Making Anagrams

Isaac DeJager

March 15, 2023

Click here to view the HackerRank page

1 Problem Description

A student is taking a cryptography class and has found anagrams to be very useful. Two string are anagrams of each other if the first string's letters can be rearranged to form the second string. In other words, both strings much contain the same exact frequency. For example, **bacdc** and **dcbac** are anagrams, but **bacdc** and **dcbad** are not.

The student decides on an encryption scheme that involves two large strings. The encryption is dependent on the minimum number of character deletions required to make the two strings anagrams. Determine this number.

Given two strings a and b, that may or not be of the same length, determine the minimum number of character deletions required to make a and b anagrams. Any characters can be deleted from either of the strings.

Example
$a = cde$
$b = def$
Delete e from a and f from b so that the remaining strings are cd and dc which are anagrams. This takes 2 character deletions.

Function Description
Complete the function *makeAnagram* which has the following parameters:

- string a: a string
- string b: another string

Returns

- int: the minimum totalc characters that must be deleted

Input Format

The first line contains a single string a. The second line contains a single string b.

Constraints

- $1 \leq |a|, |b| \leq 10^4$
- The strings a asnd b consist of lowercase English alphabetic letters, ascii[a-z]

Sample Input
cde
abc

Sample Output
4

Explanation
Delete the following characters from the strings make them anagrams:

- Remove d and e from cde to get c.
- Remove a and b from abc to get c.

It takes 4 deletions to make both strings anagrams.
The following solution is coded in C++.

2 Solution

```
//Function to calculate the ascii value of a character starting
//at a = 0
int charIndex(char c) {
    return int(c) - 97;
}

int makeAnagram(string a, string b) {

    //Initialize two integer arrays of size 26 with all values 0
    int aChars[26] = { 0 };
    int bChars[26] = { 0 };

    //Loop through string a
    for (int i = 0; i < a.length(); i++) {
        //For each character, increment the value in the aChars
        //array at the index calculated by the ascii value
        aChars[charIndex(a.at(i))]++;
    }

    //Repeat for string b
    for (int i = 0; i < b.length(); i++) {
        bChars[charIndex(b.at(i))]++;
    }

    //Sum and return the difference between each index in the arrays
    int x = 0;
    for (int i = 0; i < 26; i++) {
        x += abs(aChars[i] - bChars[i]);
    }

    return x;
}
```

3 Analysis

The program begins by generating two integer arrays of size 26, one for string a and another for string b. The purpose of these arrays is to count the number of each character. Index 0 = 'a', index 1 = 'b', and so on. Both arrays are initialized where every value is 0. This is done in $O(1)$ time.

Next, the program loops through each string and uses a function titled $charIndex$ to calculate the index of each character. $charIndex$ has one parameter: a single char c. It returns the ascii value of that character - 97. Then, the new character counting array at that index is incremented. It takes $2 * O(n)$ time to loop through both strings of size n and fill these arrays.

The final step is to loop through both char count arrays and sum the difference between each index. For example, if string a has 5 'a's and string b has 3 'a's, then a total of 2 'a's need to be added. Then repeat that for each character and sum up all the differences to find the total number of characters that need to be added between each string. It's notable that you do not need to know what string the characters need to be added two, just the total number to be distributed between them. This final step only takes $2 * O(26)$ time.

Thus the total time is:

$$O(1) + 2 * O(n) + 2 * O(26) = O(n)$$

Where n is the length of the larger string.

The space required for this algorithm is minimal. Simply two integer arrays of size 26.