PSUDOCODE FOR DRACULA BINARY TREE

```
class BinaryTreeNode
    String word
    int instanceCount
    BinaryTreeNode left
    BinaryTreeNode right

    constructor BinaryTreeNode(word)
        this.word = word
        this.instanceCount = 1
        this.left = null
        this.right = null

class DraculaBinaryTree
    BinaryTreeNode root

    constructor DraculaBinaryTree()
        root = null

    method insert(String word)
        root = insertRec(root, word)

    method insertRec(BinaryTreeNode root, String word)
        if root is null
            return new BinaryTreeNode(word)

        comparison = compare word with root.word

        if comparison < 0
            root.left = insertRec(root.left, word)
        else if comparison > 0
            root.right = insertRec(root.right, word)
        else
            increment instanceCount

        return root

    method depth() returns int
        return depthRec(root)

    method depthRec(BinaryTreeNode node) returns int
        if node is null
            return 0
        leftDepth = depthRec(node.left)
        rightDepth = depthRec(node.right)
        return max(leftDepth, rightDepth) + 1

    method uniqueWordCount() returns int
        return uniqueWordCountRec(root)

    method uniqueWordCountRec(BinaryTreeNode node) returns int
```

```
        if node is null
            return 0
        return uniqueWordCountRec(node.left) + uniqueWordCountRec(node.right) + 1

    method rootWord() returns String
        if root is not null
            return root.word
        return "Tree is empty"

    method mostFrequentWord() returns String
        return mostFrequentWordRec(root).word

    method mostFrequentWordRec(BinaryTreeNode node) returns BinaryTreeNode
        if node is null
            return null
        leftMostFrequent = mostFrequentWordRec(node.left)
        rightMostFrequent = mostFrequentWordRec(node.right)

        maxNode = node
        if leftMostFrequent is not null and leftMostFrequent.instanceCount >
maxNode.instanceCount
            maxNode = leftMostFrequent
        if rightMostFrequent is not null and rightMostFrequent.instanceCount >
maxNode.instanceCount
            maxNode = rightMostFrequent
        return maxNode

    method deepestLeaves() returns String
        return deepestLeavesRec(root)

    method deepestLeavesRec(BinaryTreeNode node) returns String
        if node is null
            return ""
        if node.left is null and node.right is null
            return node.word + " "
        leftLeaves = deepestLeavesRec(node.left)
        rightLeaves = deepestLeavesRec(node.right)
        return leftLeaves + rightLeaves

    method totalWords() returns int
        return totalWordsRec(root)

    method totalWordsRec(BinaryTreeNode node) returns int
        if node is null
            return 0
        return node.instanceCount + totalWordsRec(node.left) + totalWordsRec(node.right)

    method firstWordPreOrder() returns String
        return firstWordPreOrderRec(root)

    method firstWordPreOrderRec(BinaryTreeNode node) returns String
        if node is null
            return ""
        return node.word
```

```
    method firstWordPostOrder() returns String
        return firstWordPostOrderRec(root)

    method firstWordPostOrderRec(BinaryTreeNode node) returns String
        if node is null
            return ""
        return firstWordPostOrderRec(node.left) + firstWordPostOrderRec(node.right) + node.word

    method firstWordInOrder() returns String
        return firstWordInOrderRec(root)

    method firstWordInOrderRec(BinaryTreeNode node) returns String
        if node is null
            return ""
        return firstWordInOrderRec(node.left) + node.word + firstWordInOrderRec(node.right)

    method findInstanceCount(String word) returns int
        return findInstanceCountRec(root, word)

    method findInstanceCountRec(BinaryTreeNode node, String word) returns int
        if node is null
            return 0
        comparison = compare word with node.word
        if comparison < 0
            return findInstanceCountRec(node.left, word)
        else if comparison > 0
            return findInstanceCountRec(node.right, word)
        else
            return node.instanceCount

    method main(String[] args)
        tree = new DraculaBinaryTree()

        try
            filepath = "/Users/egj/Desktop/PA#4/Dracula.txt"
            reader = new BufferedReader(new FileReader(filepath))

            while line = reader.readLine()
                words = split line into words, remove non-alphabetic characters, and convert to
lowercase
                for each word in words
                    if word is not empty
                        tree.insert(word)

            reader.close()

            // Queries and output
            print "Text contains " + tree.totalWords() + " Total words."

            queryWords = ["transylvania", "harker", "renfield", "vampire", "expostulate"]
            for queryWord in queryWords
                instanceCount = tree.findInstanceCount(queryWord)
                print queryWord + " occurs : " + instanceCount + " times"
```

```
print "Tree is : " + tree.depth() + " nodes deep"
print "Tree contains : " + tree.uniqueWordCount() + "
```