Code Jam 2020 - Round 1A

Analysis: Pattern Matching

Test Set 1

In Test Set 1, each pattern forces our answer to have a certain suffix, and we first need to check whether the patterns introduce conflicting requirements for that suffix.

Consider the letter strings coming after the initial asterisk in each pattern. We can find the longest of those strings (or any longest, if there is a tie); call that string L. Then at least one answer exists if (and only if) every other string is a suffix of L; note that we are considering L itself to be a suffix of L. We can check each other string against L by starting at the ends of both strings and stepping backwards through them in tandem until we find a discrepancy or run out of letters to check. If we ever find a discrepancy, then the case has no answer, but otherwise, we know that L itself is an acceptable answer.

Test Set 2

In Test Set 2, we can divide the patterns into (1) patterns that start with an asterisk, (2) patterns that end with an asterisk, and (3) patterns with an asterisk in the middle.

A type (1) pattern requires the output word to have a certain suffix, just as in Test Set 1. A type (2) pattern requires the output word to have a certain prefix. A type (3) pattern introduces both of these requirements, and we can split it into a suffix requirement and a prefix requirement, and then handle those separately.

Then, we can apply our strategy from Test Set 1 twice: once for the prefix constraints (with the algorithm modified to compare prefixes instead), and once for the suffix constraints. We can concatenate the two results together to obtain a valid answer that is certainly short enough (since it can be at most 99+99 characters long).

Test Set 3

We can generalize the idea above into a solution for Test Set 3. Each pattern p in Test Set 3 also prescribes a prefix of the output word (the prefix of p up to the first asterisk) and a suffix of the output word (the suffix of p starting after the last asterisk). If we allow empty prefixes and suffixes, we get exactly one of each for every pattern. We can handle those in the same way we did for Test Set 2, ending up with a prefix P and a suffix S for the output as long as we do not find a discrepancy in either phase.

However, for patterns that have more than one asterisk, we can also have a middle part, which imposes a new type of requirement. Suppose we parse the parts between the asterisks of a pattern so that X is the prefix up to the first asterisk, Y is the suffix after the last asterisk, and M_1, M_2, ..., M_k are the strings in between the asterisks, in order. After checking that X is a prefix of P and Y is a suffix of S, as before, all that remains to ensure is that the pattern $M_1*M_2*...*M_k$ is present somewhere within the output word, strictly between P and S.

Let us call $M_1M_2...M_k$ — that is, the word that occurs in between the first and last asterisks with any other asterisk removed — the *middle word*. If we make sure a pattern's middle word occurs in the output word outside of P and S, then we fulfill the extra requirement. We can then build a full valid output then by starting with P, then adding the middle word of every pattern in any order, then appending S. We make sure to correctly handle words with a single asterisk or only

consecutive asterisks by making their middle words the empty string. Since each middle word contains at most 98 characters, and the prefix and suffix contain at most 99 characters each, the output built this way has at most $99 \times 2 + 50 \times 98$ characters, which is within the limit of 10^4.