Find all possible interpretations of an array of digits

Consider a coding system for alphabets to integers where ‘a’ is represented as 1, ‘b’ as 2, .. ‘z’ as 26. Given an array of digits (1 to 9) as input, write a function that prints all valid interpretations of input array.

Examples

```
Input: {1, 1}
Output: ("aa", 'k")
[2 interpretations: aa(1, 1), k(11)]

Input: {1, 2, 1}
Output: ("aba", "au", "la")
[3 interpretations: aba(1,2,1), au(1,21), la(12,1)]

Input: {9, 1, 8}
Output: {"iah", "ir"}
[2 interpretations: iah(9,1,8), ir(9,18)]
```

Please note we cannot change order of array. That means {1,2,1} cannot become {2,1,1}
On first look it looks like a problem of permutation/combination. But on closer look you will figure out that this is an interesting tree problem.
The idea here is string can take at-most two paths:
1. Proces single digit
2. Process two digits
That means we can use binary tree here. Processing with single digit will be left child and two digits will be right child. If value two digits is greater than 26 then our right child will be null as we don’t have alphabet for greater than 26.

Let’s understand with an example .Array a = {1,2,1}. Below diagram shows that how our tree grows.

```
                           “” {1,2,1}            Codes used in tree
                       /             \               "a" --> 1
                      /               \              "b" --> 2
                  "a"{2,1}            "l"{1}         "l" --> 12
                 /        \          /     \
                /          \        /       \
            "ab"{1}        "au"    "la"      null
             /    \
            /      \
         "aba"      null
```

Braces {} contain array still pending for processing. Note that with every level, our array size decreases. If you will see carefully, it is not hard to find that tree height is always n (array size)
How to print all strings (interpretations)? Output strings are leaf node of tree. i.e for {1,2,1}, output is {aba au la}.
We can conclude that there are mainly two steps to print all interpretations of given integer array.

Step 1: Create a binary tree with all possible interpretations in leaf nodes.

Step 2: Print all leaf nodes from the binary tree created in step 1.

Following is Java implementation of above algorithm.

```
// A Java program to print all interpretations of an integer array
import java.util.Arrays;

// A Binary Tree node
class Node {

    String dataString;
    Node left;
    Node right;

    Node(String dataString) {
```

```java
    Node(String dataString) {
        this.dataString = dataString;
        //Be default left and right child are null.
    }

    public String getDataString() {
        return dataString;
    }
}

public class arrayToAllInterpretations {

    // Method to create a binary tree which stores all interpretations
    // of arr[] in lead nodes
    public static Node createTree(int data, String pString, int[] arr) {

        // Invalid input as alphabets maps from 1 to 26
        if (data > 26)
            return null;

        // Parent String + String for this node
        String dataToStr = pString + alphabet[data];

        Node root = new Node(dataToStr);

        // if arr.length is 0 means we are done
        if (arr.length != 0) {
            data = arr[0];

            // new array will be from index 1 to end as we are consuming
            // first index with this node
            int newArr[] = Arrays.copyOfRange(arr, 1, arr.length);

            // left child
            root.left = createTree(data, dataToStr, newArr);

            // right child will be null if size of array is 0 or 1
            if (arr.length > 1) {

                data = arr[0] * 10 + arr[1];

                // new array will be from index 2 to end as we
                // are consuming first two index with this node
                newArr = Arrays.copyOfRange(arr, 2, arr.length);

                root.right = createTree(data, dataToStr, newArr);
            }
        }
        return root;
    }

    // To print out leaf nodes
    public static void printleaf(Node root) {
        if (root == null)
            return;

        if (root.left == null && root.right == null)
            System.out.print(root.getDataString() + "  ");

        printleaf(root.left);
        printleaf(root.right);
    }

    // The main function that prints all interpretations of array
    static void printAllInterpretations(int[] arr) {

        // Step 1: Create Tree
        Node root = createTree(0, "", arr);

        // Step 2: Print Leaf nodes
        printleaf(root);

        System.out.println();  // Print new line
    }

    // For simplicity I am taking it as string array. Char Array will save space
    private static final String[] alphabet = {"", "a", "b", "c", "d", "e",
        "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r",
        "s", "t", "u", "v", "w", "x", "v", "z"};

    // Driver method to test above methods
    public static void main(String args[]) {

        // aacd(1,1,3,4) amd(1,13,4) kcd(11,3,4)
        // Note : 1,1,34 is not valid as we don't have values corresponding
        // to 34 in alphabet
        int[] arr = {1, 1, 3, 4};
        printAllInterpretations(arr);

        // aaa(1,1,1) ak(1,11) ka(11,1)
        int[] arr2 = {1, 1, 1};
        printAllInterpretations(arr2);

        // bf(2,6) z(26)
        int[] arr3 = {2, 6};
        printAllInterpretations(arr3);

        // ab(1,2), l(12)
        int[] arr4 = {1, 2};
        printAllInterpretations(arr4);

        // a(1,0} j(10)
        int[] arr5 = {1, 0};
        printAllInterpretations(arr5);

        // "" empty string output as array is empty
        int[] arr6 = {};
        printAllInterpretations(arr6);

        // abba abu ava lba lu
        int[] arr7 = {1, 2, 2, 1};
        printAllInterpretations(arr7);
    }
}
```

Output:

```
aacd  amd  kcd
aaa  ak  ka
bf  z
ab  l
a  j

abba  abu  ava  lba  lu
```

Exercise:

1. What is the time complexity of this solution? [Hint : size of tree + finding leaf nodes]
2. Can we store leaf nodes at the time of tree creation so that no need to run loop again for leaf node fetching?
3. How can we reduce extra space?