Data Structure and Algorithm

Homework #2

Due: 1:20pm, Tuesday, April 11, 2017

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1-2, please put all your solutions in a PDF file and name it *[StudentID]_hw2.pdf*, *e.g.*, *b05902987_hw2.pdf*. Please also **include your name and student ID on the first page** of the PDF file before submitting it to the online judge (http://140.112.91.212/judge/). You can either type them or write on papers and scan them. If you choose the latter, please make sure that your writing is recognizable and clear enough, otherwise you might receive some penalties. For Problem 3 and 4, they should be submitted individually and will be judged by the online judge (http://140.112.91.212/judge/).
- Discussions with others are encouraged. However, you should write down the solutions in your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted and/or the people you discussed with) on the first page of your solution to that problem.
- For all programming problems, only C99 standard C langauge programs are accepted. (i.e., C++, or anything else, is not supported by the online judge system)
- For all the problems, up to one day of delay is allowed; however, you will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - \tfrac{\text{Delay Time}}{86400})$$

Penalty will be calculated separately for each of the following three parts (1) problem 1-2 (non-programming problems); (2) problem 3; (3) problem 4.

Problem 1. Pancake (30%)

Pancake sorting is a sorting algorithm with a special operation $FLIP(k)$, which reverses the first k numbers of the sequence. We give an example below.

Array: 4 3 5 2 1
FLIP(2)
Array: 3 4 5 2 1
FLIP(3)
Array: 5 4 3 2 1
FLIP(5)
Array: 1 2 3 4 5

We define the *pancake number* as the minimum number of $FLIP$s required to sort a given sequence. You can try to verify that the pancake number of the sequence above is 3, as is shown.

1.1. (9%) Please calculate the pancake number of the following sequences:

   (a) (3%) 2 1 3 5 4

   (b) (3%) 5 4 1 2 3

   (c) (3%) 1 4 3 2 5

1.2. (7%) Consider the algorithm given below

```
void Pancake_Sort(int A[], int N){
    for(int i=N ; i>=2 ; i--){
        if(is_sorted(A)) break;
        int p = 1;
        for(int j=1 ; j<=i ; j++){
            if(A[j] > A[p]){
                p = j;
            }
        }
        if(p == i) continue;
        FLIP(A, p);
        FLIP(A, i);
    }
    return;
}
```

Please construct a sequence A, which is a permutation of 1 to 10 ([1, 3, 5, 7, 9, 2, 4, 6, 8, 10], for instance) and maximize the times of calling $FLIP$ function. In other words, construct a permutation of length 10 with the largest possible pancake number.

1.3. (14%) William H. Gates, known as Bill Gates, devised an algorithm for pancake sorting in his second year at Harvard. He gave an upper bound of the maximum pancake number of length N, which is $\frac{(5N+5)}{3}$, and published the result in the paper *"Bounds for sorting by prefix reversal"*[1]. (Hint: if you have no idea for this problem, you may read the introduction of the paper.)

(a) (7%) Prove that for any $N \geq 1$, any sequence of N distinct integers has a pancake number of no more than $2N - 2$. (Hint: you can give an algorithm in pseudo code and prove that the algorithm calls $FLIP$ for no more than $2N - 2$ times.)

(b) (7%) Prove that for any $N \geq 1$, there exists at least one permutation of 1 to N whose pancake number is at least $N - 1$.

[1]Bill Gates and Christos Papadimitriou, Bounds For Sorting By Prefix Reversal. Discrete Mathematics, vol 27, pp 47-57, 1979.

Problem 2. String Matching (20%)

2.1. (4%) Use *Rabin-Karp* matcher to find pattern $P = 87$ in text $T = 287471356248$ with moduler 13 and base 10. Please write down the number of valid match and spurious hit. Draw the procedure similar to $Figure$ 32.5(b) in the textbook (p.16 in string matching slides).

2.2. (3%) Give a linear-time algorithm and write down the pseudo code to determine whether a text T is a cyclic rotation of another string T_0 . For example, *arc* and *car* are cyclic rotations of each other.

2.3. (5%) For a string *tatatata*, we can denote it as $(ta)^4$ or $(tata)^2$. Now let $X \in \Sigma^*$ and $Y \in \Sigma^*$, how to find the largest value r such that $X = Y^r$? Suppose X has the length of m. Please design an algorithm and write down the pseudo code. Note that your algorithm should take linear time.
Hint: Use the prefix function.

2.4. (8%) Now, we want to introduce *Boyer-Moore* for you.
Please simulate KMP algorithm and *Boyer-Moore* algorithm for searching the pattern $P = 9487$ in text $T = 947892879487$. Draw each step of pattern shifting of both algorithms, and compare their behavior.
To draw the procedure for *Boyer-Moore* algorithm, you must know what is $The\ Good\ Suffix\ Rule$ and $The\ Bad\ Character\ Rule$. In each step, you must write down the rule you use and explain why.
Hint: Google Boyer-Moore algorithm :)

Problem 3. Interesting String Pair (Programming problem) (25%)

Eddy likes to play with strings, especially, a pair of strings.

Now, Eddy gets a pair of strings, S and T. He wants to find out some interesting properties between them. Eddy defines the interesting value of a pair of string S and T as follows:

$$V(S,T) = max(occ(S,T), occ(T,S))$$

, where $occ(S,T)$ is the number of occurrences of S in T. Formally, $occ(S,T)$ is number of substrings of T which are equivalent to S.

For example,

$$occ(\texttt{a},\texttt{a}) = 1, occ(\texttt{a},\texttt{b}) = 0, occ(\texttt{a},\texttt{aa}) = 2, occ(\texttt{ab},\texttt{bab}) = 1, occ(\texttt{aa},\texttt{a}) = 0, occ(\texttt{aba},\texttt{ababa}) = 2,$$

However, original strings S and T are too large for Eddy. He can't directly compute the interesting value of them. Eddy believes that the original interesting value will be positively correlated to the intesting values of some pairs of substring of S and substring of T. However, it's still too difficult for Eddy to compute those interesting values. Thus, you decide to help Eddy compute the interesting values of those pairs of substrings of S and T.

Input format

The first line contains one positive integer C indicating that Eddy has C pairs of string S and T.

For each pair of strings: First line contains a string S. Second line contains a string T. Third line contains an integer Q indicating that Eddy wants to find out the interesting values of Q pairs of substring of S and substring of T, respectively. The following Q lines each contains 4 integers i_1, j_1, i_2, j_2 indicating that Eddy wants to find out the interesting value of substring of S from i_1 to j_1 and substring of T from i_2 to j_2 (inclusive on both sides), *i.e.*, $V(S[i_1, j_1], T[i_2, j_2])$.

Input constraint

It is guaranteed that

- the lengths of S and T are both no more than 10^5.
- The sum of the length(s) of the given C pairs of strings S and T is less than or equal to 2×10^5 characters
- All pairs of strings S and T consist of lowercase English alphabet only, *i.e.*, *a-z*.
- $0 < Q \leq 10^5$
- **either** $i_1 = 0, j_1 = |S| - 1$ **or** $i_2 = 0, j_2 = |T| - 1$
- The sum of all Q's for the given C pairs of strings S and T is less than or equal to 2×10^5
- $0 \leq i_1 \leq j_1 \leq |S| - 1$, $0 \leq i_2 \leq j_2 \leq |T| - 1$, where $|S|$ is the length of S and $|T|$ is the length of T

- For 20% points, sum of the length(s) of the given C pairs of strings S and T is less than or equal to 2×10^3 characters and the sum of Q of the given C pairs of strings S and T is less than or equal to 10^2
- For 80% points, sum of the length(s) of the given C pairs of strings S and T is less than or equal to 2×10^5 characters and the sum of Q of the given C pairs of strings S and T is less than or equal to 10^2

Output format

For each pair of substrings Eddy asks, you should output one line consisting of only one integer, which is the interesting value of those substrings.

Sample Input

```
2
aaa
aa
3
0 0 0 1
0 2 0 1
2 2 0 1
abababa
aba
5
0 6 0 2
0 0 0 2
0 4 0 2
0 5 0 2
0 6 1 2
```

Sample Output

```
2
2
2
3
2
2
2
3
```

Hint

- *A substring of string S is a prefix of a suffix of string S.*
- *If we sort all the suffixes of a string in lexicographical order, those with similar prefixes will be adjacent. For example, Following are sorted suffixes of* `"ababa"`

  `a   aba   ababa   ba   baba`
- *To compare the lexicographical order of two string S_1, S_2 both length equal 2^k, we can first compare first $2^{(k-1)}$ characters. If $S_1[0 : 2^{(k-1)} - 1]$ and $S_2[0 : 2^{(k-1)} - 1]$ are the same, we then compare last $2^{(k-1)}$ characters. If we can compare each part in $O(1)$, we can compare the order of S_1 and S_2 in $O(1)$.*
- *Are two strings too much to deal with? How about concatenating them into one string?*

***Problem* 4.** Secret Code (Programming problem) (25%)

HsinHsin and MuMu are best friends in college. They pass notes in class because they can't help sharing their thoughts and feelings all the time. They developed a coding method to help cipher the text so that people will not learn their secrets when the notes are peaked at either accidentally or intentionally.

The algorithm for ciphering the text is depicted as below:

1. Let $\mathbb{S}$ be the set of characters that appear in the text we'd like to cipher.
2. Sort the characters in $\mathbb{S}$ by their number of occurrences in descending order. If there are several characters with the same number of occurrences, they will be sorted alphabetically.
3. For the i-th character in $\mathbb{S}$, replace all its occurrences in the original text with j-th character in $\mathbb{S}$, where $j = |\mathbb{S}| - i - 1$

(Note: we only consider the lowercase English alphabet, *i.e.*, a-z, here. That is, all punctuations and spaces in the original text will be left unchanged in the output cipher.)

For example, if HsinHsin wants to tell MuMu that *"ta is handsome."*, he would first count the occurrences of all characters in the sentence. After sorting, the sorted character set would be: *"asdehimnot"*. That is, we would replace *"a"* with *"t"*, *"s"* with *"o"*, etc. The ciphered text is therefore *"at ho itdnosem"*. Now, please design a program for HsinHsin and MuMu to create the ciphered text for a given sentence.

Input Format

The first line contains an integer M, the number of sentences to cipher, followed by M lines. Each of the M lines would be a sentence that HsinHsin and MuMu would like to cipher.

Input Constraint

It is guaranteed that

- Each sentence is not empty and consists of lowercase letters, punctuations, and spaces only.
- The sum of the length(s) of the given M sentence(s) is less than or equal to 1000000.

Output format

For each of the M given sentenes, please output the cipher in one line.

Sample Input

```
2
ta is handsome.
i love you!
```

Sample Output

```
at ho itdnosem.
u lyev oyi!
```