331. Verify Preorder Serialization of a Binary Tree

Aug. 3, 2019 | 4.7K views

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as `#`.

```
     _9_
    /   \
   3     2
  / \   / \
 4   1  #  6
/ \ / \   / \
# # # #   # #
```

For example, the above binary tree can be serialized to the string `"9,3,4,#,#,1,#,#,2,#,6,#,#"`, where `#` represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character `'#'` representing `null` pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as `"1,,3"`.

Example 1:

```
Input: "9,3,4,#,#,1,#,#,2,#,6,#,#"
Output: true
```

Example 2:

```
Input: "1,#"
Output: false
```

Example 3:

```
Input: "9,#,#,1"
Output: false
```

Solution

Approach 1: Iteration

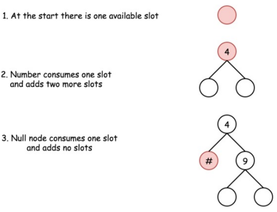
Intuition

Let's start from the simplest but not optimal solution to discuss the idea.

Binary tree could be considered as a number of slots to fulfill. At the start there is just one slot available for a number or null node. Both number and null node take one slot to be placed. For the null node the story ends up here, whereas the number will add into the tree two slots for the child nodes. Each child node could be, again, a number or a null.

> The idea is straightforward : take the nodes one by one from preorder traversal, and compute the number of available slots. If at the end all available slots are used up, the preorder traversal represents the valid serialization.

- In the beginning there is one available slot.
- Each number or null consumes one slot.
- Null node adds no slots, whereas each number adds two slots for the child nodes.



Algorithm

- Initiate the number of available slots: `slots = 1`.
- Split preorder traversal by comma, and iterate over the resulting array. At each step :
  - Both a number or a null node take one slot : `slots = slot - 1`.
  - If the number of available slots is negative, the preorder traversal is invalid, return False.
  - Non-empty node `node != '#'` creates two more available slots: `slots = slots + 2`.
- Preorder traversal is valid if all available slots are used up : return `slots == 0`.

Implementation

Java | Python

Copy

```python
class Solution:
    def isValidSerialization(self, preorder: str) -> bool:
        # number of available slots
        slots = 1

        for node in preorder.split(','):
            # one node takes one slot
            slots -= 1

            # no more slots available
            if slots < 0:
                return False

            # non-empty node creates two children slots
            if node != '#':
                slots += 2

        # all slots should be used up
        return slots == 0
```

Complexity Analysis

- Time complexity : $O(N)$ to iterate over the string of length N.
- Space complexity : $O(N)$ to keep split array in memory.

Approach 2: One pass

Intuition

Approach 1 uses $O(N)$ space to keep split array in memory, and for sure that should be optimised. The idea is to iterate over the string itself and not over the array of nodes.

During the iteration, one has to update the number of available slots at each comma character. First, one should decrease the number of slots by one, because both empty and non-empty node take one slot. Second, if the node is a non-empty one, i.e. the character just before the comma is not equal to `#`, one should add two more slots for the child nodes.

The last node should be considered separately, since there is no comma after it.

Algorithm

- Initiate the number of available slots: `slots = 1`.
- Iterate over the string. At each comma :
  - Both a number or a null node take one slot : `slots = slot - 1`.
  - If the number of available slots is negative, the preorder traversal is invalid, return False.
  - Non-empty node, detected by non-`#` character before comma, creates two more available slots: `slots = slots + 2`.
- The last node should be considered separately, since there is no comma after it.
- Preorder traversal is valid if all available slots are used up : return `slots == 0`.

Implementation

Java | Python

Copy

```python
class Solution:
    def isValidSerialization(self, preorder: str) -> bool:
        # number of available slots
        slots = 1

        prev = None  # previous character
        for ch in preorder:
            if ch == ',':
                # one node takes one slot
                slots -= 1

                # no more slots available
                if slots < 0:
                    return False

                # non-empty node creates two children slots
                if prev != '#':
                    slots += 2
            prev = ch

        # the last node
        slots = slots - 1 if ch != '#' else slots - 1
        # all slots should be used up
        return slots == 0
```

Complexity Analysis

- Time complexity : $O(N)$ to iterate over the string of length N.
- Space complexity : $O(1)$, it's a constant space solution.

Rate this article:

Previous Next

Comments: 2

Sort By

Type comment here... (Markdown is supported)

Preview Post

Ast15 16 November 19, 2019 1:12 AM

Those solutions have nothing to do with stack.

13 Share Reply

Merciless 549 February 7, 2020 11:35 PM

My stack solution:

```
class Solution:
    def isValidSerialization(self, preorder: str) -> bool:
        pre = preorder.split(",")
```

Read More

0 Share Reply