3) (10 pts) DSN (Tries)

Given a dictionary of words stored in a trie rooted at `root`, and a string, `str`, consider the problem of determining the length of the longest prefix of `str` that is also a prefix of one of the words stored in the trie. You may assume that if a link in the trie exists, a valid word is stored down that path in the trie. You may use string functions as needed, but please try to do so efficiently. (One point will be deducted for inefficient use of a particular string function.) For example, if str = "capitulate" and the trie stored the set of words {"actor", "bank", "cat", "capitol", and "caption"}, then the function should return 5 since, the first five letters of "capitulate" are the same as the first five letters of "capitol", and no other word stored in the trie shares the first six letters with "capitulate."

Complete the code for the function that solves this problem below. `root` is a pointer to the root of the trie and `str` is a pointer to the string. **You may assume that at least one word is stored in the trie.** The function signature and trie node struct definition are given below. Note that due to the function signature, **you must write your code iteratively.**

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

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


    // Grading: 2 pts - give 1 pt if called multiple times.
    int len = strlen(str);

    // Grading: 2 pts loop
    for (int i=0; i<len; i++) {

        // 3 pts - 2 pts for statement, 1 pt for placement (before NULL check)
        root = root->children[str[i]-'a'];

        // 2 pts - 1 pt for check, 1 pt for return
        if (root == NULL) return i;
    }

    // Grading: 1 pt
    return len;
}
```