Code Jam 2014 - Round 1B

Analysis: The Repeater

Let us first examine the winning condition for Omar. For this we will consider the following example:

```
3
aabcaa
abbcaa
abccaa
```

We represent the given strings by borrowing an idea from Run-length Encoding. Using the idea from this encoding the string â€œaabcdaâ€ can be represented as â€œ2:a, 1:b, 1:c, 1:d, 1:aâ€. The representation shows an ordered list of pairs of frequency and the corresponding character, letâ€™s call it the ***frequency string***. Similarly, we can figure out the frequency strings for all the original strings. The frequency strings for the strings in the example are listed below:

```
aabcaa  ->  2:a, 1:b, 1:c, 2:a
abbcaa  ->  1:a, 2:b, 1:c, 2:a
abccaa  ->  1:a, 1:b, 2:c, 2:a
```

Note that in the problem, an action is defined as either (i) repeating a character, or (ii) deleting a repeated character. In the frequency string, our allowed actions can be re-stated as follows. The first action is similar to increasing the frequency of a character, while the second action is similar to decreasing the frequency of a character. However note we can not decrease the frequency of a character below 1. Given these strings, we notice that Omar will win when the ordered list of characters (excluding the frequencies) are identical. Otherwise, Fegla will win as the allowed actions can only increase the frequency of a character in the encoded frequency string or decrease the frequency (but it is never allowed to go below 1). So if the ordered list of characters of any two strings do not match it will not be possible for Omar to win.

So we know how to figure out if Omar can win or not, we are now interested in the minimum number of actions to win the game. Notice that the actions performed on the i^{th} character in the frequency strings can be applied independently from characters in other positions in the frequency strings. Thus, we can process a set of characters at position i before processing the characters in the next position. For the example above, we can solve the leading â€œaâ€ first, then â€œbâ€, then â€œcâ€, and finally the trailing â€œaâ€. From hereon, we will describe solving for one character (at the i^{th} position) in the frequency strings. All other characters can be solved in the same way.

Brute force algorithm:

In the brute force solution, we will try all possible frequencies for a single character, that is for the character â€œaâ€, we will try all possible frequencies of â€œaâ€ which are â€œ1:aâ€, then â€œ2:aâ€, then â€œ3:aâ€, etc. We call each frequency we try the **target frequency**. In this problem, as the length of the string is only upto a 100 therefore we can try all possible frequencies from 1 through 100. For each target frequency, we can compute the sum of total actions (which will be the absolute difference between the frequency of that character and the target frequency) and maintain the target frequency that gives us the minimum sum of total actions.

The time complexity for the brute force algorithm is O(**L*N*X**), where **L** is the maximum length of any string, **N** is the number of strings, and **X** is the number of characters in the frequency string.

This solution should be sufficient to solve the provided small and large data set.

Efficient algorithm:

In the brute force solution, we tried all possible frequencies for a single character. But we donâ€™t need to do so, we can just try only one target frequency!

We would like to point out that intuitively, one might think that the **mean** would be the target frequency. Letâ€™s go through an example. Suppose we are given the frequencies {1, 1, 100}. The mean is 34. The sum of absolute differences from the provided frequencies to the target frequency is 33 + 33 + 66 = 132. But what if the target frequency was 30 instead of 34? Then the sum of absolute differences is 128, which is less than 132! In fact, if we tried all possible values for the target frequency (from 1 through 100), we would find that the minimum is at 1 with a sum of 99. In fact, it is the **median** that minimizes sum of absolute differences (1 is the median of {1, 1, 100}).

In our example above, the frequencies for leading â€œaâ€ are {2, 1, 1}, for â€œbâ€ are {1, 2, 1}, for â€œcâ€ are {1, 1, 2}, and for trailing â€œaâ€ are {2, 2, 2}. Therefore for the leading â€œaâ€, we would use the median which is 1 as the target frequency. Similarly for â€œbâ€ and â€œcâ€ we use 1, while for the trailing â€œaâ€ the target frequency is 2.

You might be wondering why the median is the right target frequency, we refer to an excellent explanation by Michal ForiÅ¡ek to the question â€œWhy does the median minimize the sum of absolute deviations?â€ which we quote here:

> *Imagine the given values as trees along a road. The quantity you are trying to minimize is the sum of distances between you and each of the trees.*
>
> *Suppose that you are standing at the median, and you know the current value of the sum. How will it change if you move in either direction? Note that regardless of which direction you choose, in each moment you will be moving away from at least half of the trees, and towards at most half of the trees. Therefore, the sum of distances can never decrease -- and that means the sum of distances at the beginning had to be optimal.*

In the explanation above, minimizing the sum of absolute deviations is equivalent to minimizing the total actions required to convert any ith character in the frequency string to a target frequency in our problem.