Programming Part

String Matching (20 pts)

Problem Description

In this question, we will combine the Rabin-Karp string matching algorithm and the KMP string matching algorithm to solve a "multi-line" string matching problem. Although this sounds complex, it is a direct application of the original algorithms and not too difficult to understand.

The multi-line string matching problem is defined as follows. Given a multi-line text $T[1:k, 1:n]$ (two-dimensional array) of dimensions k and n, and a multi-line pattern $P[1:k, 1:m]$ (also a two-dimensional array) of dimensions k and m, the goal is to find all valid shifts s, such that $T[1:k, s+1:s+m] = P[1:k, 1:m]$. Note that the number of lines k in T and P are the same here. The strings T and P use only the uppercase and lowercase English alphabets, i.e., $\Sigma = \{A, B, C, \ldots, Z, a, b, c, \ldots, z\}$. In addition, we define the values of the characters 'A' to 'Z' and 'a' to 'z' to $0, 1, \ldots, 25$ and $26, 27, \ldots, 51$, respectively. Those are used in the string arrays T and P. Also, see the illustration in Figure 1.

To solve the problem, let's first apply the Rabin-Karp hashing scheme in the direction of the first dimension for both T and P. Let's define the hash of a multi-line pattern $Q[1:k, 1:m]$ for a single column j, $1 \le j \le m$, as follows:

$$\tau(Q, j) = d^{k-1} \cdot Q[1, j] + d^{k-2} \cdot Q[2, j] + d^{k-3} \cdot Q[3, j] + \cdots + d \cdot Q[k-1, j] + Q[k, j] \pmod{q} \quad (1)$$

where $d = |\Sigma| = 52$, the size of the alphabet forming the strings, and q is a large prime number.

Thus, with this definition, we can convert the orignal multi-line text T into an ordinary one-dimensional text T':

$$T'[j] = \tau(T, j), 1 \le j \le n \quad (2)$$

and the original multi-line pattern P into an ordinary one-dimensional text P':

$$P'[j] = \tau(P, j), 1 \le j \le m \quad (3)$$

Subsequently, we can apply the original KMP string matching algorithm with input text T' and pattern P'. However, when there exists a valid shift s for T' and P', i.e., $T'[s+j] = \tau(T, s+j) = \tau(P, j) = P'[j]$ for $1 \le j \le m$, because of the possibility of spurious hits, it only implies that it is *possible* that $T[1:k, s+1:s+m] = P[1:k, 1:m]$. Similar to how the Rabin-Karp algorithm works, in this case we will need to compare $T[1:k, s+1:s+m]$ and $P[1:k, 1:m]$ to verify if they exactly match.

Please implement your algorithm according to the above description to solve this problem. Note that the running time of pre-processing with Rabin-Karp hashing is $O(k(n+m))$. Then, the running time of the KMP string matching algorithm stays the same, which is $O(m)$ for pre-processing, i.e., calculation of the prefix function, and $O(n)$ for matching[1].

Input

The first line includes four integers k, n, m, and q, separated by a single space character. T is of dimensions k and n and P is of dimensions k and m. Finally, q specifies the large prime number used to produce the hash values in Eq. 1.

The next k lines specify the strings of each line of T, each having n characters.

Then, the next k lines specify the strings of each line of P, each having m characters.

Output

The first line of output should have n numbers, $T'[1], T'[2], \ldots, T'[n]$, as defined in Eq. 2, separated by a space character.

The second line of output should have m numbers, $P'[1], P'[2], \ldots, P'[m]$, as defined in Eq. 3, separated by a space character.

The third line of output should have the values of all valid shifts of the "converted" T' and P' string matching problem. That is, output all s values such that $T'[s+1 : s+m] = P'[1 : m]$. If there is no valid shift, output -1.

The fourth line of output should have the values of all shifts which are spurious hit. That is, you have $T'[s+1 : s+m] = P'[1 : m]$, but $T[1 : k, s+1 : s+m] \neq P[1 : k, 1 : m]$. If there is no such case, output -1.

Constraint

- $1 \leq k, n, m \leq 10^6$
- $1 \leq km \leq kn \leq 10^6$
- $2 \leq q \leq 10^9 + 7$
- The input string use only the uppercase and lowercase.

Sample Testcases

Sample Input 1

```
1 6 3 53
ABCxyz
ABC
```

Sample Output 1

```
0 1 2 49 50 51
0 1 2
0
-1
```

[1]Here we assume the numbers of valid shifts and suprious hits are small, and do not discuss the running time of the final verification step comparing $T[1 : k, s+1 : s+m]$ and $P[1 : k, 1 : m]$.

Sample Input 2

```
3 5 3 1000000007
abcde
fghij
klmno
ABC
FGH
KLM
```

Sample Output 2

```
71952 74709 77466 80223 82980
270 3027 5784
-1
-1
```

Sample Input 3

```
1 6 1 2
abcdef
a
```

Sample Output 3

```
0 1 0 1 0 1
0
0 2 4
2 4
```

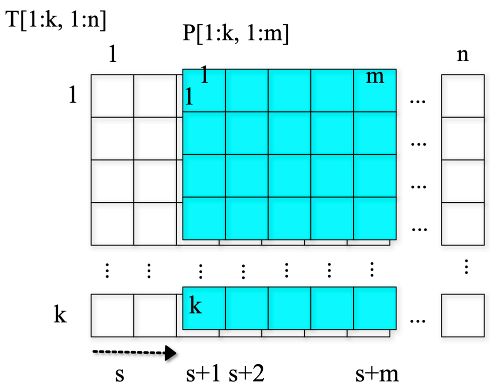

Figure 1: Multi-line String Matching of input T and P