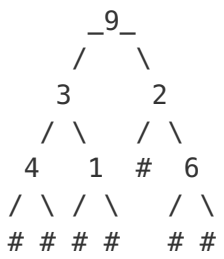
Verify Preorder Serialization of a Binary Tree

One way to serialize a binary tree is to use pre-oder 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 `#`.



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

Example 2:
`"1,#"`
Return `false`

Example 3:
`"9,#,#,1"`
Return `false`

Credits:
Special thanks to @dietpepsi for adding this problem and creating all test cases.

Solution 1

Some used stack. Some used the depth of a stack. Here I use a different perspective. In a binary tree, if we consider null as leaves, then

- all non-null node provides 2 outdegree and 1 indegree (2 children and 1 parent), except root
- all null node provides 0 outdegree and 1 indegree (0 child and 1 parent).

Suppose we try to build this tree. During building, we record the difference between out degree and in degree `diff` = `outdegree - indegree`. When the next node comes, we then decrease `diff` by 1, because the node provides an in degree. If the node is not `null`, we increase diff by `2`, because it provides two out degrees. If a serialization is correct, diff should never be negative and diff will be zero when finished.

```
public boolean isValidSerialization(String preorder) {
    String[] nodes = preorder.split(",");
    int diff = 1;
    for (String node: nodes) {
        if (--diff < 0) return false;
        if (!node.equals("#")) diff += 2;
    }
    return diff == 0;
}
```

written by dietpepsi original link here

Solution 2

This is very simple problem if you use stacks. The key here is, when you see two consecutive "#" characters on stack, pop both of them and replace the topmost element on the stack with "#". For example,

preorder = 1,2,3,#,#,#,#

Pass 1: stack = [1]

Pass 2: stack = [1,2]

Pass 3: stack = [1,2,3]

Pass 4: stack = [1,2,3,#]

Pass 5: stack = [1,2,3,#,#] -> two #s on top so pop them and replace top with #. -> stack = [1,2,#]

Pass 6: stack = [1,2,#,#] -> two #s on top so pop them and replace top with #. -> stack = [1,#]

Pass 7: stack = [1,#,#] -> two #s on top so pop them and replace top with #. -> stack = [#]

If there is only one # on stack at the end of the string then return True else return False.

Here is the code for that,

```
class Solution(object):
def isValidSerialization(self, preorder):
    """
    :type preorder: str
    :rtype: bool
    """
    stack = []
    top = -1
    preorder = preorder.split(',')
    for s in preorder:
        stack.append(s)
        top += 1
        while(self.endsWithTwoHashes(stack,top)):
            h = stack.pop()
            top -= 1
            h = stack.pop()
            top -= 1
            if top < 0:
                return False
            h = stack.pop()
            stack.append('#')
        #print stack
    if len(stack) == 1:
        if stack[0] == '#':
            return True
    return False

def endsWithTwoHashes(self,stack,top):
    if top<1:
        return False
    if stack[top]=='#' and stack[top-1]=='#':
        return True
    return False
```

written by harshaneel original link here

Solution 3

We just need to remember how many empty slots we have during the process.

Initially we have one (for the root).

for each node we check if we still have empty slots to put it in.

- a null node occupies one slot.
- a non-null node occupies one slot before he creates two more. the net gain is one.

```
class Solution(object):
    def isValidSerialization(self, preorder):
        """
        :type preorder: str
        :rtype: bool
        """
        # remember how many empty slots we have
        # non-null nodes occupy one slot but create two new slots
        # null nodes occupy one slot

        p = preorder.split(',')

        #initially we have one empty slot to put the root in it
        slot = 1
        for node in p:

            # no empty slot to put the current node
            if slot == 0:
                return False

            # a null node?
            if node == '#':
                # ocuppy slot
                slot -= 1
            else:
                # create new slot
                slot += 1

        #we don't allow empty slots at the end
        return slot==0
```

written by hohomi original link here