Custom Tree Problem

You are given a set of links, e.g.

```
a ---> b
b ---> c
b ---> d
a ---> e
```

Print the tree that would form when each pair of these links that has the same character as start and end point is joined together. You have to maintain fidelity w.r.t. the height of nodes, i.e. nodes at height n from root should be printed at same row or column. For set of links given above, tree printed should be –

```
-->a
   |-->b
   |   |-->c
   |   |-->d
   |-->e
```

Note that these links need not form a single tree; they could form, ahem, a forest. Consider the following links

```
a ---> b
a ---> g
b ---> c
c ---> d
d ---> e
c ---> f
z ---> y
y ---> x
x ---> w
```

The output would be following forest.

```
-->a
   |-->b
   |   |-->c
   |   |   |-->d
   |   |   |   |-->e
   |   |   |-->f
   |-->g

-->z
   |-->y
   |   |-->x
   |   |   |-->w
```

You can assume that given links can form a tree or forest of trees only, and there are no duplicates among links.

Solution: The idea is to maintain two arrays, one array for tree nodes and other for trees themselves (we call this array forest). An element of the node array contains the TreeNode object that corresponds to respective character. An element of the forest array contains Tree object that corresponds to respective root of tree.

It should be obvious that the crucial part is creating the forest here, once it is created, printing it out in required format is straightforward. To create the forest, following procedure is used –

```
Do following for each input link,
1. If start of link is not present in node array
     Create TreeNode objects for start character
     Add entries of start in both arrays.
2. If end of link is not present in node array
     Create TreeNode objects for start character
     Add entry of end in node array.
3. If end of link is present in node array.
     If end of link is present in forest array, then remove it
     from there.
4. Add an edge (in tree) between start and end nodes of link.
```

It should be clear that this procedure runs in linear time in number of nodes as well as of links – it makes only one pass over the links. It also requires linear space in terms of alphabet size.

Following is Java implementation of above algorithm. In the following implementation characters are assumed to be only lower case characters from 'a' to 'z'.

```
// Java program to create a custom tree from a given set of links.

// The main class that represents tree and has main method
public class Tree {

    private TreeNode root;

    /* Returns an array of trees from links input. Links are assumed to
       be Strings of the form "<s> <e>" where <s> and <e> are starting
       and ending points for the link. The returned array is of size 26
       and has non-null values at indexes corresponding to roots of trees
       in output */
    public Tree[] buildFromLinks(String [] links) {

        // Create two arrays for nodes and forest
        TreeNode[] nodes = new TreeNode[26];
        Tree[] forest = new Tree[26];

        // Process each link
        for (String link : links) {

            // Find the two ends of current link
            String[] ends = link.split(" ");
            int start = (int) (ends[0].charAt(0) - 'a'); // Start node
            int end   = (int) (ends[1].charAt(0) - 'a'); // End node

            // If start of link not seen before, add it two both arrays
            if (nodes[start] == null)
            {
                nodes[start] = new TreeNode((char) (start + 'a'));

                // Note that it may be removed later when this character is
                // last character of a link. For example, let we first see
                // a--->b, then c--->a.  We first add 'a' to array of trees
                // and when we see link c--->a, we remove it from trees array.
                forest[start] = new Tree(nodes[start]);
            }

            // If end of link is not seen before, add it to the nodes array
            if (nodes[end] == null)
                nodes[end] = new TreeNode((char) (end + 'a'));

            // If end of link is seen before, remove it from forest if
            // it exists there.
            else forest[end] = null;

            // Establish Parent-Child Relationship between Start and End
            nodes[start].addChild(nodes[end], end);
        }
        return forest;
    }

    // Constructor
```

```
    public Tree(TreeNode root) { this.root = root;  }

    public static void printForest(String[] links)
    {
        Tree t = new Tree(new TreeNode('\0'));
        for (Tree t1 : t.buildFromLinks(links)) {
           if (t1 != null)
           {
              t1.root.printTreeIdented("");
              System.out.println("");
           }
        }
    }

    // Driver method to test
    public static void main(String[] args) {
        String [] links1 = {"a b", "b c", "b d", "a e"};
        System.out.println("------------ Forest 1 ----------------");
        printForest(links1);

        String [] links2 = {"a b", "a g", "b c", "c d", "d e", "c f",
                            "z y", "y x", "x w"};
        System.out.println("------------ Forest 2 ----------------");
        printForest(links2);
    }
}

// Class to represent a tree node
class TreeNode {
    TreeNode []children;
    char c;

    // Adds a child 'n' to this node
    public void addChild(TreeNode n, int index) { this.children[index] = n;}

    // Constructor
    public TreeNode(char c) { this.c = c;  this.children = new TreeNode[26];}

    // Recursive method to print indented tree rooted with this node.
    public void printTreeIdented(String indent) {
            System.out.println(indent + "-->" + c);
            for (TreeNode child : children) {
              if (child != null)
                 child.printTreeIdented(indent + "   |");
            }
    }
}
```

Output:

```
------------ Forest 1 ----------------
-->a
   |-->b
   |   |-->c
   |   |-->d
   |-->e

------------ Forest 2 ----------------
-->a
   |-->b
   |   |-->c
   |   |   |-->d
   |   |   |   |-->e
   |   |   |-->f
   |-->g

-->z
   |-->y
   |   |-->x
   |   |   |-->w
```

Exercise:

In the above implementation, endpoints of input links are assumed to be from set of only 26 characters. Extend the implementation where endpoints are strings of any length.