Contest Analysis

Overview | Problem A | Problem B | Problem C

The Last Word: Analysis

Small dataset

In the Small dataset, there will be at most 15 letters in **S**. At each step of the game, we are given a letter to add to either the front or the back of the current word. The number of possible words after adding the i-th letter (during step i) is at most twice the number of possible words after step i-1, since we have two choices of where to add the new letter. This means that the number of possible last words that can be made from all of the letters is at most 2^{15}. We can generate each of these possible last words and find which one comes last alphabetically.

Here is one way of doing this in Python:

```
def alphabetically_last_word(S):
  possible_words = set([''])
  for c in S:
    possible_words = set([c + r for r in possible_words] + [r + c for r
  return max(possible_words)
```

Large dataset

The approach in the previous paragraph is too slow for the Large dataset, and so some additional observations are required. During step i, we are adding a single new letter $\mathbf{S}_i$ to the front or the back of our current word X_{i-1}, yielding a new word $X_i = X_{i-1}\mathbf{S}_i$ or $X_i = \mathbf{S}_i X_{i-1}$. To have the end result be as alphabetically late as possible, it is always better to have X_i be as alphabetically late as possible as well. We can show this formally: during every step of the process that produces the alphabetically latest answer, after i letters have been chosen, our string X_i should be the alphabetically latest substring that we can produce from the first i letters of **S** under the given rules.

Assume we are at the i-th step, and we can either add the new letter $\mathbf{S}_i$ at the beginning or the end of X_{i-1}. Let Y_i be the alphabetically earlier of $\mathbf{S}_i X_{i-1}$ and $X_{i-1}\mathbf{S}_i$, and Z_i be the alphabetically later of the two. Suppose that Y_i were the optimal choice at this step. Then we could write our last word as AY_iB for some A and B. We could instead choose Z_i and insert the letters during future steps in the same way to yield AZ_iB as the last word. No matter what the values of A and B are, it is always true that AZ_iB comes no alphabetically earlier than AY_iB, because Z_i comes no alphabetically earlier than Y_i. This means that any word using Y_i can be turned into a word that is at least as late in alphabetical order by substituting Z_i at this step instead. It follows that choosing Z_i is always correct.

This means that our X_i must be the alphabetically latest of $X_{i-1}\mathbf{S}_i$ and $\mathbf{S}_i X_{i-1}$. Therefore, when we add $\mathbf{S}_i$ to X_{i-1}, we only need to check whether putting $\mathbf{S}_i$ in the front or putting $\mathbf{S}_i$ in the back would produce the alphabetically latest string.

Here is some simple Python code that implements the optimized procedure:

```
def alphabetically_last_word(S):
  result = ''
  for c in S:
    result = max(c + result, result + c)
  return result
```

Note that the solutions for the Small and Large datasets are very similar. The only difference is that the solution to the Large recognizes which one of the possible words that can be formed at each step will necessarily be part of the optimal last word. Instead of keeping an amount of information that may grow exponentially with the number of steps in the game, the code for the Large keeps track of a single string at each step, allowing it to run much faster and use less memory. The presented solution for the Small dataset requires exponential time and memory, whereas the presented solution for the Large dataset requires only polynomial time and memory.

Another way to think about this is that the `max` operation commutes with the set-building step inside the code for the Small, allowing us to keep the maximum at each step rather than computing the maximum at the end. This observation shows a path for extending a solution that solves the Small dataset into the one we explained that can also solve the Large dataset. (Check out "A possible stepping stone..." in this essay.)

Powered by

Google Cloud Platform