PSUDOCODE FOR BINARY TREE

```
class BinaryTreeNode:
    string word
    integer 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(word):
        root = insertRec(root, word)

    method insertRec(node, word):
        if node == null:
            return new BinaryTreeNode(word)

        comparison = compare(word, node.word)

        if comparison < 0:
            node.left = insertRec(node.left, word)
        else if comparison > 0:
            node.right = insertRec(node.right, word)
        else:
            node.instanceCount++

        return node

    method depth():
        return depthRec(root)

    method depthRec(node):
        if node == null:
            return 0

        leftDepth = depthRec(node.left)
        rightDepth = depthRec(node.right)

        return max(leftDepth, rightDepth) + 1

    method uniqueWordCount():
        return uniqueWordCountRec(root)

    method uniqueWordCountRec(node):
        if node == null:
            return 0

        return uniqueWordCountRec(node.left) + uniqueWordCountRec(node.right) + 1

    method rootWord():
        if root != null:
            return root.word
        return "Tree is empty"

    method mostFrequentWord():
        return mostFrequentWordRec(root).word

    method mostFrequentWordRec(node):
        if node == null:
            return null

        leftMostFrequent = mostFrequentWordRec(node.left)
        rightMostFrequent = mostFrequentWordRec(node.right)

        maxNode = node
        if leftMostFrequent != null and leftMostFrequent.instanceCount >
maxNode.instanceCount:
            maxNode = leftMostFrequent
        if rightMostFrequent != null and rightMostFrequent.instanceCount >
maxNode.instanceCount:
            maxNode = rightMostFrequent

        return maxNode

    // Other methods follow the same pattern as in code...

    method main():
        tree = new DraculaBinaryTree()

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

            string line
            while (line = reader.readLine()) != null:
                words = splitAndCleanWords(line)
                for word in words:
                    if word is not empty:
                        tree.insert(word)

            reader.close()

            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() + " distinct words"
            print "Word at root is : " + tree.rootWord()
            print "Deepest word(s) is/are : " + tree.deepestLeaves()
            print "Total word count : " + tree.totalWords()

            mostFrequentWord = tree.mostFrequentWord()
            mostFrequentCount = tree.findInstanceCount(mostFrequentWord)
            print "Most Frequent is : '" + mostFrequentWord + "' occurring " + mostFrequentCount
+ " times"
            print "First word pre-order traversal : " + tree.firstWordPreOrder()
            print "First word post-order traversal : " + tree.firstWordPostOrder()
            print "First word in-order traversal : " + tree.firstWordInOrder()

        catch IOException e:
            e.printStackTrace()

function splitAndCleanWords(line):
    // Split the line into words and remove non-alphabet characters
    words = line.split("\\s+")
    cleanedWords = []
    for word in words:
        cleanedWord = removeNonAlphabetChars(word)
        if cleanedWord is not empty:
            cleanedWords.append(cleanedWord)
    return cleanedWords

function removeNonAlphabetChars(word):
    // Remove non-alphabet characters from the word
    return word.replaceAll("[^a-zA-Z]", "").toLowerCase()
```