3 Problem: Find pattern in text

Problem Introduction

In this problem, your goal is to implement the Rabin–Karp's algorithm.

Problem Description

Task. In this problem your goal is to implement the Rabin–Karp's algorithm for searching the given pattern in the given text.

Input Format. There are two strings in the input: the pattern P and the text T.

Constraints. $1 \le |P| \le |T| \le 5 \cdot 10^5$. The total length of all occurrences of P in T doesn't exceed 10^8. The pattern and the text contain only latin letters.

Output Format. Print all the positions of the occurrences of P in T in the ascending order. Use 0-based indexing of positions in the the text T.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 5 sec, `Python`: 5 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 3 sec, `Ruby`: 3 sec, `Scala`: 3 sec.

Memory Limit. 512Mb.

Sample 1.

Input:

```
aba
abacaba
```

Output:

```
0 4
```

Explanation:
The pattern *aba* can be found in positions 0 (**aba**caba) and 4 (abac**aba**) of the text *abacaba*.

Sample 2.

Input:

```
Test
testTesttesT
```

Output:

```
4
```

Explanation:
Pattern and text are case-sensitive in this problem. Pattern $Test$ can only be found in position 4 in the text $testTesttesT$.

Sample 3.

Input:

```
aaaaa
baaaaaaa
```

Output:

```
1 2 3
```

Explanation:
Note that the occurrences of the pattern in the text can be overlapping, and that's ok, you still need to output all of them.

Starter Files

The starter solutions in C++, Java and Python3 read the input, apply the naive $O(|T||P|)$ algorithm to this problem and write the output. You need to implement the Rabin–Karp's algorithm instead of the naive algorithm and thus significantly speed up the solution. If you use other languages, you need to implement a solution from scratch.

What to Do

Implement the fast version of the Rabin–Karp's algorithm from the lectures.

Some hints based on the problems encountered by learners:

- Beware of integer overflow. Use `long long` type in C++ and `long` type in Java where appropriate. Take everything $\pmod p$ as soon as possible while computing something $\pmod p$, so that the numbers are always between 0 and $p - 1$.
- Beware of taking negative numbers $\pmod p$. In many programming languages, $(-2)\%5 \neq 3\%5$. Thus you can compute the same hash values for two strings, but when you compare them, they appear to be different. To avoid this issue, you can use such construct in the code: $x \leftarrow ((a\%p) + p)\%p$ instead of just $x \leftarrow a\%p$.
- Use operator `==` in Python instead of implementing your own function `AreEqual` for strings, because built-in operator `==` will work much faster.
- In C++, method `substr` of `string` creates a new string, uses additional memory and time for that, so use it carefully and avoid creating lots of new strings. When you need to compare pattern with a substring of text, do it without calling `substr`.
- In Java, however, method `substring` does NOT create a new `String`. Avoid using `new String` where it is not needed, just use `substring`.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.