Data Structure and Algorithm, Spring 2024

Homework 3

Due: 13:00:00, Friday, May 17, 2024

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.
- 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 (including chatGPT) 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.
- In Homework 3, the problem set contains a special Problem 0 (see below) and 4 other problems. The set is divided into two parts, the non-programming part (Problems 0, 1, 2) and the programming part (Problems 3, 4).
- 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 texts on a dark/black background is prohibited. Your solution must be as simple as possible. At the TAs' discretion, solutions which 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.
- The PDF file for the non-programming part, including Problem 0 (Proper references), 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 tagging the correct pages of the subproblem in Problem 1 and Problem 2 can cost you a 20% penalty. And failure to tagging the correct pages of Problem 0 can cost you 5 points for each problem without reference.

- For problems in the programming part, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa2024.csie.org/.

  Each day, you can submit up to 5 times for each problem. 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 programs and/or 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, but possibly not M1/M2`
  - `gcc main.c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

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

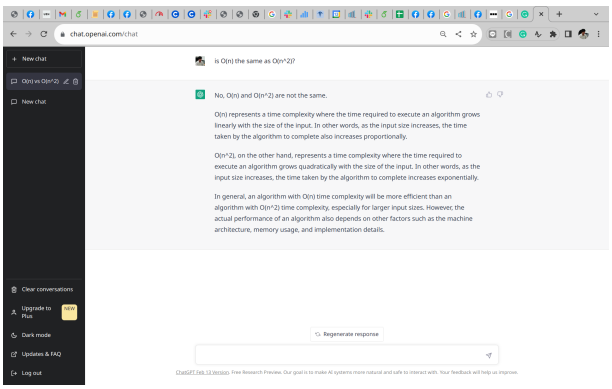
- If you have questions about HW3, please go to the Discord channel and discuss (*strongly preferred*, which will provide everyone with a more interactive learning experience). If you need an email answer, please follow the rules outlined below to get a fast response:

  - Please include the following tag in the subject of your email: `"[HW3.Px]"`, specifying the problem where you have questions. For example, `"[HW3.P2] Is k in subproblem 5 an integer?"` Adding the tag 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.

Problem 0 - Proper References (0 pts)

For each **non-programming and programming 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. For P0, mark all the pages where you have listed references, whether it is on the same page or on multiple pages. 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)
- Problem 2: all by myself
- Problem 3:



- Problem 4: https://stackoverflow.com/questions/982388/

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.*

For sub-problems in **Problem 1** and **Problem 2** listed below, unless stated otherwise, please ensure that your answers consist of the following components:

- Please describe your algorithm using **pseudo code** and provide a **brief explanation** of why it works. Your pseudo code for each sub-problem should be concise and easy to read, **consisting of no more than 30 lines**. Please note that overly complex or lengthy pseudo code may result in some penalties.
- Analyze the time complexity and extra-space complexity of your algorithm. Please note that if your algorithm is not efficient in terms of time or space, you may not receive full credit for your solution.

Problem 1 - Drive Safely, Attentively! (100 pts)

Story

Steve is a outstanding project manager in Dynamic Speed Autos. Ze is responsible for a brand new project named Driving Safety Alerting. This is a system that helps drivers to keep track of the surroundings and alerts drivers when there is danger. Each vehicle is set with a distinct positive integer value, which is called the **size** of the vehicle. However, their team encountered some troubles, so they invite you as a consultant to help them solve the problems.

1. (15pts) **Debugging Steve's Algorithm.** Steve utilizes the following pseudo code as their algorithm to sort the vehicles by their **size**. v is a 1-based array holding the *sizes* of the vehicles, and $v[i]$ denote the size of the i-th vehicle.

   Please give an input array v to this algorithm, showing them that it might not have sufficient performance for certain cases. You do not need to put together pseudo code for this problem.

```
SteveSort(v)
1   n = v.length
2   if n == 0
3       return EmptyArray
4   if n == 1
5       return v[1]
6   pv = v[⌊n/2⌋]
7   p1 = EmptyArray
8   p2 = EmptyArray
9   for i = 1 to n
10      if i == ⌊n/2⌋
11          continue
12      elseif v[i] < pv
13          append v[i] to p1
14      else
15          append v[i] to p2
16  return Concatenate(SteveSort(p1), pv, SteveSort(p2))
17  // Concatenate(a, b, c) concatenates thee three given arrays a, b, c.
18  // If any of a, b, c is an element, treat the element as arrays of size 1.
```

Your answer should meet the following requirements.

- Assume $v.length = 16$, and the values of elements in $v \in \{1, 2, ..., 16\}$. Give the content of v in this format: $[v[1], v[2], \ldots, v[16]]$. For example, $[13, 2, 3, 5, 4, 6, 7, 12, 16, 10, 11, 8, 1, 14, 15, 9]$.
- Your input should cause the algorithm to use at least 120 comparisons, i.e., the loop in line 9 should be executed at least 120 times.
- Write the number of comparisons when using your input.

2. (20 pts) **Documenting Specific Ambulance.**

Steve notices that in n vehicles, there is an ambulance that needs to be tracked. However, ze only remembers that the size of the ambulance is the k-th among the n vehicles. Can you design an algorithm which finds the vehicle whose size is the k-th among the n vehicles? Assume that $1 \leq k \leq n$. Your algorithm should run in $O(n)$-time **on average**, and uses **exactly** $O(1)$ extra space.

3. (20 pts) **Different Sections in Arrays.**

   In order to perform some investigation, Steve uses two *sorted* arrays $A = \{a_0, a_1, \ldots, a_{n_a-1}\}$ and $B = \{b_0, b_1, \ldots, b_{n_b-1}\}$, of lengths n_a and n_b, respectively. The elements in these two arrays represent the vehicle **sizes**, and they are distinct in each array. Ze needs to merge A and B into one sorted arrays. The following outlines the procedures:

   - Create a new array $C = \{c_0, c_1, \ldots, c_{n_c-1}\}$ of length $n_c = n_a + n_b$.
   - Let $c_k = a_k$ for $0 \leq k < n_a$, and $c_k = b_{k-n_a}$ when $n_a \leq k < n_a + n_b$.
   - Use bubble sort to sort array C.

   Design an algorithm to help Steve to *calculate minimum required number of swaps* to complete sorting array C. Your algorithm should run in $O(n_a + n_b)$-time and uses $O(1)$ extra space.

Data Swapping Analysis.

Steve designed a special CPU. The CPU supports various special instructions to swap two elements in an array. Ze wants to use this CPU to sort n vehicles according to their **sizes**. The array $A = \{a_0, a_1, \ldots, a_{n-1}\}$ stores the information, with each element representing the size of one of the n vehicles.

4. (15 pts) The first type of instruction can swap a_i, a_j for $0 \leq i < j < n$ and $j - i = 1$, i.e., swapping neighboring elements in an array. Please design an algorithm that *calculates* the **minimum number of swaps** required to sort array A, which stores the vehicle sizes, when using this type of instruction. Your algorithm should run in $O(n \log n)$-time, and you do not need to consider extra space complexity.

   Hint: What you have done in the previous sub-problem would help!

5. (30 pts) The second type of instruction can swap an arbitrary pair of elements a_i, a_j for $0 \leq i < j < n$. Please design an algorithm that *calculates* the **minimum number of swaps** required to sort array A, which stores the vehicle sizes, when using this type of instruction. Your algorithm should run in $O(n \log n)$-time, and you do not need to consider extra space complexity.

   Hint: What $O(n \log n)$-time pre-processing can you do before finding the number of swaps?

Story

After solving those troublesome technical issues, Steve's team is given the Divine Silver Award and recognized by their company, Dynamic Speed Autos. To show their appreciation, they invited you to their party, and you can Drink Some Apple Cider with them!

Fun : How many DSAs could you find in this problem?

Problem 2 - Delicate String Analysis (100 pts)

Story

After years of hard work, Steve has become the president of the Dertian Space Agency (DSA). One day, the Dertian Space Agency intercepted some messages sent by unknown creatures in space. Steve and his team decided to carefully analyze the patterns hidden within these messages.

In the following sub-problems, consider the following assumptions:

- Assume the character set forming the string to be Σ and the size of the set $d = |\Sigma|$ is NOT a constant.
- Assume $d < n$. n is separately defined in each sub-problem.
- Assume that a spurious hit never happens.
- The following assumes the use of the standard Rabin-Karp algorithm with modulo operation, introduced in the lecture.

1. (15 pts) In this sub-problem, you are given a string S of length n and an integer k. The latest CPU developed by DSA has a special instruction which removes the first k characters from S and then appends an input string of length k *of your choice*, P, to S. The resultant string is $S[k+1:n]P$ and will be stored back to S.

   Develop an $O(n)$-time algorithm to calculate the minimum number of executions of this special instruction, on the given string S, after which the resultant string is identical to the original S. The special instruction *must execute at least once*, i.e., the minimum number needs to be larger than zero. You do not need to consider extra-space complexity for this sub-problem.

   Hint: Each execution of the special instruction can append arbitrary k characters to the input. Therefore, with a sufficient number of executions of the special instruction, the original string can always be re-created with these newly added characters while the characters of the original string are entirely removed.

2. (25 pts) A dual-ended string pair is defined as follows. Given two strings S_1 and S_2, $|S_1| < |S_2|$. If S_1 is both a prefix and a suffix of S_2, (S_1, S_2) is called a dual-ended string pair.

   Given m different strings whose lengths are less than or equal to n. Modified from the original Rabin-Karp algorithm, please derive a $O(nm + m^2)$-time algorithm to determine

the number of dual-ended string pairs that exists among the given m strings. There is no need to analyze the extra-space complexity.

Story

While analyzing the messages from those unknown creatures, the DSA discovered an important characteristic in the way these beings transmit messages: perfection.

3. (30 pts) A string S achieves perfection when, after appending a number of arbitrary characters **at the end** of S, the resultant string S' reads the same forwards and backwards.

   Examples of such S' include: `noon`, `level`, `wasitacatisaw`. Now, given a string S of length n, design an algorithm to calculate the minimum number of characters that need to be appended to S to achieve perfection. Note that in the case that the string already achieves perfection, this minimum number is zero. Your algorithm should run in $O(n)$-time. You do not need to consider extra-space complexity.

Story

The Dertian Space Agency has developed a new method to encode a secret number, ensuring that communications cannot be easily deciphered by their enemies. Since you are DSA's most trusted friend, they are willing to share with you the decryption method.

4. (30 pts) Let S denote the given encrypted message. Any pattern P is *a good pattern* with respect to S if it has multiple occurrences in S, i.e., using the definition in the string matching problem, it has multiple *valid shifts*. The secret number is the length of the longest pattern among all such good patterns. The Dertian Space Agency now wants to streamline the decryption process. Please help them to design an algorithm to determine the secret number, given an encrypted message S of length n.

   Your algorithm should run in $O(n \log n)$-time and uses $O(q)$ extra-space, where q is the modulus used in the Rabin-Karp algorithm. You can assume initializing all elements in an array to 0 is an $O(1)$-time operation.

   Hint: You might want to use a binary array to store some information, where each index of the array corresponds to a "slot", and each slot can be in one of two states: empty or occupied. This concept can be further developed into a hash table, which is a data structure we will discuss in Week 14!

Problem 3 - Dynamic Spectrum Artistry (100 pts)

Problem Description

Story

One day, the employees at Dertian Space Agency grew tired of playing **Dungeons of Sacred Abyss** repeatedly, as they had already mastered its challenges. Recognizing their need for something new and exciting, Steve, the agency's president, reached out to Nextgen Technology Universal Company of Software (NTUCS) for help in creating a brand-new game.

Dynamic Spectrum Artistry is a fresh and innovative game from NTUCS. This game combines the enjoyable activity of coloring with an adventurous exploration of a colorful and vibrant universe. Its goal is to reignite the enthusiasm and creativity of the employees at Dertian Space Agency, providing them with an engaging and enjoyable gaming experience.

In this game, we have *blocks* which can be painted with different colors, and directed *arrows* that points from one block to another block. Let B denote the set of blocks and A denote the set of arrows in the game.

The colors of the blocks should satisfy the following rule. If $b_1, b_2 \in B, b_1 \neq b_2$, and there exist both a path from b_1 to b_2 and a path from b_2 to b_1, following the arrows in A, then $b_1.color = b_2.color$. On the other hand, if there exist either no path from b_1 to b_2 or no path from b_2 to b_1, following the arrows in A, then $b_1.color \neq b_2.color$.

Steve would like to determine *the number of colors* that needs to be used to color all the blocks in B. In addition, Steve would like to determine if there exist an acyclic path formed by arrows in A, such that the blocks included in this path have *all colors* used in B.

Reference Reading

The following provides information about *Strongly Connected Components* that may be helpful to solve this problem.

Strongly Connected Components Definition: [from Wikipedia] The strongly connected components (SCCs) of a directed graph form a partition into subgraphs that are themselves strongly connected. A directed graph is called *strongly connected* if there

is a path in each direction between each pair of vertices of the graph. That is, a path exists from the first vertex in the pair to the second, and another path exists from the second vertex to the first. If each SCC is contracted to a single vertex, the resulting graph G^{SCC} is a directed acyclic graph (DAG), the condensation of G.

Finding SCCs. There are multiple methods for finding SCCs in a directed graph. In the following section, we will introduce one such approach: the Kosaraju's algorithm, though you are free to use other methods to solve this problem. The pseudo code provided below is from *Chapter 20.5* of the textbook, which we strongly encourage you to read and understand. Below we will introduce the pseudo code and provide a brief explanation.

STRONGLY-CONNECTED-COMPONENTS(G)

1 Perform DFS(G) to compute finishing times $u.f$ for each vertex u
2 Compute the transpose graph $G^T = (V, E^T)$ by reversing the direction of all edges
3 Perform DFS(G^T), but in the main loop of DFS,
  consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 Output the vertices of each tree in the depth-first forest
  formed in line 3 as a separate SCC

Assume we are trying to find all SCCs in a given directed graph $G = (V, E)$. Assume G has strongly connected components $C_1, C_2, \ldots, C_k$, each of which is a set of vertices in the component. We can also define the finish time for a set of vertices U to be the maximum of the finish times of all of its vertices, i.e., $f(U) = \max_{u \in U}\{u.f\}$.

Let C and C' be distinct SCCs in G. If we have an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$, then we always have $f(C) > f(C')$[a]. This says if we have an edge going from a vertex in C to a vertex in C', then $f(C)$ is larger. Now if we reverse the edge directions to obtain $G^T = (V, E^T)$ (line 3), and we have an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$, then because of the reversal of the edge directions we have the opposite result $f(C) < f(C')$[b]. Moreover, between two distinct SCCs C_i and C_j there can only exist edges going either from C_i to C_j or from C_j to C_i, but not both. Otherwise, C_i and C_j would be within the same SCC.

Now we are ready to take a look at STRONGLY-CONNECTED-COMPONENTS. Line 1 and line 2 pre-processes G by calculating the finish times of all vertices and obtain the transpose graph G^T.

Line 3 deserves additional explanation. We visit the SCC with the largest finish time (in the DFS in line 1) $C_{\max}$. This implies there can only exist edges going into $C_{\max}$ from other SCC, but not the opposite (otherwise, C_{max} would not have the largest finish

time). If we start DFS traversal from any vertex in $C_{\max}$, it will visit all vertices in $C_{\max}$, but not moving to the other SCCs as there is no such edge. Then, DFS will move to the SCC with the second largest finish time $C_{\max-1}$, and in this case there can only exist edges going into $C_{\max-1}$ from any SCCs other than $C_{\max}$. DFS will now visit all vertices in $C_{\max-1}$ but not moving to other SCCs. The procedure continues until all SCCs are visited and the results are output in line 4.

Condensing the SCCs. We can transform the original graph G into a condensed graph G^{SCC} such that all vertices in the same SCC is merged into a single vertex. Any edge between vertices in the same SCC is now a self edge and removed. Multi-edges (duplicate edges) between SCCs are also removed.

Finding a path connecting all SCCs. We start by performing a topological sort on G^{SCC}. Let $v_1, v_2, \ldots, v_{|V^{SCC}|}$ denote the sorted sequence of vertices. If for any neighboring vertices in the sequence $v_i, v_{i+1}, 1 \leq i \leq |V^{SCC}| - 1$, there exist an edge (v_i, v_{i+1}), then we are able to establish a path connecting all vertices in G^{SCC}, i.e., connecting all SCCs in the original graph G.

[a] see the proof of Lemma 20.14 in the textbook
[b] see the proof of Corollary 20.15 in the textbook

Input

The first line contains 3 integers $|B|$, $|A|$, and `mode`. $|B|$ is the number of blocks, and $|A|$ is number of arrows in the game. `mode` has a value of either 1 or 2. If `mode` is 1, you only need to answer the first question. If `mode` is 2, you have to answer both questions. Each of the next $|A|$ lines contains two integers u_i and v_i, representing the starting block and ending block of the i-th arrow. The integers in the same line are separated by single white spaces.

Output

In the first line, please print the number of colors that needs to be used to color all the blocks in B. The second line is only required if `mode` is 2. If so, in the second line, indicate whether there exist an acyclic path formed by arrows in A, such that the blocks included in this path have all colors used in B. If the path does not exist, print 0; otherwise, print 1.

Constraints

- $0 \leq |A| \leq \min(|B| \cdot (|B| - 1), 10^6)$
- $1 \leq u_i, v_i \leq |B|$
- There are no duplicated arrows.
- `mode` $\in \{1, 2\}$
- Time Limit: 3 s
- Memory Limit: 262144 KB

Subtasks

Subtask 1 (15 pts)

- $1 \leq |B| \leq 10^3$
- `mode` $= 1$

Subtask 2 (25 pts)

- $1 \leq |B| \leq 10^5$
- `mode` $= 1$

Subtask 3 (15 pts)

- $1 \leq |B| \leq 10^3$
- `mode` $= 2$
- The test case in this subtask is the same as in subtask 1; the only difference is the `mode`.

Subtask 4 (45 pts)

- $1 \leq |B| \leq 10^5$
- `mode` $= 2$
- The test case in this subtask is the same as in subtask 2; the only difference is the `mode`.

Sample Testcases

Sample Input 1

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

Sample Output 1

```
2
1
```

Explanation for Sample 1

Figure 2.1

Block 1 can reach block 2, and vice versa; hence, blocks 1 and 2 should be painted the same color. Conversely, although block 2 can reach block 3, the reverse is not true; therefore, blocks 2 and 3 should NOT be painted the same color. You have used a total of 2 colors. As there exists a path that traverses all colors ($2 \to 3$), the answer to the second problem is 1.

Sample Input 2

```
5 5 2
1 2
2 3
3 1
2 4
5 4
```

Sample Output 2

```
3
0
```

Explanation for Sample 2

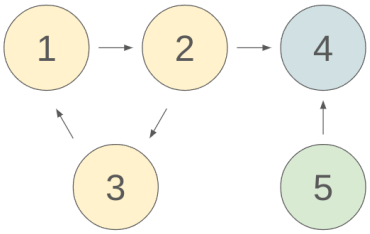


Figure 2.2

Block 1 can reach block 2 (1→2), and vice versa (2→3→1); thus, blocks 1 and 2 should be painted the same color. Similarly, blocks 2 and 3, as well as blocks 1 and 3, should also be painted the same color, resulting in colors being shared among blocks 1, 2, and 3. Additionally, blocks 4 and 5 must each be painted a different color from one another. You have used a total of 3 colors. Since there is no path that traverses all colors, the answer to the second problem is 0.

Problem 4 - Data Structures and Anticheating (100 pts)

Problem Description

Story

After prior attempts in developing games, Nextgen Technology Universal Company of Software (NTUCS) unfortunately lost too much money and decided to use its excellence in Data Structures and Algorithm technology to develop other kinds of software.

Meanwhile, cheating is unfortunately found in homework on Dense Structures of Asphalt (DSA) class in New Tempura University (NTU). It is suspected that a pair of submissions appeared to be cheating.

A case of cheating usually involves a source submission and a copier submission. Each submission can be represented by a string consisting of lower-case letters, i.e., $\Sigma = \{a, b, \ldots, x, y, z\}$, except in subtask 2, where $\Sigma = \{a\}$. Thus, in the following, we will use the term "submission" and "string" interchangeably.

First, we define *hit count* $\mathfrak{H}(P, Q)$ of an input pattern P and a reference string Q as the number of occurrences of P in Q. This is the same as finding the number of valid shifts in the string matching problem, given by

$$\mathfrak{H}(P, Q) = |\{s : 0 \leq s \leq |Q| - |P| \text{ and } Q[s+1 : s+|P|] = P\}| \ .$$

Next, we define *plagiarism likelihood* $\mathfrak{L}(R)$ of a string R as the number of substrings of R such that they are both the prefix and the suffix of R. This is equivalent to finding the number of prefixes of R that are also the suffixes of R, given by

$$\mathfrak{L}(R) = |\{k : 1 \leq k \leq |R| \text{ and } R[1 : k] \sqsupset R\}| \ .$$

Note that $\mathfrak{L}(R)$ is at least 1 because the input string R itself is both the prefix and the suffix of itself.

Finally, combining these two definitions, we can define the *similarity score* $\mathfrak{S}(I, S)$ of an input string I and a source submission S as follows:

$$\mathfrak{S}(I, S) = \mathfrak{H}(I, S) \cdot \mathfrak{L}(I) \ .$$

Nextgen Technology Universal Company of Software does not have a sufficient number of software engineers so they ask for your help. In this problem, for a given pair of copier submission C and source submission S, please output the similarity scores of all prefixes of C (as input string) and source submission S as follows: $\mathfrak{S}(C[1:1], S), \mathfrak{S}(C[1:2], S), \ldots, \mathfrak{S}(C[1:|C|], S)$.

Input

The first line has the string S representing the source submission. The second line has the string C representing the copier submission.

Output

Print $|C|$ lines. In the i-th line, $1 \leq i \leq |C|$, output the similarity score $\mathfrak{S}(C[1:i], S)$.

Constraints

- $0 < |S| \leq 5 \times 10^6$
- $0 < |C| \leq 5 \times 10^6$
- Time Limit: 3 s
- Memory Limit: 262144 KB

Subtasks

Subtask 1 (10 pts)

- $|S| \leq 100$

Subtask 2 (10 pts)

- The character set Σ to form S and C only has character 'a', i.e., $\Sigma = \{a\}$ and $|\Sigma| = 1$.

Subtask 3 (20 pts)

- $|S| \leq 5000$

Subtask 4 (60 pts)

- No other constraints.

Sample Testcases

Sample Input 1	Sample Output 1
aaaaa aaa	5 8 9

Explanation for Sample 1

- Prefix `a`: The **hit count** is 5 as `a` appeared in `aaaaa` 5 times. The **plagiarism likelihood** is 1 since `a` is both a prefix and a suffix. The **similarity score** is thus 5.
- Prefix `aa`: The **hit count** is 4 as `aa` appeared in `aaaaa` 4 times. The **plagiarism likelihood** is 2 since `a` and `aa` are both prefixes and suffixes. The **similarity score** is thus 8.
- Prefix `aaa`: The **hit count** is 3 as `aaa` appeared in `aaaaa` 3 times. The **plagiarism likelihood** is 3 since `a`, `aa` and `aaa` are all prefixes and suffixes. The **similarity score** is thus 9.
- This input satisfies the constraint of subtask 2.

Sample Input 2	**Sample Output 2**
`ntucsiedsa` `csie`	`1` `1` `1` `1`

Explanation for Sample 2

- Prefix `c`: The **hit count** is 1 as `c` appeared in `ntucsiedsa` 1 times. The **plagiarism likelihood** is 1 since only `c` is both a prefix and a suffix. The **similarity score** is thus 1.
- Prefix `cs`: The **hit count** is 1 as `cs` appeared in `ntucsiedsa` 1 times. The **plagiarism likelihood** is 1 since only `cs` is both a prefix and a suffix. The **similarity score** is thus 1.
- Prefix `csi`: The **hit count** is 1 as `csi` appeared in `ntucsiedsa` 1 times. The **plagiarism likelihood** is 1 since only `csi` is both a prefix and a suffix. The **similarity score** is thus 1.
- Prefix `csie`: The **hit count** is 1 as `csie` appeared in `ntucsiedsa` 1 times. The **plagiarism likelihood** is 1 since only `csie` is both a prefix and a suffix. The **similarity score** is thus 1.

Sample Input 3	Sample Output 3
`baabaab`	`3`
`baab`	`2`
	`2`
	`4`

Explanation for Sample 3

- Prefix `b`: The **hit count** is 3 as `b` appeared in `baabaab` 3 times. The **plagiarism likelihood** is 1 since only `b` is both a prefix and a suffix. The **similarity score** is thus 3.
- Prefix `ba`: The **hit count** is 2 as `ba` appeared in `baabaab` 2 times. The **plagiarism likelihood** is 1 since only `ba` is both a prefix and a suffix. The **similarity score** is thus 2.
- Prefix `baa`: The **hit count** is 2 as `baa` appeared in `baabaab` 2 times. The **plagiarism likelihood** is 1 since only `baa` is both a prefix and a suffix. The **similarity score** is thus 2.
- Prefix `baab`: The **hit count** is 2 as `baab` appeared in `baabaab` 2 times. The **plagiarism likelihood** is 2 since `b` and `baab` are both prefixes and suffixes. The **similarity score** is thus 4.

Hints

- Prefix = Suffix? Sounds familiar, doesn't it?