Data Structure and Algorithm, Spring 2018

Homework 2

Release: Tuesday, April 3, 2018

Due: 23:59:59, Sunday, April 15, 2018

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

Rules and Instructions

- In homework 2, the problem set contains 5 problems and is divided into two parts, non-programming part (problem 1, 2, 3) and programming part (problem 4, 5).
- Please go to *DSA Judge* (`https://dsa.csie.org`) to complete this homework. If you have any problem, check out the uploading tutorial in problem 0.
- For problems in non-programming part, you should combine your solutions in ONE pdf file, and then hand it in via the judge system. 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. The pdf file should contain your STUDENT ID and your NAME. If not, you would get minor penalties. Your solution must be as simple as possible. At the TAs' discretion, too complicated solutions will be counted as wrong.
- For problems in programming part, you should write your code in C programming language, and then hand them in via the judge system. You have 5 chances per day for each problem. The compilation command of the judge system will be `gcc main.c -static -std=c11 -O2 -lm`.
- Discussions with others are encouraged. However, you should write down your solutions in your own words. In addition, for each problem, you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem.
- Do NOT lend your judge system account to others or borrow other students' accounts. Both cases would be regarded as plagiarism.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$LateScore = (\frac{86400 - DelayTime(sec.)}{86400})OriginalScore$$

Non-Programming Part

Problem 1. Pusheens (15%)

Pusheen is a cute animal which has some kind of superpower such as Teleport and Division. You are a teacher in Pusheen Kingdom. One day, you decide to take a class of Pusheens on a field trip:

1. (4%) You've divided them into 3 groups according to their seat numbers, and then you have to put them in order such that their group number is increasing. For example, given $[2, 1, 0, 2, 1]$, the output should be $[0, 1, 1, 2, 2]$. So, let's sort your Pusheens!

   Time complexity: $O(N)$; N is the number of Pusheens.
   Space complexity: $O(1)$
   There are only 3 numbers: $0, 1, 2$. Please complete the algorithm below to achieve the increasing order. Note that you have to use **two meaningful for-loops** and you cannot declare other variables; otherwise, you won't get any score. Briefly explain why the algorithm reaches the time and space complexity.

```
1 Algorithm Sort_Pusheen_1 (Pusheen_Array)
2     int n0 = 0, n1 = 0, n2 = 0;
3     for(i=0; i<length(Pusheen_Array); i++){
4         //complete the code
5     }
6     for(i=0; i<length(Pusheen_Array); i++){
7         //complete the code
8     }
9     return Pusheen_Array
```

2. (5%) After having lunch, some Pusheens split into two pieces, so the number of Pusheens in each group is different from that in the morning. That means, you have to sort them again! To speed up the process, you have to use their Teleport ability, which equals to the Swap function. Swap(i, j) exchanges the elements of index i and index j.

   Time complexity: $O(N)$; N is the number of Pusheens.
   Space complexity: $O(1)$
   There are only 3 numbers: $0, 1, 2$. Please complete the algorithm below to achieve the increasing order. Note that you can use only **one for-loop** and you have to use **Swap(i, j)**; otherwise, you won't get any score. Also, you cannot declare other variables. Briefly explain why the algorithm reaches the time and space complexity.

```
Algorithm Sort_Pusheen_2 (Pusheen_Array)
    int n0 = 0, n2 = length(Pusheen_Array)-1;
    for(i=0; i<length(Pusheen_Array); i++){
        //complete the code using Swap
    }
    return Pusheen_Array
```

3. (6%) At the end of the trip, you divide them into 5 groups to make them go home easily. Since they are tired, they can't use Teleport. Please sort your pusheens!
   Time complexity: O(N); N is the number of Pusheens.
   Space complexity: O(1)

   There are 5 numbers: $0, 1, 2, 3, 4$. Please complete the algorithm below to achieve the increasing order. Note that you can use only **one for-loop** and you **cannot use Swap**; otherwise, you won't get any score. Also, you cannot declare other variables. Briefly explain why the algorithm reaches the time and space complexity.

```
Algorithm Sort_Pusheen_3 (Pusheen_Array)
    int n0 = -1, n1 = -1, n2 = -1, n3 = -1, n4 = -1;
    for(i=0; i<length(Pusheen_Array); i++){
        //complete the code
    }
    return Pusheen_Array
```

Problem 2. Naughty Rabbits (20%)

You are a rabbit keeper; you keep a lot of rabbits in your house. One day, you decide to sort your rabbits according to their ages to form a line. The young ones stand in the front of the line; the old ones stand back. You don't know their ages at all, but the rabbits know one another's age well. It is guaranteed that no two rabbits are the same age. Also, your rabbits are intelligent! They can answer a certain form of Yes-No questions like "Is Rabbit A younger than Rabbit B?". If you ask Rabbit C that question, it will reply "Yes." or "No." There's one more problem left: you plan to ask those rabbits to figure out the order, but not all rabbits are kind. There are good rabbits that always tell the truth and naughty rabbits that sometimes lie. However, you cannot come up with a better alternative so you choose to ask for their help. Suppose there are N rabbits and strictly more than $\lceil \frac{N}{2} \rceil$ of them are good rabbits.

1. (4%) Please modify Merge Sort and then write down a $O(N^2 \log N)$ solution to sort the rabbits. Extra space complexity would be restricted to $O(1)$. (However, you can use $O(N)$ space complexity storing the results of merge operation as an exception.) Briefly explain the time complexity.

2. (6%) Please apply a new version of Quick Sort to the problem: randomly select k(odd number) rabbits, and choose the median of the k rabbits to be the pivot; if the number of rabbits is less than k, simply apply Insertion Sort. Assume that the median of k rabbits is also found by Insertion Sort, please prove that the worst case time complexity of sorting your N rabbits is $O(\frac{N^3}{k} + N^2 k)$.

Next, you would like to experiment on a new sorting algorithm Stooge Sort:

```
1 Algorithm Stooge-Sort (Rabbit_Array, start = 0, end = N − 1)
2     If Rabbit_Array[start] older than Rabbit_Array[end], swap them
3     If end-start+1 ≥ 3, Let M = ⌊((end−start+1)/3)⌋
4         Stooge-Sort(Rabbit_Array, start = start, end = end − M)
5         Stooge-Sort(Rabbit_Array, start = start + M, end = end)
6         Stooge-Sort(Rabbit_Array, start = start, end = end − M)
7 return Rabbit_Array
```

3. (3%) Discuss the purpose of the second line in the Stooge Sort algorithm. Could it be removed from the algorithm? Why or why not?

4. (7%) The time complexity of Stooge Sort algorithm upon the case of naughty rabbits would be $O(N^{log_{1.5}c})$. Calculate the constant c and prove the time complexity.

Problem 3. TA has just learned KMP TAT (25%)

For your convenience, you can call the function *prefix_function(string)* to get the prefix function of a certain string in the following problems.

1. (5%) Let's denote your student ID (with every alphabet capitalized, e.g. B06902123) as X. Now, we want to use Rabin-Karp algorithm to test whether a string S contains X as substring. Please provide a string that does not contain X but would make the "spurious hit" happen.

   - Assume the modular is 77 and the base is 36 (There could be $26+10$ different characters in student ID), and '0' $\rightarrow$ 0, '1' $\rightarrow$ 1, ..., 'A' $\rightarrow$ 10, ... 'Z' $\rightarrow$ 35
   - Brief explanation should be provided to get full credit.

2. (7%) *Mumu* is a busy person, so he wants to avoid typing too many characters by leveraging Ctrl-C and Ctrl-V to copy the duplicated substring and paste it multiple times to documents. That is, **repeated strings** such as "*ohohoh*" can be represented as $(oh)^3$. For a given string X and its length n, please fill in the blank to help him identify whether X is a repeated string, that is, $X = (S)^k$ for some string S and $k > 1$. In order to get full credit, you need to prove its correctness.

```
1 int is_repeated_string (char *X, int n) {
2     int *pi = prefix_function(X);
3     if (____________________) //should be an O(1) operation
4         return 1; //True
5     return 0; //False
6 }
```

For the following questions 3 and 4, you need to briefly explain their correctness and prove the time complexity in order to get full credit.

3. (6%) Palindrome is a type of string that reads the same backwards as forwards, such as "madam" and "racecar". Provide a **linear-time algorithm** that can find a shortest feasible string Y that makes X + Y a palindrome. For example, "ba" is the answer to input string "abaca" since "abacaba" is the shortest possible palindrome for the given prefix "abaca".

4. (7%) Now, we want to introduce a simple version of wildcard matching. Given a pattern and a string, the matching algorithm will return a boolean value representing whether the pattern is included in the string. In the pattern, there could be any number of special character "*", which can be matched with an arbitrary string. Here are some examples:

   - pattern "ab*t" can be matched with "abt", "abxyt", "abrt", ...
   - pattern "ab" can be matched with "ab", "cabt", ...
   - pattern "*" can be matched with "a", "ab", "abc", ...

   So, given a pattern P and string S, along with the corresponding length n and m, please modify KMP to provide an $O(n+m)$ algorithm that determines whether P can be matched with a substring of S. For simplicity, only English alphabets and "*" will appear in P.

Programming Part

Problem 4. Arvin, who never wins (20%)

Arvin invented a game, but he could not win. Actually, he has never won. Thus, he seeks for your help. The game is described as below:

Having two strings N and M, players take turns fitting prefixes of M onto N. Fitting the same prefix onto a position selected previously in the game is not allowed. The player who fails to find any possible fit will lose. Additionally, to justify your answer, you must also output the number of **aggregated** coverage of all letters in N $mod(10^9 + 7)$ of the game.

An example is shown here:

$N = abbaba$

$M = aba$

Player1 : [a]bbaba

Player2 : abb[a]ba

Player1 : abbab[a]

Player2 : [ab]baba

Player1 : abb[ab]a

Player2 : abb[aba]

Player1 fails to find next fit.

'a' is covered 7 times.

'b' is covered 3 times.

, which sums up to a total of 10 coverage.

Player2 wins and the total coverage will be 10 $mod(10^9 + 7) = 10$.

Given a game, are you able to determine the winner and the number of coverage?

Input Format

The first line contains the two players' names $Name_1$ and $Name_2$.

The second line contains two strings N, M.

$0 < |Name_1|, |Name_2| \leq 10$

Output Format

You should output 2 lines.

The first line contains the name of the winner.

The second line contains the total coverage of letters in the game.

Input Constraint

For 30% of the testcases, $0 < |N| \leq 5000$, $0 < |M| \leq 50$.

For 60% of the testcases, $0 < |N| \leq 10^4$, $0 < |M| \leq 10^4$.

For 100% of the testcases, $0 < |N| \leq 3 \times 10^7$, $0 < |M| \leq 3 \times 10^7$.

Sample Input

```
Arvin Patrick
```

```
abbaba aba
```

Sample Output

```
Patrick
10
```

Problem 5. Q-Moe no strings (20%)

Q-Moe likes to play with strings. *Q-Moe* defines a function Q on two strings A and B as follows::

* $Q(A, B)$ = the number of substrings of A such that $Moe(B, substring) = True$.

, where $Moe(A, B) = True$ means that there exists a permutation of the character set such that A is equal to B on this permutation. If not, Moe(A,B)=$False$. In other words, the two strings will share the same pattern. A trivial instance would be Moe("aabb" ,"ccdd") = $True$ since there is a valid permutation σ : $\begin{pmatrix} a & b & c & d \\ c & d & a & b \end{pmatrix}$ of the character set {a,b,c,d}. On this permutation σ, σ(a)=c, σ(b)=d.

Moe indicates two strings in the same form:

- Moe("a","a") = $True$
- Moe("a","b") = $True$
- Moe("a","aa") = $False$
- Moe("xyx","ada") = $True$
- Moe("dsa","ada") = $False$
- Moe("$aabb$","$xxyy$") = $True$
- Moe("$aabb$","$aaaa$") = $False$
- Moe("$qazwsxedcrfvtg$","$abcdefghijlkmn$") = $True$

Function Q represents the number of qualified substrings in A:

- Q("a","a") = 1
- Q("a","b") = 1
- Q("a","aa") = 0
- Q("$abcd$","ab") = 3
- Q("$abcd$","cc") = 0
- Q("$aabb$","ab") = 1
- Q("$aabb$","bb") = 2

In this problem, the character set would be {a,b,...,z}. Please output the Q value of two given strings.

Input Format

The first line contains a string A, and the second line contains a string B.
$0 < |A|, |B| \leq 10^7$.

All strings A and B consist of lowercase English alphabets $a - z$ only.

Output Format

You should output one line consisting of only one integer, which is the Q value of those substrings.

Input Constraint

For 11% of the testcases, the character set of B is $\{a\}$.

For another 22% of the testcases, $0 < |A|, |B| \leq 10^3$. Also, the character set of A, B is $\{a, b\}$.

For another 29% of the testcases, the character set of A, B is $\{a, b\}$.

For another 25% of the testcases, $0 < |A|, |B| \leq 10^6$.

For another 13% of the testcases, there are no other constraints.

Sample Input 1

```
abcd
ab
```

Sample Output 1

```
3
```

Sample Input 2

```
abcd
cc
```

Sample Output 2

```
0
```

Sample Input 3

```
aabb
ab
```

Sample Output 3

```
1
```

Hint

- The concept of Rabin-Karp may be helpful.
- Choosing prime numbers as the base of Rabin-Karp may effectively avoid collision.
- Try to come up with a better policy rather than naive comparison when "spurious hit" occurs.