CONVERTING A NON-BINARY TREE TO BINARY FORM

Implementing a non-binary tree with a corresponding linked structure has its problems. Each node of the non-binary tree requires as many links as the maximum number of children in the tree. Most of those links, however, go unused. For example, the following tree requires a node class with three link fields to accommodate the children of A and D:

![Tree Diagram]

A possible class declaration is:

```java
class TreeNode
{
    public char key; // key value
    public TreeNode child1; // link to first child
    public TreeNode child2; // link to second child
    public TreeNode child3; // link to third child

    public TreeNode(char k, TreeNode c1, TreeNode c2, TreeNode c3)
    {
        key = k; child1 = c1;
        child2 = c2; child3 = c3;
    }
}
```

The code to create the above tree is:

```java
TreeNode t1 = new TreeNode('E', null, null, null);
TreeNode t2 = new TreeNode('F', null, null, null);
TreeNode t3 = new TreeNode('C', t1, t2, null);
t1 = new TreeNode('G', null, null, null);
t2 = new TreeNode('H', null, null, null);
t4 = new TreeNode('I', null, null, null);
TreeNode t5 = new TreeNode('D', t1, t2, t4);
t1 = new TreeNode('B', null, null, null);
TreeNode root = new TreeNode('A', t1, t3, t5);
```
And the tree in memory appears as:

The tree has 9 nodes with 3 link fields each, totaling 27 link fields. Of those only 8 of them, or 30%, actually point to something.

A binary tree is much more space efficient because each node needs at most two links. Furthermore, you can store any non-binary tree in the two-link binary tree structure by following this conversion formula:

Given a non-binary tree, for each node G within it, create a binary tree node B, set B’s left child to G’s leftmost child and B’s right child to G’s right sibling.

For example, the general tree given above, stored in a binary tree structure, appears as: