4 Substring equality

Problem Introduction

In this problem, you will use hashing to design an algorithm that is able to preprocess a given string s to answer any query of the form "are these two substrings of s equal?" efficiently. This, in turn, is a basic building block in many string processing algorithms.

Problem Description

Input Format. The first line contains a string s consisting of small Latin letters. The second line contains the number of queries q. Each of the next q lines specifies a query by three integers a, b, and l.

Constraints. $1 \leq |s| \leq 500\,000$. $1 \leq q \leq 100\,000$. $0 \leq a, b \leq |s| - l$ (hence the indices a and b are 0-based).

Output Format. For each query, output "Yes" if $s_a s_{a+1} \ldots s_{a+l-1} = s_b s_{b+1} \ldots s_{b+l-1}$ are equal, and "No" otherwise.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 2 sec, `Python`: 10 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 5 sec, `Ruby`: 5 sec, `Scala`: 5 sec.

Memory Limit. 512MB.

Sample 1.

Input:

```
trololo
4
0 0 7
2 4 3
3 5 1
1 3 2
```

Output:

```
Yes
Yes
Yes
No
```

0 0 7 → `trololo` = `trololo`

2 4 3 → `trololo` = `trololo`

3 5 1 → `trololo` = `trololo`

1 3 2 → `trololo` ≠ `trololo`

What to Do

For a string $t = t_0 t_1 \cdots t_{m-1}$ of length m and an integer x, define a polynomial hash function

$$H(t) = \sum_{j=0}^{m-1} t_j x^{m-j-1} = t_0 x^{m-1} + t_1 x^{m-2} + \cdots + t_{m-2} x + t_{m-1}\,.$$

Let $s_a s_{a+1} \cdots s_{a+l-1}$ be a substring of the given string $s = s_0 s_1 \cdots s_{n-1}$. A nice property of the polynomial hash function H is that $H(s_a s_{a+1} \cdots s_{a+l-1})$ can be expressed through $H(s_0 s_1 \cdots s_{a+l-1})$ and

$H(s_0 s_1 \cdots s_{a-1})$, i.e., through hash values of two prefixes of s:

$$\begin{aligned} H(s_a s_{a+1} \cdots s_{a+l-1}) &= s_a x^{l-1} + s_{a+1} x^{l-2} + \cdots + s_{a+l-1} = \\ &= s_0 x^{a+l-1} + s_1 x^{a+l-2} + \cdots + s_{a+l-1} - \\ &- x^l (s_0 x^{a-1} + s_1 x^{a-2} + \cdots + s_{a-1}) = \\ &= H(s_0 s_1 \cdots s_{a+l-1}) - x^l H(s_0 s_1 \cdots s_{a-1}) \end{aligned}$$

This leads us to the following natural idea: we precompute and store the hash values of all prefixes of s: let $h[0] = 0$ and, for $1 \le i \le n$, let $h[i] = H(s_0 s_1 \cdots s_{i-1})$. Then, the identity above becomes

$$H(s_a s_{a+1} \cdots s_{a+l-1}) = h[a+l] - x^l h[a]\,.$$

In other words, we are able to get the hash value of any substring of s in just constant time! Clearly, if $H(s_a s_{a+1} \cdots s_{a+l-1}) \ne H(s_b s_{b+1} \cdots s_{b+l-1})$, then the corresponding two substrings ($s_a s_{a+1} \cdots s_{a+l-1}$ and $s_b s_{b+1} \cdots s_{b+l-1}$) are different. However, if the hash values are the same, it is still possible that the substrings are different — this is called a *collision*. Below, we discuss how to reduce the probability of a collision.

Recall that in practice one never computes the exact value of a polynomial hash function: everything is computed modulo m for some fixed integer m. This is done to ensure that all the computations are efficient and that the hash values are small enough. Recall also that when computing $H(s) \bmod m$ it is important to take every intermediate step (rather then the final result) modulo m.

It can be shown that if s_1 and s_2 are two *different* strings of length n and m is a prime integer, then the probability that $H(s_1) \bmod m = H(s_2) \bmod m$ (over the choices of $0 \le x \le m-1$) is at most $\frac{n}{m}$ (roughly, this is because $H(s_1) - H(s_2)$ is a non-zero polynomial of degree at most $n-1$ and hence can have at most n roots modulo m). To further reduce the probability of a collision, one may take two different modulus.

Overall, this gives the following approach.

1. Fix $m_1 = 10^9 + 7$ and $m_2 = 10^9 + 9$.
2. Select a random x from 1 to 10^9.
3. Compute arrays $h_1[0..n]$ and $h_2[0..n]$: $h_1[0] = h_2[0] = 0$ and, for $1 \le i \le n$, $h_1[i] = H(s_0 \cdots s_{i-1}) \bmod m_1$ and $h_2[i] = H(s_0 \cdots s_{i-1}) \bmod m_2$. We illustrate this for h_1 below.

```
allocate h1[0..n]
h1[0] ← 0
for i from 1 to n:
   h1[i] ← (x · h1[i − 1] + s_i) mod m1
```

4. For every query (a, b, l):
   (a) Use precomputed hash values, to compute the hash values of the substrings $s_a s_{a+1} \cdots s_{a+l-1}$ and $s_b s_{b+1} \cdots s_{b+l-1}$ modulo m_1 and m_2.
   (b) Output "Yes", if

   $$\begin{aligned} H(s_a s_{a+1} \cdots s_{a+l-1}) \bmod m_1 = H(s_b s_{b+1} \cdots s_{b+l-1}) \bmod m_1 \text{ and} \\ H(s_a s_{a+1} \cdots s_{a+l-1}) \bmod m_2 = H(s_b s_{b+1} \cdots s_{b+l-1}) \bmod m_2\,. \end{aligned}$$

   (c) Otherwise, output "No".

Note that, in contrast to Karp–Rabin algorithm, we do not compare the substrings naively when their hashes coincide. The probability of this event is at most $\frac{n}{m_1} \cdot \frac{n}{m_2} \le 10^{-9}$. (In fact, for random strings the probability is even much smaller: 10^{-18}. In this problem, the strings are not random, but the probability of collision is still very small.)