3) (10 pts) DSN (Advanced Tree Structures: Tries)

Write a recursive function that takes the root of a trie and counts how many odd-lengthed strings there are in the trie. For example, if the trie contains the empty string (“”), “bananas”, “avocados”, and “randomness”, the function should return 1, because only one of those strings has a length that is odd (“bananas”).

We will make our initial call to your function like so: `countOddStrings(root, 0);`

Part of the fun in this problem is figuring out what to do with that second parameter.

Please do **NOT** write any helper functions. Restrict yourself to the function whose signature is given below.

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

int countOddStrings(TrieNode *root, int k) {

  int total = 0;

  if (root == NULL)
    return 0;

  if (k % 2 == 1 && root->flag == 1) total++;

  for (i = 0; i < 26; i++)
    total += countOddStrings(root->children[i], k + 1);

  return total;
}
```

Note: An alternate solution is to pass (k + 1) % 2 in the recursive call, so the k keeps toggling between 0 (even) and 1 (odd).

Grading (10 pts):
1 pts for checking root == NULL as a base case.
2 pts the line that increments total if k % 2 == 1 and root->flag == 1 (award partial credit)
2 pts for the for loop (double check that the range of ‘i’ is valid)
2 pts for the recursive call (make sure they use k + 1 and not k++ or ++k, as the latter two cause issues)
2 pts for capturing the return values of the recursive calls and using them to increment total
1 pt for returning total