Code Jam 2013 - Round 1B

Analysis: Garbled Email

There are multiple approaches possible to this problem. For the small test case, one can use a dynamic programming approach to calculate for each prefix of the given word what's the smallest number of substitutions needed to form this word so that the last substitution was *k* characters ago, for *k* = 1, 2, 3, ...

For example, for the word "codejam", we will find that "c" cannot be formed without a substitution, but can be formed (for instance from "a") by a substitution 1 character away. We find this by going over all dictionary words. Then, we go over all dictionary words to try and form "co" (we can do this, for instance, from "do" with one substitution 2 characters ago). We can also consider one letter words to extend the "c" we already know how to form, but this won't work, since "o" isn't a word, and we're too near to the last substitution. Next goes "cod", which actually is a word, so can be formed with zero subsitutions. Next goes "code" — for this we have a number of choice, like combining the "c" we know how to form and "ode", or the "cod" and a one substitution to form "e" from "a", or — the best one, since requiring no subsitutions — just using the word "code".

In this fashion for each prefix and each distance of the last substitution we can find out what's the least number of substitutions needed to form this prefix by looking at all smaller prefixes (including the empty one), all smaller dictionary words, and figuring out whether we can combine them.

For the large test case, we can't afford to go over the whole dictionary that often. So, we start by building a hash table. For each dictionary word, we insert that word into the hash table, and also insert the word with each possible set of changed letters in the word replaced by '*' characters.

For example, for the word "coders", we store in the hash table:

- `coders`
- `*oders`
- `c*ders`
- `co*ers`
- `cod*rs`
- `code*s`
- `coder*`
- `*oder*`

Next we use dynamic programming to build a table that contains, for each prefix of the email and location within that prefix of the last changed letter, the minimum number of changes required to transform a sequence of dictionary words into the prefix, if it is possible. (To save time, we can merge together all the states for a prefix where the last changed letter was 5 or more positions from the end of the prefix, because if the last changed letter is more than 4 positions back, it doesn't matter how much more.)

Each of the states where this is possible corresponds to a partial solution. We consider each possible way of adding one more word to create a longer partial solution. To do this, we try each combination of:

- The length of the next word, **L** ($1 \le \mathbf{L} \le 10$).
- Each possible set of positions of changed letters in the next word, **S**.

For each of these combinations, we construct a string by taking the next **L** letters of the original email, and changing the positions in **S** to '*' characters. Then we search for this string in the hash table. If we find it, we can update the table to reflect a partial solution formed by appending that word.

The answer is the minimum number of changes found to produce the prefix that is the entire email.