

## 3) (10 pts) DSN (Tries)

Write a recursive function that takes the root of a trie, *root*, and a single character, *alpha*, and returns the number of strings in the trie that contain that letter. You may assume that letter is a valid lowercase alphabetic character ('a' through 'z'). At some point, you will probably find it enormously helpful to write a helper function to assist with part of this task.

Note that we are *not* simply counting how many times a particular letter is represented in the trie. For example, if the trie contains only the strings “apple,” “avocado,” and “persimmon,” then the following function calls should return the values indicated:

```
countStringsWithLetter(root, 'p') = 2
countStringsWithLetter(root, 'm') = 1
```

The TrieNode struct and function signature are as follows:

```
typedef struct TrieNode {
    struct TrieNode *children[26];
    int numwords;
    int flag; // 1 if the string is in the trie, 0 otherwise
} TrieNode;

int countStringsWithLetter(TrieNode *root, char alpha) {
    int i, total = 0;
    if (root == NULL) return 0;

    for (i = 0; i < 26; i++) {
        if (i == alpha - 'a') {
            if (root->children[i] != NULL)
                total += (root->children[i])->numwords;
        }
        else
            total += countStringsWithLetter(root->children[i], alpha);
    }
    return total;
}
```

**Grading:**

- +1 for the base case
- +1 for initializing a counter (such as *total*) to 0
- +2 for the loop to go through all 26 letters.
- +1 for using (alpha - 'a') or (alpha - 97) in their comparison
- +2 for adding numwords to total in the case of the letter match (*be nice and don't take off if they miss the NULL check for root->children[i]*)
- +3 for adding in the recursive call in the case that the letters don't match. (1 pt for recursive call, 1 pt for total +=, 1 pt for parameter root->children[i], don't take off if they forget the parameter alpha)