LeetCode 297

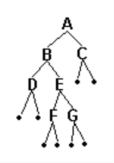
此问题的原型是根据扩展二叉树的前序遍历结果，来复原二叉树。解决此问题需要一些铺垫就是一本通1340-扩展二叉树。

1340：【例3-5】扩展二叉树

时间限制: 1000 ms　　内存限制: 65536 KB
提交数:9649　通过数: 7043

【题目描述】

由于先序、中序和后序序列中的任一个都不能唯一确定一棵二叉树，所以对二叉树做如下处理，将二叉树的空结点用·补齐，如图所示。我们把这样处理后的二叉树称为原二叉树的扩展二叉树，扩展二叉树的先序和后序序列能唯一确定其二叉树。



现给出扩展二叉树的先序序列，要求输出其中序和后序序列。

【输入】

扩展二叉树的先序序列。

【输出】

输出其中序和后序序列。

【输入样例】

```
ABD..EF..G..C..
```

扩展二叉树是把空节点用符号`.`来进行表示，这样可以通过这个特殊符号来作为递归的终止条件。因为上面这个例题它每个节点只是一个单一字母，所以只需要用`.`来表示空节点即可。但是leetcode 297不同，每个节点是一个数字，比如`123`，如果没有任何处理，根本无法判断是`1`和`23`还是`12`和`3`。所以一种解决办法就是增加分隔符`#`，那么两个`#`之间的就是节点的数字。

序列化过程：前序遍历整个二叉树，每个节点之间用`#`进行分隔。

反序列化过程：第一个点肯定是根节点，然后构建左子树，然后构建右子树。

```
1  /**
2   * Definition for a binary tree node.
```

```
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Codec {

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder res = new StringBuilder();
        if (root == null) return res.toString();

        preorderTraversal(root, res);

        return res.toString();
    }

    private void preorderTraversal(TreeNode root, StringBuilder
res) {
        if (root == null) {
            res.append(".#");
            return;
        }

        res.append(root.val).append("#");
        preorderTraversal(root.left, res);
        preorderTraversal(root.right, res);
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if (data.isEmpty()) return null;

```

```java
        int[] pos = new int[1]; // Use array to maintain reference
        pos[0] = 0;
        return build(data, pos);
    }

    private TreeNode build(String data, int[] pos) {
        int nextPos = data.indexOf("#", pos[0]);
        /**
            if the tree is null, the serialization result is a null string
            so no `#` in the string.
         */
        if (nextPos == -1) {
            return null;
        }

        // the root node
        String tmp = data.substring(pos[0], nextPos);
        pos[0] = nextPos + 1;
        // if the root node is `.`, it means it is null
        if (tmp.charAt(0) == '.') return null;
        else {
            TreeNode root = new TreeNode(Integer.parseInt(tmp));
            root.left = build(data, pos);
            // after build the left sub tree, the pos[0] has been updated
            root.right = build(data, pos);
            return root;
        }
    }
}

// Your Codec object will be instantiated and called as such:
```

```
68  // Codec codec = new Codec();
69  // codec.deserialize(codec.serialize(root));
```