3) (10 pts) DSN (Tries)

Write an **iterative, non-recursive** function that takes the root of a trie (*root*) and a string (*str*) and returns the number of new nodes we would have to add to our trie in order to insert that string. You may assume that *str* is non-NULL, non-empty, and contains lowercase alphabetic characters only (i.e., it won't contain uppercase letters or non-alphabetic characters). However, you must handle the case where *root* is NULL.

Special Restrictions:

a. Please do not use pointer arithmetic (e.g., str + 1).
b. Do not modify or corrupt the trie or the string. (Do not add nodes to the trie!)
c. Do not call *strlen()* repeatedly, as it is an O(k) function (where *k* is the length of the string). If you need to call *strlen()*, find a way to do it only once for the given string.

The trie node struct and function signature are as follows. Do NOT write any helper functions.

```
#include <string.h>
typedef struct TrieNode {
    struct TrieNode *children[26];
    int flag;  // 1 if the string is in our trie, 0 otherwise
} TrieNode;

int newNodeCount(TrieNode* root, char* str) {

    int len = strlen(str);

    for (int i=0; i<len; i++) {
        if (root == NULL) return len-i;
        root = root->children[str[i]-'a'];
    }

    return 0;

}
```

Grading: 2 pts for calling strlen only once.
1 pt for loop through string
2 pts for checking for NULL
2 pts for the return value when hitting NULL
2 pts for advancing to the appropriate next pointer
1 pt for returning 0 in the case the word is in the trie.