Data Structure and Algorithm, Spring 2023

Homework 3

Red Correction Date: 05/06/2023 11:00

Due: 13:00:00, Thursday, May 25, 2023

TA E-mail: dsa_ta@csie.ntu.edu.tw

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- In Homework 3, the problem set contains a special Problem 0 (see below) and 5 other problems. The set is divided into two parts, the non-programming part (Problems 0, 1, 2, 3) and the programming part (Problems 4, 5).
- For problems in the non-programming part, you should combine your solutions in *one* PDF file. Your file should generally be legible with a white/light background—using white/light text on a dark/black background is prohibited. Your solution must be as simple as possible. At the TAs' discretion, solutions that are too complicated can be penalized or even regarded as incorrect. If you would like to use any theorem which is not mentioned in the classes, please include its proof in your solution. **If you use pseudocode to describe your algorithm, you need to briefly explain how your pseudocode works.**
- The PDF file for the non-programming part should be submitted to Gradescope as instructed, and you should use Gradescope to tag the pages that correspond to each subproblem to facilitate the TAs' grading. Failure to tag the correct pages of the subproblem can cost you a 20% penalty.
- For the programming part, you should have visited the *DSA Judge* (https://dsa2023.csie.org/) and familiarized yourself with how to submit your code via the judge system in Homework 0.
- For problems in the programming part, you should write your code in C programming language, and then submit the code via the judge system. Each day, you can submit up to 5 times for each problem. To encourage you to start early, we allow 10 times of submissions per day in the first week **(from 2023/05/04 to 2023/05/10)**. The judge system will compile your code with

```
gcc main.c -static -O2 -std=c11
```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with a program. You can also use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling. Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:
  - `gcc main.c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc main.c -O2 -std=c11 -g # use it with gdb`
- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$Late\ Score = \max\left(\left(\frac{5 \times 86400 - Delay\ Time(sec.)}{5 \times 86400}\right) \times Original\ Score, 0\right)$$

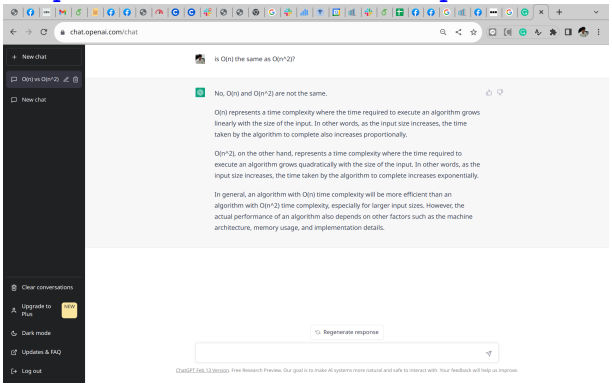
- If you have questions about HW2, please go to the discord channel and discuss (*strongly preferred*, which will provide everyone with a more interactive learning experience). If you really need an email answer, please follow the rules outlined below to get a fast response:
  - The subject should contain two tags, `"[hw2]"` and `"[Px]"`, specifying the problem where you have questions. For example, `"[hw2][P3] Is k in subproblem 3 an integer"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Non-Programming Part

Problem 0 - Proper References (0 pts)

For each problem below, please specify the references (the Internet URL you consulted with or the classmates/friends you discussed with) in your PDF submitted to Gradescope. If you have used chatGPT or similar tools, please provide a quick snapshot of your interaction with the tools. *You do not need to list the TAs/instructors.* If you finished any problem all by yourself (or just with the help of TAs/instructors), just say "all by myself." While we did not allocate any points to this problem, failure to complete this problem can lead to penalty (i.e. negative points). Examples are

- Problem 1: Alice (B86506002), Bob (B86506054), and https://stackoverflow.com/questions/982388/



- Problem 2: all by myself
- ...

Listing the references in this problem does *not* mean you could copy from them. We cannot stress this enough: *you should always write the final solutions alone and understand them fully.*

Problem 1 - Alexander loves sweet dumplings I (100 pts)

In the following problems, assume the size of the character to be k, which is not constant in the following problems. You can assume $k < n$.

1. (15 pts) Given a string s of length m. Explain how you can construct a string t of length n, such that s appears the most number of times in t, utilizing the prefix function of string s.

2. (15 pts) Given a string s of length n and corresponding q queries. Each query contains a set of four numbers: $\{l_1, r_1, l_2, r_2\}$. Derive an algorithm to determine whether the substring $s[l_1..r_1]$ is equal to the substring $s[l_2..r_2]$ with time complexity $O(n+q)$. Write down your algorithm in detail (pseudo code is not necessary), and analyze the time complexity of the algorithm. Please modify the original Rabin-Karp algorithm to solve this problem.

3. (30 pts) Given a string s of length n. Please design an algorithm with time complexity $O(n)$ to find all integers c, such that s can be divided into c equal substrings. Write down your algorithm in detail (pseudo code is not necessary), **prove the correctness**, and analyze the time complexity of the algorithm.

4. (30 pts) Given two strings s and key with lengths n and m respectively, find the smallest index i such that the substring $s[i..i+m-1]$ has a lexicographic order strictly smaller than key, with time complexity $O(n+m)$. Write down your algorithm in detail (pseudo code is not necessary), and analyze the time complexity of the algorithm. Your algorithm should be based on the original KMP algorithm, but with a modified definition of the prefix function.

5. (10 pts) Please provide a set of strings s and key that would produce incorrect result from the GPT's_Algorithm. Besides, explain how and where in GPT's_Algorithm they would introduce incorrect results.

GPT's_Algorithm

```
1  int find_string(char *s, char *key) {
2      int m = strlen(key);
3      int n = strlen(s);
4      int idx_s = 0;
5      while (idx_s <= n − m) {
6          int idx_key = 0;
7          while (idx_key < m && s[idx_s] <= key[idx_key]) {
8              if (s[idx_s] < key[idx_key]) {
9                  int idx_ans = idx_s − idx_key;
10                 return idx_ans;
11                 // the starting index of the answer
12                 // that is, s[idx_ans] ∼ s[idx_ans + m − 1] is the answer
13             }
14             idx_key++;
15             idx_s++;
16         }
17         if (idx_key == 0) {
18             idx_s++;
19         }
20     }
21     return −1; // not found
22 }
```

Appendix

The original problem description with a story can be found at the link below. Enjoy!

https://hackmd.io/@IE85Ixx3QjC19iPEZUp1dA/ryeJGJeVn

However, please base your answer on the problem description in this PDF.

Problem 2 - Sorting in Linear Time (100 pts)

1. (15 pts) You are given an array A with 6 elements $[152, 234, 57, 8, 601, 310]$. Please perform *radix sort* on A. Show your progress by presenting the content of A after each of the three rounds of radix sort.

For the problems listed below, we define an array to have a *range* of K if the array elements only have values within the interval $[0, K]$.

2. (25 pts) Given an array B of length N and range K, an inversion exist with two indices i and j, $i < j$, and $B[i] > B[j]$. Now, assume that you can make use of a data structure called Magic Counter. The magic counter C supports two functions: $insert(x)$ and $sum(R)$. Let M be the maximum value that will be inserted into the Magic Counter C[1].

   - $insert(x)$ increments the occurrences (i.e., the counter) of x in C by 1 in $O(\log M)$-time.
   - $sum(R)$ returns the sum of all the occurrences for all values from 0 to R in C in $O(\log M)$-time.

   Design an algorithm that counts the total number of inversions in B in $O(N \log K)$-time. You can use the two functions of Magic Counter C directly in your pseudo code.

For the problems below, we define a substring of a string as a consecutive sequence of characters within the string. For example, "AB", "BCD", and "E" are substrings of the string "ABCDE", while "AC" and "BCE" are not. In the following two problems, you are asked to consider *the number of substrings*. Here, substrings which start or end at different positions in the string are considered different and counted individually. For example, in "APPLE", there are two substrings "P", starting at the second position and at the third position, respectively. Both should be counted in *the number of substrings*.

3. (25 pts) Given a string of length N and a character σ, describe an algorithm to **calculate the number** of its substrings that do not contain the specific character σ in the character set, and show that it runs in $O(N)$-time. For example, if the string is "APPLE" and the character σ = 'P', the string has four substrings "A", "L", "E", and "LE" that do not contain the character 'P'.

[1] In fact, the magic counter can be implemented by data structures such as Fenwick Tree, you can find more details here.

4. (35 pts) Given a string of length N and a character set of size C (which include all the possible characters that are in the string), describe an algorithm to **calculate the number** of its substrings that do not contain **each** character in the character set and show that it runs in $O(C + N)$-time. That is, complete the task specified in the previous problem for each of the C characters in the set, but in $O(C + N)$-time.

Problem 3 - Fans of H7Lin (100 pts)

Hsin is a super big fan of H7Lin. After she learned the interesting data structure, disjoint set, in school, she would like to optimize the disjoint set in her own way. Specifically, she creates the disjoint set data structure that supports the following functions:

- `MAKE-SET(x)`: Create a new set with only x.
- `FIND-SET(x)`: Find out the ID of the set that contains x.
- `UNION(x,y)`: Merge the set that contains x and the set that contains y into one set. If x and y are in the same set originally, then nothing changes. Note that x can be equal to y.

Her implementation can be found here in this link, in which she applies a "randomized" way to union two sets in line 23. Besides, she uses a variable `cnt` to estimate the time it takes to run her code.

1. (15 pts) Try to generate a test case that makes the variable `cnt` printed in line 52 more than 120. Give the results of your test data when the program is executed (e.g. screenshots). Briefly explain how you achieve this using **at most three lines of text**. The test case should consist of 15 `UNION(x,y)`, while $0 \leq x \leq 15, 0 \leq y \leq 15\ \forall x, y$. Note that you cannot change the random seed, so all the values of `rand()` can be predicted. For example, the test case shown in the code can be expressed by:

   ```
   UNION(0, 1)
   UNION(1, 2)
   UNION(2, 3)
   UNION(3, 4)
   UNION(4, 5)
   UNION(5, 6)
   UNION(6, 7)
   UNION(7, 8)
   UNION(8, 9)
   UNION(9, 10)
   UNION(10, 11)
   UNION(11, 12)
   ```

```
UNION(0, 0)
UNION(1, 1)
UNION(2, 2)
```

After you generate the test case, Hsin is so angry, as she thinks running her code takes too much time. Therefore, she applies a more powerful randomized technique to union the sets. The implementation can be found in this link.

2. (20 pts) Try to generate a test case that makes the variable `cnt` printed in line 53 more than 120. Give the results of your test data when the program is executed (e.g. screenshots). Briefly explain how you achieve this with **at most five lines of text**. The test case should consist of 15 `UNION(x, y)`, while $0 \leq x \leq 15, 0 \leq y \leq 15$ $\forall x, y$.

   Note: In fact, the average case time complexity of this algorithm is much better than that of the previous one.

One day, as H7Lin has so many fans around the world, she would like to invite her fans to her country and play a game altogether. Hsin, of course, joins in the game. The game is held in a big cornfield, but there is no corn in it. The cornfield is a rectangle, and it consists of $n \times m$ grids, where n, m is the number of rows and the number of columns, respectively. The grid in the r-th row and the c-th column is denoted by $(r, c), 0 \leq r < n, 0 \leq c < m$. All the participants are in the grid $(0, 0)$ at the beginning.

In the following two problems, you can directly call these functions without any explanation: `MAKE-SET(x)`, `FIND-SET(x)`, and `UNION(x,y)` with the linked list representation, the union-by-size technique, and the path compression technique. Let $\alpha(N)$ is the inverse of the Ackermann function. From the textbook, we know that a sequence of M `MAKE-SET(x)`, `FIND-SET(x)`, and `UNION(x,y)` operations, N of which are `MAKE-SET(x)`, can be performed in worst-case time $O(M\alpha(N))$, when applying the union-by-size technique and the path compression technique.

3. (15 pts) H7Lin will do two types of operations:

   - `INSTALL(x,y,flag)`: If `flag` is equal to 0, then for all $0 \leq i < m$, install a bouncing device which allows a participant to bounce from the grid (x, i) to the grid (y, i), and install a bouncing device which allows a participant to bounce from the grid (y, i) to the grid (x, i). If `flag` is equal to 1, then install bouncing devices between the grid (i, x) and the grid (i, y) for all $0 \leq i < n$. For instance, after the operation `INSTALL(0,1,0)`, a participant can bounce from $(0, 3)$ to $(1, 3)$, but he/she cannot bounce from $(0, 3)$ to $(1, 2)$.

- `QUERY(x,y)`: H7Lin wants to know whether the participants can move from $(0, 0)$ to (x, y) through an arbitrary number of bounces with the installed bouncing devices (possibly 0).

There are Q operations in total. Please implement the two operations such that the Q operations run in $O(Q\alpha(n + m))$-time.

However, Hsin feels that it's so boring. To pursue extra excitement, whenever she bounces, she only wants to double-bounce with *two* bouncing devices. However, both of the two bouncing devices must bounce vertically or horizontally. For instance, in one "double-bounce", Hsin may bounce from $(0, 0)$ to $(4, 0)$ and bounce from $(4, 0)$ to $(2, 0)$. However, she **cannot** bounce from $(0, 0)$ to $(4, 0)$ and bounce from $(4, 0)$ to $(4, 1)$.

4. (25 pts) Please implement the following two operations:

   - `INSTALL-v2(x,y,flag)`: The definition is the same as `INSTALL(x,y,flag)` in the previous problem.
   - `QUERY-v2(x,y)`: H7Lin wants to know whether Hsin can use an arbitrary number of *double-bounces* from $(0, 0)$ to (x, y) utilizing the installed bouncing devices (possibly 0).

   If there are Q operations of `INSTALL-v2(x,y,flag)` and `QUERY-v2(x,y)`, it should run in $O(Q\alpha(n + m))$-time.

Nevertheless, the severe weather in the country where H7Lin lives makes the bouncing devices break down after a period of time.

5. (25 pts) Let's define the two operations:

   - `INSTALL-v3(x,y,flag,l,r)`: The way to install the bouncing devices is the same as `INSTALL(x,y,flag)` in problem 3. However, those bouncing devices only work in the time interval $[l, r]$. Note that for any given two time intervals (l_i, r_i) and (l_j, r_j) from the given `INSTALL(x,y,flag)` operations, the following condition is satisfied. Without losing generality, assume $l_i \leq l_j$. Then, either $r_i \geq r_j$ holds, i.e., the first time interval completely contain the second time interval, or $l_j \geq r_i$ holds, i.e., the two time intervals have no overlap. All time instances, i.e., any l or r, are non-negative integers.
   - `QUERY-v3(x,y,t)`: H7Lin wants to know whether the participants, excluding Hsin, can bounce from $(0, 0)$ to (x, y) at the time t through an arbitrary number of bouncing devices.

To be able to implement these operations, we can leverage an additional function for the disjoint set data structure:

- `UNDO()`: Remove the change caused by the last `UNION(x,y)`. You can assume that it runs in $O(\log N)$-time after N `MAKE-SET(x)`.

The operations will be executed as follows. First, Q `INSTALL-v3(x,y,flag,l,r)` operations are given to install all bouncing devices. Let T denote the maximum time. Then, T queries are given in order as follows: `QUERY`$(x_1, y_1, 1)$, `QUERY`$(x_2, y_2, 2)$, ..., `QUERY`$(x_{T-1}, y_{T-1}, T-1)$, `QUERY`(x_T, y_T, T).

Please implement the two operations `INSTALL-v3(x,y,flag,l,r)` and `QUERY-v3(x,y,t)`, and it should run in $O((T+Q)\log(n+m))$-time. Note that you can directly call the functions without any explanation, assuming the corresponding running time:

- `MAKE-SET(x)`, $O(1)$. Assume N `MAKE-SET(x)` are executed.
- `FIND-SET(x)`, $O(\log N)$
- `UNION(x,y)`, $O(\log N)$
- `UNDO()`, $O(\log N)$

Problem 4 - Rotating Magic (100 pts)

Problem Description

Remember the magical fairy in the house of Demeter Sphynx Abyssinian (DSA)? The fairy's magic spell can be represented by a string S, consisting of uppercase English alphabets of length of N. Each substring of S is called a segment of the spell, and a segment generates *magic effect* if it is exactly the same as the pattern P, which has a length of M. However, a segment may produce *the opposite effect* if it is a rotation of P. For example, given a string $A(A = c_1c_2...c_n)$ consisting of n alphabets, $c_i...c_n + c_1...c_{i-1}$ is considered a rotation of A, for any $i \in [2, n]$.

To help the fairy debug zir magic spell, can you calculate the total number of segments in zir spell that may generate either the original or the opposite magic effect?

Input

The first line contains two integers N and M. The second line contains the magic spell string S, which has length N. The third line contains the pattern P, which has length M.

Output

Output the answer as an integer in a single line.

Constraints

- $1 \leq M \leq N \leq 10^6$
- S and P consist of only uppercase English letters.

Subtask 1 (10 pts)

- $1 \leq N \leq 10^4$
- $1 \leq M \leq 10^2$

Subtask 2 (15 pts)

- $N = M$

Subtask 3 (25 pts)

- $1 \leq N \times M \leq 2 \times 10^7$

Subtask 4 (50 pts)

- No other constraints.

Sample Testcases

Sample Input 1

```
7 5
DEABCDE
ABCDE
```

Sample Output 1

```
3
```

Hints

Explanation of Sample 1:
Substrings of length M in S are $[DEABC, EABCD, ABCDE]$.
The first two are rotations of P and the last one is equal to P.

Problem 5 - Mega Knights (100 pts)

Problem Description

There are n mega knights initially located at n distinct positions. Each mega knight K_i ($1 \leq i \leq n$) can cause damage a_i (i.e., his attack points) if he attacks, and has h_i health points, representing the maximum damage the knight can bear.

When a mega knight K_a receives an instruction to attack another mega knight K_s (i.e., the target), he will gather *all knights at the same position as* K_a to move to the position of K_s to perform the attack. Because the attack has an area of effect, all knights at the same position as K_s suffer the attack, after which their health points are all reduced by the sum of the attack points of all attacking knights. Any mega knight whose health points is less or equal to 0 after the attack is considered dead. After the attack, the attacking mega knights stay at the position where K_s is located.

A dead mega knight cannot perform an attack, nor can it be targeted, even when an instruction is given to do so. When such an instruction is given, no attack or movement will happen. That is, in the above example, if either K_a or K_s is dead, nothing happens subsequently. In addition, when the attacking mega knight is located at the same position as the target, even when such an instruction is given, no attack or movement will happen either. That is, in the above example, if K_a and K_s are located at the same position before the attack, nothing happens subsequently.

You will be given m attacking instructions. Calculate how many times each mega knight successfully attacks, including those that the knight is on the instruction, and those that the knight is following others.

Input

The first line contains two positive integers n and m separated by a space – the number of mega knights and the number of attack rounds.
The second line contains n positive integers $h_1, h_2, \ldots, h_n$ separated by spaces – the health points of each mega knight.
The third line contains n positive integers $a_1, a_2, \ldots, a_n$, separated by spaces – the attack points of each mega knight.
The next m lines each contain two positive integers K_a and K_s separated by a space – the indices of attacking and the target mega knight.

Output

Output a line containing n integers separated by spaces. The i-th integer represents the total number of successful attacks made by the i-th mega knight.

Constraints

- $1 \leq n, m \leq 2 \times 10^5$
- $1 \leq a_i, h_i \leq 10^9$
- $1 \leq K_a, K_s \leq n$
- $K_a \neq K_s$

Subtask 1 (15 pts)

- $1 \leq n, m \leq 10^3$

Subtask 2 (20 pts)

- It is guaranteed that no mega knight will die.

Subtask 3 (65 pts)

- No other constraints

Sample Testcases

Sample Input 1

```
6 5
7 7 7 7 7 14
3 1 4 1 5 9
2 3
2 1
4 3
1 6
6 3
```

Sample Output 1

```
1 3 2 2 0 0
```

Sample Input 2

```
6 5
7 7 7 7 7 14
3 1 4 1 5 9
1 6
3 2
4 5
5 3
2 6
```

Sample Output 2

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