Kick Start 2018 - Round A

Analysis: Scrambled Words

Scrambled Words: Analysis

Let us first discuss how to check if two words are equivalent. A word is equivalent to another if their corresponding starting and ending letters are same and the sets of letters in between are either the same or permutations of each other. To check if two words are equivalent, after verifying that their starting and ending letters are equal, we populate a frequency array of the characters in each string and check if the two frequency arrays are same or not. Since the strings in this problem consist of only lowercase English letters, the array only needs to hold 26 integers (the counts of those letters' frequencies), and we can compare two such arrays very quickly. arrays will be very fast. An example of this would be the frequency away [1, 1, 2, 0, ... 0] for the string `bcca`.

Small dataset

Given the relatively small caps on **L** and **N** in the Small dataset, one solution that may come to mind is as follows: For each word in the dictionary, iterate over the Professor's text and check for substring match at each position in the text using the method of frequency arrays described above. But we should think carefully about how we check substrings against each other. Suppose that the length of our dictionary word is W. If we do it naively, for each position in the text (assume the length of the dict word is W), we check the substring of length W starting at that position by iterating over it, which is suboptimal. We should maintain a running frequency array. Suppose the frequency array is populated for the substring starting at position *i*. When we move our substring window one character to the right, we need to decrement the frequency of text[i] and increment the frequency of text[i+l]. This will give us the frequency array of the substring starting at position *i+1*. These two operations take constant time.

Since we iterate over the whole text for every dictionary word and there are **L** words and the length of the text is **N**, the total time complexity of this solution will be O(**NL**).

Large dataset

The above approach will not work in the large dataset because there are too many words in the dictionary.

The solution for this dataset hinges on the observation that if the total length of the words in the dictionary is bounded by **M**, the maximum possible number of distinct word lengths is atmost √**M**. This can be seen as follows. Let X be the bound on the number of distinct lengths; then X*(X+1)/2 ≤ **M**.

The above solution is now modified by iterating over only those lengths which are present in the dictionary, and for each such length, we perform the same process as above, checking against only the dictionary words with that length.. Let **M** be the upper bound on the dictionary which is given as 10^5. Since there are at most √**M** distinct lengths, the overall complexity of the solution is O(**N** √**M**).