Prolonged Password
Prolonged Password

Given:
- A string \( S \) of alphabet characters.
- A function \( f(S,T) \) which transforms each character \( S_i \) into a string \( T_{Si} \).
- An integer \( K \) denoting how many times \( f(S,T) \) is performed, i.e. \( f^K(S,T) \).
- An integer \( M \) denoting the number of queries.
  - Each query contains an integer \( m_i \).

Determine:
- For each query, the \( m_i \)th character of \( f^K(S,T) \)

1 \leq |S| \leq 10^6; 2 \leq |T_x| \leq 50; 1 \leq K \leq 10^{15}; 1 \leq M \leq 1000; 1 \leq m_i \leq 10^{15}.
Prolonged Password

Example:

$S = \text{bccabac}$

$T_a = \text{ab}$

$T_b = \text{bac}$

$T_c = \text{ac}$

$T_d .. T_z$ are not important in this example.

$f^0(S,T) = \text{bccabac}$

$K = 1 \rightarrow f^1(S,T) = \text{bacacabbacabac}$

$K = 2 \rightarrow f^2(S,T) = \text{bacabacabacabacabbacbacabacabacabacabacabacabac}$
Prolonged Password

- How to generate $f^K(S,T)$ for large $K$?
  - $K$ can be very large, i.e. $10^{15}$ → a hint for $O(\log K)$ solution

- How to store $f^K(S,T)$?
  - Recall the constraints: $1 \leq |S| \leq 10^6$ and $2 \leq |T_x| \leq 50$
  - The complete $f^K(S,T)$ can be $10^6 \cdot 50^{10^{15}}$
  - Each query falls within the first $10^{15}$ characters → we cannot store $10^{15}$ characters
  - We need to output only ONE character per query → we have to exploit this.
Prolonged Password

• We don’t need to generate the whole $f^K(S,T)$.
  
  • Define $= |f^K(S,T)|$
  
  • Iterate through the string $S$ to find out which character we should recurse down into.
  
  • E.g.,

  $\begin{array}{cccccc}
    a & b & a & a & c \\
    \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
    30 & 20 & 30 & 30 & 50 \\
  \end{array}$

  Then, the 85th character can be obtained by expanding ‘a’ at index-3.

• $O\left( MK \max_i |T_i| + M|S| \right)$
Prolonged Password

To handle large $K$: Matrix Exponentiation

$N_{aa} = \text{count of character ‘a’ in } T_a.$
$N_{ab} = \text{count of character ‘b’ in } T_a.$
...
$N_{za} = \text{count of character ‘a’ in } T_z.$
$N_{zb} = \text{count of character ‘b’ in } T_z.$

$r_a = \text{count of character ‘a’}.$
$r_b = \text{count of character ‘b’}.$
...
$r_z = \text{count of character ‘z’}.$

\[
(r_a \ldots r_z) \begin{pmatrix}
N_{aa} & \cdots & N_{za} \\
\vdots & \ddots & \vdots \\
N_{az} & \cdots & N_{zz}
\end{pmatrix}
\]

\[
l^0(c, T) = r \\
l^1(c, T) = r \cdot N \\
l^2(c, T) = r \cdot N \cdot N \\
\ldots \\
l^K(c, T) = r \cdot N^K \\
\]

\[
\text{len}^K(c, T) = \|l^K(c, T)\|_1
\]
Prolonged Password

Another problem: $K$ is too large, $len^K(S, T)$ will be overflow.

Observation:
• $2 \leq |T_i| \rightarrow$ it means the string length doubles at each iteration.
• $2^{10^{15}}$ is way too large, but $m_i \leq 10^{15}$
• $10^{15} \leq 2^{50}$
• We can cut down $K$ by exploiting cycle in the transformation function.

$a \rightarrow bda$
$b \rightarrow cdc$    $a \rightarrow b \rightarrow c \rightarrow a$
$c \rightarrow ab$
Prolonged Password

Summary:
• Cut down K to \( \leq 50 \).
• Solve by recursing and using matrix exponentiation.
Prolonged Password

Summary:
• Cut down $K$ to $\leq 50$.
• Solve by recursing and using matrix exponentiation.

However, if you solve each query independently, you will get TLE as $M \leq 1000$.

→ You need to solve all queries at once (in one pass).
Magical String
Magical String

Given:

• A string $S$ which has no substring containing 3 or more identical characters.
• An integer $K$, the number of maximum operations.

An operation on $S$: Convert $S_i$ into another character (non-asterisk) s.t. $S$ contains a substring of 3 or more identical characters. Turn such (maximal) substring into an asterisk.

Determine:

❖ The maximum number of characters in $S$ which can be turned into asterisks with at most $K$ operations.

$1 \leq K, |S| \leq 1000$
Magical String

Example:
\[ S = abacaac \]

If \( K = 1 \)
\[ abacaac \rightarrow abaaac : ab^*c \]
ANS: 4

If \( K = 2 \)
\[ abacaac \rightarrow aaacaac : *caac \rightarrow *caaa : *c* \]
ANS: 6
Magical String

Example:

$S = \text{abacaac}$

If $K = 1$

$\text{abacaac} \rightarrow \text{abaaaac} : \text{ab*c}$

ANS: 4

If $K = 2$

$\text{abacaac} \rightarrow \text{aacaac} : \text{*caac} \rightarrow \text{*caaa} : \text{*c*}$

ANS: 6

This example suggests that the solution is not incremental, i.e. the solution for $(S,K)$ does not necessarily use the solution for $(S,< K)$.
Magical String

Example:
S = abacaac

If K = 1
abacaac → aba_aac : ab*c
ANS: 4

If K = 2
abacaac → a_aacaac : *caac → *caa_a : *c*
ANS: 6

This example suggests that the solution is not incremental, i.e. the solution for (S,K) does not necessarily use the solution for (S,< K)

Greedy does not work!

Also, the operations order does matter.
Magical String

first attempt ... dynamic programming

\( f(S, K) \rightarrow \) The maximum number of characters in \( S \) which can be turned into asterisks with at most \( K \) operations (i.e. the answer we want).

\[
f(S, K) = \max_{i \in \text{valid}(S, i)} \left( f(A, j) + f(B, K - j - 1) \right)
\]

Time complexity: \( O(|S|^3 \cdot K^2) \)

Definitely **TLE**

\[
\begin{array}{l}
\text{abacaaccbaabacbbba} \\
\text{abaca} \quad \text{aabacbbba}
\end{array}
\]
Magical String

... we need a muse and see the problem from a different perspective

Consider the **Weighted Interval Scheduling Problem**.

→ Given N intervals each with its weight, find a subset of intervals (at most of size K) s.t. there are no overlapping intervals and the total weight is maximized.

It’s a similar problem!

```
abacaaccbaabacbbba
aba
acaa
aac
acc
baa
aaba
cbb
bba
```
Magical String

... we need a muse and see the problem from a different perspective

Consider the **Weighted Interval Scheduling Problem**.

→ Given $N$ intervals each with its weight, find a subset of intervals (at most of size $K$) s.t. there are no overlapping intervals and the total weight is maximized.

It’s a similar problem!

```
abacaaccbaabacbbba
  aba
  acaa
  aac
  acc
  baa
  aaba
  cbb
  bba
```

... but different

```
abacaa
  aba
  acaa
```
Magical String

In Weighted Interval Scheduling Problem, we can only take one interval.

In Magical String, we can take “both” intervals.
Magical String

• Let SINGLE be the set of all intervals obtained individually from S.
• Let EXTEND be the set of all intervals obtained by extending SINGLE
  • \([a, b]\) is in EXTEND iff its size is \(\geq 3\) and there is an interval \([L, R]\) in SINGLE which can be **cut** into \([a, b]\) by other intervals in SINGLE.
  • By definition, all intervals in SINGLE are in EXTEND.

→ The solution for Weighted Interval Scheduling Problem with EXTEND as the intervals is the solution for Magical String.

| abacaa | [1, 3] |
| aba    | [3, 6] |
| acaa   | [4, 6] |
| caa    |        |

\([4, 6]\) is obtained by cutting \([3, 6]\) with \([1, 3]\).
Magical String

- Generate SINGLE $O(|S|)$
- Generate EXTEND $O(|S|^2)$

Size of EXTEND = $O(|S|)$

- Solve WISP with $K: N$ intervals $O(NK)$
Magical String

- Generate SINGLE $O(|S|)$
- Generate EXTEND $O(|S|^2)$

Size of EXTEND = $O(|S|)$

- Solve WISP with $K:N$ intervals $O(NK)$