













Verify Preorder Serialization of a Binary Tree / 7 lines Easy Java Solution

7 lines Easy Java Solution





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; **Julie Wang** Reputation: * -1 Thank you for providing this very neat solution!





gaooo

Reputation: # 0



Thank you for such a neat solution!

I have a follow-up question on this: does this also work if the String given is an inorder of postorder traversal? Thank you for your time!







james.wang.cccgmail.com

Reputation: ** 8



Thanks for such a neat solution dietpepsi. I spent sometime trying to generalize this solution to InOrder and PostOrder traversal.

- 1. I think the condition that diff == 0 at the end holds for all three, first of all
- 2. Through observation, I concluded the following, which I am NOT certain if correct and complete or not.

So will really appreciate it if someone can offer some better idea how to generalize this approach to the InOrder and PostOrder:

```
public boolean isValidSerialization PostOrder(String postorder) {
        String[] nodes = postorder.split(",");
        int diff = 1;
        for (String node : nodes) {
                diff--;
                if (!node.equals("#")) diff += 2;
                // Post-Order traversal fail criteria
               if (diff > 0) return false;
        return diff == 0;
public boolean isValidSerialization InOrder(String inorder) {
        String[] nodes = inorder.split(",");
        int diff = 1;
        for (String node : nodes) {
                diff--;
                if (!node.equals("#")) diff += 2;
               // In-Order traversal fail criteria
               if (diff > 1) return false;
        return diff == 0;
```







generalize the idea to postorder is easy. Just reverse traversal the strings, it is the same.

But the tricky one is inorder. After all inorder is not able to deserialize even if the serialization is correct.

For example, given "#,1,#,2,#" • It can either be a tree rooted with 1 or a tree rooted with 2. Both are valid. Thus I really doubt inorder can verify easily anyhow.





lixx2100 Reputation: **★** 656



Nice solution. My solution is quite similar to yours.

If we treat null 's as leaves, then the binary tree will always be **full**. A full binary tree has a good property that # of leaves = # of nonleaves + 1. Since we are given a pre-order serialization, we just need to find the **shortest** prefix of the serialization sequence satisfying the property above. If such prefix does not exist, then the serialization is definitely invalid; otherwise, the serialization is valid if and only if the prefix is the entire sequence.

```
// Java Code
public boolean isValidSerialization(String preorder) {
    int nonleaves = 0, leaves = 0, i = 0;
    String[] nodes = preorder.split(",");
    for (i=0; i<nodes.length && nonleaves + 1 != leaves; i++)</pre>
        if (nodes[i].equals("#")) leaves++;
        else nonleaves++;
    return nonleaves + 1 == leaves && i == nodes.length;
```





ElementNotFoundException

Reputation: ★ 1098



Very nice solution. Let's assume

If a serialization is correct, diff should never be negative and diff will be zero when finished

is sounded, how do we argue the opposite is also true, i.e.,

if diff never becomes negative and diff is zero at the end, then the serialization is correct

That is, we know a -> b is true, but it does not necessarily mean b -> a is also true. The method I used also has this kind of unresolved question (if we are strict to ourselves) and I cannot explain it well yet.





Will

Reputation: ★ 25



"If a serialization is correct, diff should never be negative and diff will be zero when finished." How did you get this conclusion? Sorry I'm not good at graph and indegree/outdegree thing. Thanks!





RainbowSecret | • @lixx2100



Reputation: ★ 722



Only count the number of the null-node and the non-null node can ensure the tree is OK ????? Is this condition Complete ???



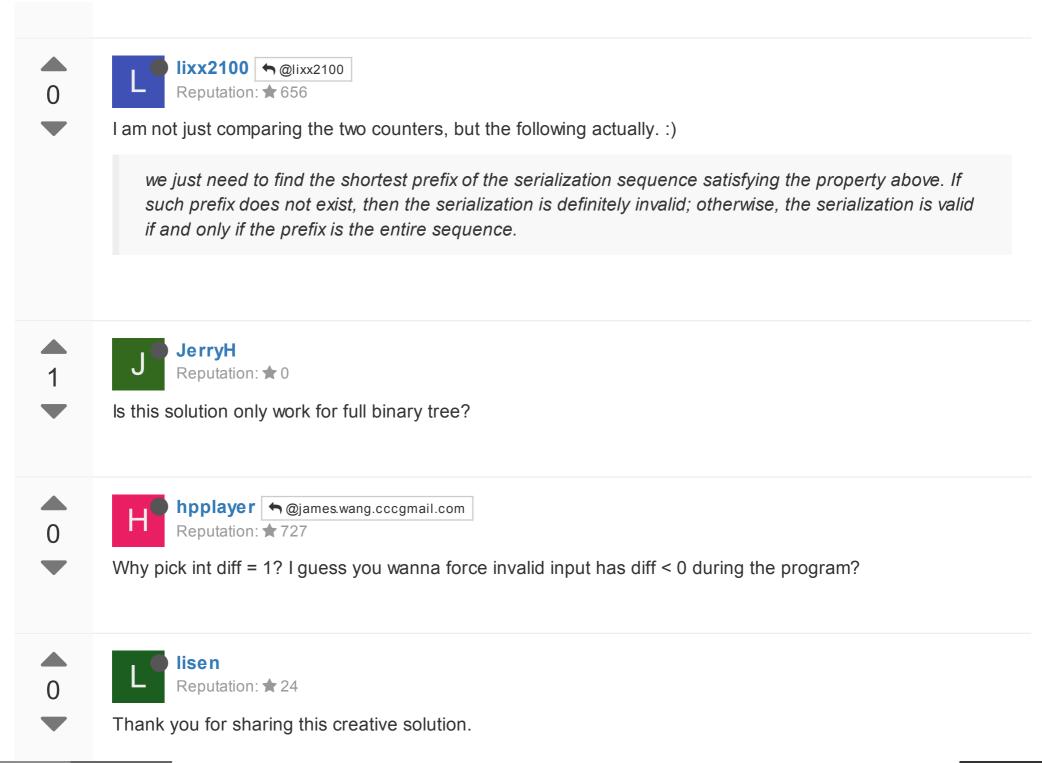


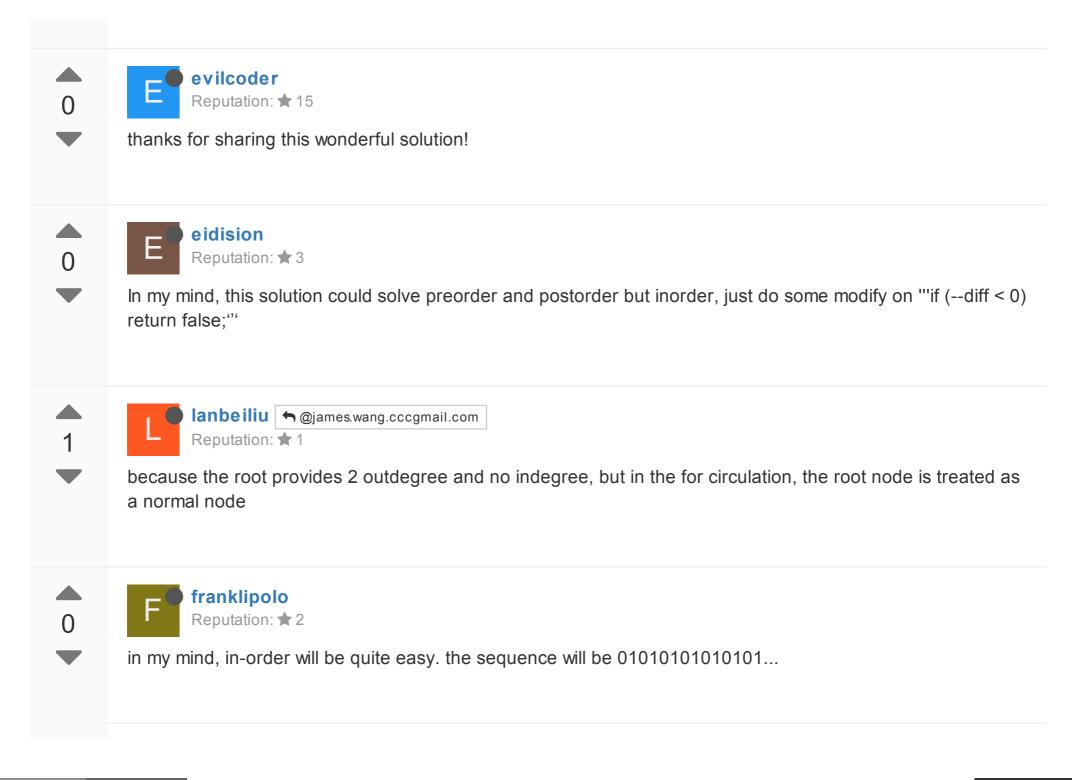
RainbowSecret

Reputation: ★ 722

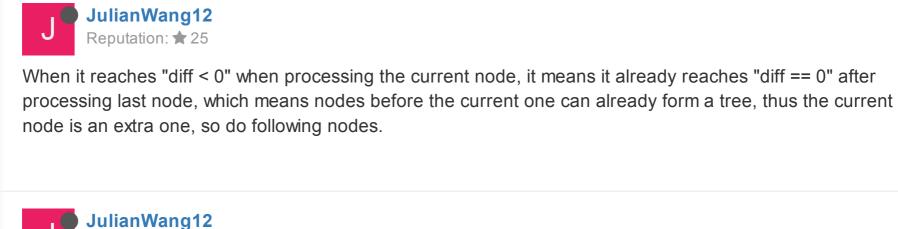


I am wondering how you think of this idea?









I think you probably made a logic error, and so does @dietpepsi in his explanation. If we use this solution here, we must assume the following is true, which I think is related to preorder traversal.

if diff never becomes negative and diff is zero at the end, then the serialization is correct

SOLUTION-SHARING 15815 EASIEST 238 JAVA 7710

Reputation: ★ 25

12878 44 **POSTS VIEWS**

Log in to reply

0