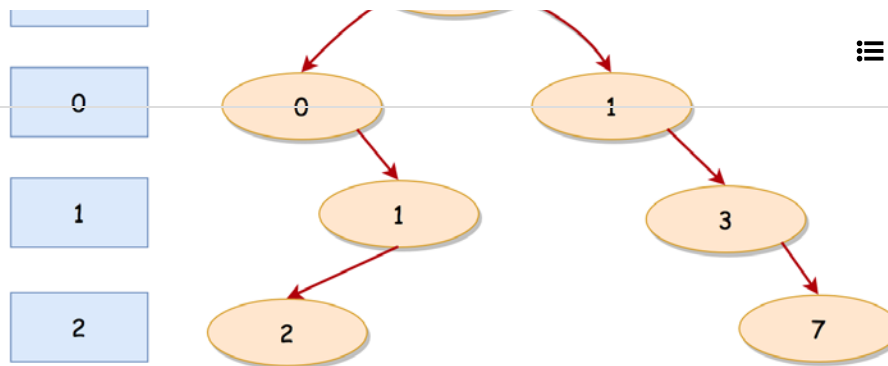
With the above idea, as we can see, any traversal will do, including the BFS and DFS.

Better yet, we don't need to keep the indices of the entire level, but the first and the last index.

We could use the table to keep **only** the index of the first element for each level, i.e. `depth -> first_col_index`, which we illustrate in the following graph.

first_col_index





Along with the traversal, we could compare the index of every node with the corresponding first index of its level (i.e. `first_col_index`).

Rather than keeping all the indices in the table, we save time and space by keeping only the index of the first element per level.

Algorithm

The tricky part is how we can obtain the index for the first element of each level.

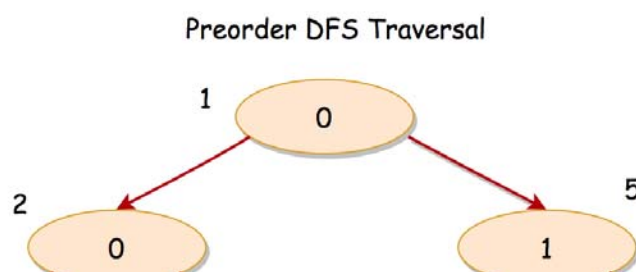
As we discussed before, we use a table with `depth` of the node as the key and the index of the first element for that depth (level) as the value.

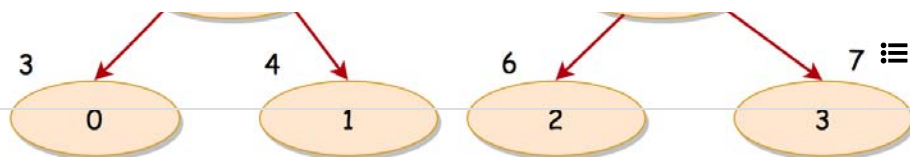
If we can make sure that we visit the first element of a level before the rest of elements on that level, we then can easily populate the table along with the traversal.

In fact, a DFS traversal can assure the above priority that we desire. Even better, it could be either *preorder*, *inorder* or *postorder* DFS traversal, as long as we **prioritize** the visit of the left child node over the right child node.

Although in principle DFS prioritizes depth over breadth, it could also ensure the *level-wise* priority. By visiting the left node before the right child node in DFS traversal, we can ensure that the nodes that lean more to the left got visited earlier.

We showcase a **preorder DFS** traversal, with an example in the following graph:





We label each node with a number that indicates the *global* order of visit. As one can see, the nodes at the same level do get visited from left to right. For instance, on the second level, the first node would be visited at the step 2, while the next node at the same level would be visited at the step 5.

We give some sample implementations of DFS in the following.

Java

Python3

Copy

```

1  class Solution {
2      private Integer maxWidth = 0;
3      private HashMap<Integer, Integer> firstColIndexTable;
4
5      protected void DFS(TreeNode node, Integer depth, Integer colIndex) {
6          if (node == null)
7              return;
8          // initialize the value, for the first seen colIndex per level
9          if (!firstColIndexTable.containsKey(depth)) {
10             firstColIndexTable.put(depth, colIndex);
11         }
12         Integer firstColIndex = firstColIndexTable.get(depth);
13
14         maxWidth = Math.max(this.maxWidth, colIndex - firstColIndex + 1);
15
16         // Preorder DFS. Note: it is important to put the priority on the left child
17         DFS(node.left, depth + 1, 2 * colIndex);
18         DFS(node.right, depth + 1, 2 * colIndex + 1);
19     }
20
21     public int widthOfBinaryTree(TreeNode root) {
22         // table contains the first col_index for each level
23         this.firstColIndexTable = new HashMap<Integer, Integer>();
24
25         // start from depth = 0, and colIndex = 0
26         DFS(root, 0, 0);
27
28         return this.maxWidth;
29     }
30 }

```

Complexity Analysis

Let N be the total number of nodes in the input tree.

- Time Complexity: $\mathcal{O}(N)$.
 - Similar to the BFS traversal, we visit each node once and only once in DFS traversal. And

each visit takes a constant time to process as well.

Articles > 662. Maximum Width of Binary Tree

- Space Complexity: $\mathcal{O}(N)$

- Unlike the BFS traversal, we used an additional table to keep the index for the first element per level. In the worst case where the tree is extremely skewed, there could be as many levels as the number of nodes. As a result, the space complexity of the table would be $\mathcal{O}(N)$.
- Since we implement DFS traversal with recursion which would incur some additional memory consumption in the function call stack, we need to take this into account for the space complexity.
- The consumption of function stack is proportional to the depth of recursion. Again, in the same worst case above, where the tree is extremely skewed, the depth of the recursion would be equal to the number of nodes in the tree. Therefore, the space complexity of the function stack would be $\mathcal{O}(N)$.
- To sum up, the overall space complexity of the algorithm is $\mathcal{O}(N) + \mathcal{O}(N) = \mathcal{O}(N)$.

Rate this article:

Previous (/articles/two-sum-less-than-k/)

Next (/articles/excel-sheet-column-number/)

Comments: 12

Sort By ▼



Type comment here... (Markdown is supported)

Preview

Post



(/andi_numbers)

ANDi_numbers (/andi_numbers) ★ 11 ⌚ a day ago

bad problem. C++ solution is correct but the integer overflows. Works when I implemented it in python3. Problems on leetcode shouldn't make it unfair for some languages.

10 ^ v | Share | Reply

SHOW 4 REPLIES



(/user6434v)

user6434v (/user6434v) ★ 22 ⌚ June 21, 2020 7:56 AM

DFS solution is nice!

2 ^ v | Share | Reply



denis19 (/denis19) ★ 19 ⌚ June 15, 2020 12:17 PM

Articles > 662. Maximum Width of Binary Tree

Good one, thanks for posting. Helped me understand level-order traversal a bit more.

(/denis19)

2 ^ v | Share | Reply



Amogh007 (/amogh007) ★ 18 ⌚ June 13, 2020 7:56 AM

Wow zero comments surprising ! Good solution with the dfs approach!

(/amogh007)

2 ^ v | Share | Reply



user0414A (/user0414a) ★ 19 ⌚ 20 hours ago

Short and simple Python BFS

(/user0414a)

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
```

Read More

1 ^ v | Share | Reply



fbl3 (/fbl3) ★ 26 ⌚ a day ago

great dfs solution

(/fbl3)

1 ^ v | Share | Reply



devaksoni (/devaksoni) ★ 1 ⌚ June 13, 2020 10:26 PM

Why do we need to multiple with 2 here ? Is it because of 2 nodes on that level.

(/devaksoni)

```
queue.addLast(new Pair<>(node.left, 2 * elem.getValue()
e()));
if (node.right != null)
```

Read More

1 ^ v | Share | Reply

SHOW 1 REPLY



foxhlchen (/foxhlchen) ★ 0 ⌚ 14 hours ago

Java BFS solution is ill for one of the test cases. elem.getValue(), head.getValue() have already been overflowed. the answer coincidentally becomes 1, the correct answer. So it passes the test case.

(/foxhlchen)

0 ^ v | Share | Reply



fymehta99 (/fymehta99) ★ 2 ⌚ 15 hours ago

In java, if using Queue interface instead of LinkedList it is giving null pointer exception, can someone explain why?

(/fymehta99)

0 ^ v | Share | Reply



(/flyseeksky)

flyseeksky (/flyseeksky) ★ 14 ⌚ a day ago

Articles > 662. Maximum Width of Binary Tree

Though not critical, but there is a small **error** in BFS solution: in a zero-based array representation of a full binary tree, the left child index is $2*n+1$ and the right child index is $2*n+2$.

0 Share Reply

SHOW 4 REPLIES

Copyright © 2020 LeetCode

[Help Center \(/support/\)](/support/) | [Terms \(/terms/\)](/terms/) | [Privacy \(/privacy/\)](/privacy/)[United States \(/region/\)](/region/)