

Задача А. Подстроки

Имя входного файла: `substr.in`
Имя выходного файла: `substr.out`
Ограничение по времени: 2 секунды
Ограничение по памяти: 256 мегабайт

Даны K строк из маленьких латинских букв. Требуется найти их наибольшую общую подстроку.

Формат входного файла

В первой строке число K ($1 \leq K \leq 10$). В следующих K строках — собственно K строк (длины строк от 1 до 10 000).

Формат выходного файла

Наибольшая общая подстрока.

Примеры

<code>substr.in</code>	<code>substr.out</code>
3 abacaba mycabarchive acabistrue	<code>cab</code>

Задача В. Частотный анализ

Имя входного файла: *стандартный ввод*
Имя выходного файла: *стандартный вывод*
Ограничение по времени: 3 секунды
Ограничение по памяти: 256 мегабайт

Будем бороться за частоту наших рядов!

Рассмотрим массив целых чисел $a_1, a_2, \dots, a_n$, а также набор различных значений, встречающихся в нём: $b_1, b_2, \dots, b_k$.

Назовём массивом частот последовательности a массив $c_1, c_2, \dots, c_k$, где c_i равно количеству вхождений b_i в массив a .

Дано два массива. Требуется выбрать пару их интервалов одинаковой длины такую, что массивы частот этих интервалов совпадают после сортировки.

Среди всех таких пар требуется выбрать как можно более длинную.

Формат входного файла

В первой строке ввода записано целое число n_1 — количество элементов первого массива. Во второй строке записано n_1 целых чисел — сам массив.

В третьей строке ввода записано целое число n_2 — количество элементов второго массива. Во четвёртой строке записано n_2 целых чисел — сам массив.

Элементы массивов — целые числа от 0 до 10^9 включительно. Числа n_1 и n_2 строго положительны.

Формат выходного файла

На первой строке выведите одно целое число: максимально возможную длину интервалов L . На второй строке выводите номера начальных элементов интервалов в первом и втором массиве соответственно.

Элементы массивов нумеруются с единицы.

Если оптимальных ответов больше одного, выведите любой из них.

Примеры

стандартный ввод	стандартный вывод
5 1 2 3 4 5 3 10 10 20	2 1 2
3 1 1 1 3 1 2 1	1 1 1

Задача С. Палиндромы

Имя входного файла: `palindrome.in`
Имя выходного файла: `palindrome.out`
Ограничение по времени: 0.5 секунд
Ограничение по памяти: 256 мегабайт

Строка называется палиндромом, если она одинаково читается как слева направо, так и справа налево. Например, `abba` — палиндром, а `омах` — нет. Для строки α будем обозначать $\alpha[i..j]$ ее подстроку длины $j - i + 1$ с i -й по j -ю позицию включительно (позиции нумеруются с единицу). Для заданной строки α длины N ($1 \leq N \leq 100\,000$) требуется подсчитать число q пар (i, j) , $1 \leq i < j \leq n$, таких что $\alpha[i..j]$ является палиндромом.

Формат входного файла

Входной файл содержит одну строку α длины N , состоящую из маленьких латинских букв.

Формат выходного файла

В выходной файл выведите искомое число q .

Примеры

<code>palindrome.in</code>	<code>palindrome.out</code>
<code>aaa</code>	3
<code>abba</code>	2
<code>омах</code>	0

Задача D. Плохое хеширование

Имя входного файла: `bad-hashing.in`
Имя выходного файла: `bad-hashing.out`
Ограничение по времени: 2 секунды
Ограничение по памяти: 256 мегабайт

Маленький мальчик Петя решает задачу про строки. Не будем останавливаться подробно на её условии. Для нас важно только то, что задача сводится к следующей: дано несколько строк, состоящих исключительно из маленьких букв английского алфавита, и нужно брать пары строк и проверять, равны ли они.

Петя придумал, как делать это быстро: сначала он посчитал *хеш-функцию* от каждой строки, а затем, когда нужно сравнить две строки, просто сравнивает значения хеш-функции от этих строк. Конечно, если значения хеш-функции различны, строки тоже различны. А вот если значения одинаковы, это ещё не гарантирует равенства самих строк.

Мы хотим *взломать* Петино решение, то есть придумать такие две различные строки, что значения хеш-функции от них одинаковы. Чтобы сделать это, разберёмся, что за функцию Петя использует для хеширования.

При ближайшем рассмотрении оказалось, что Петя реализовал *полиномиальный хеш* от строки. Полиномиальный хеш задаётся множителем p и модулем q . Для пустой строки ε значение хеш-функции $h(\varepsilon) = 0$, а для любой строки S и любого символа c хеш-функция рекуррентно определяется как $h(S+c) = (h(S) \cdot p + \text{code}(c)) \bmod q$. Здесь $\text{code}(c)$ — это ASCII-код символа c . Как известно, коды маленьких букв английского алфавита идут подряд: $\text{code}('a') = 97$, $\text{code}('b') = 98$, ..., $\text{code}('z') = 122$. Можно выписать и нерекуррентную формулу: если строка $S = s_1 s_2 \dots s_n$, то $h(S) = (\text{code}(s_1) \cdot p^{n-1} + \text{code}(s_2) \cdot p^{n-2} + \dots + \text{code}(s_n) \cdot p^0) \bmod q$.

Это достаточно распространённый метод хеширования, однако Петя не подумал о том, что его решение могут взломать, и допустил две существенные ошибки при выборе p и q . Во-первых, модуль q слишком мал, он равен всего лишь 2^{32} (Петя просто считает значение хеш-функции в 32-битном целом беззнаковом типе данных и не обращает внимания на переполнение). Во-вторых, при этом множитель p чётный.

Зная множитель p , взломайте решение Пети.

Формат входного файла

Первая строка ввода содержит целое число p — множитель полиномиального хеширования ($0 < p < 2^{32}$). Гарантируется, что p чётно.

Формат выходного файла

В первых двух строках выведите две различные строки S и T , для которых $h(S) = h(T)$. Строки должны состоять исключительно из маленьких букв английского алфавита (ASCII-коды 97–122) и иметь длину от 1 до 100 000 символов. Заметим, что длины строк не обязательно должны совпадать. Если возможных ответов несколько, разрешается вывести любой из них.

Примеры

<code>bad-hashing.in</code>	<code>bad-hashing.out</code>
4	ae ba
1000	vabavydw budqhmng

Пояснение

В первом примере $h(S) = (97 \cdot 4 + 101) \bmod 2^{32} = 489$ и

$$h(T) = (98 \cdot 4 + 97) \bmod 2^{32} = 489.$$

Во втором примере $h(S) = 118\,097\,098\,097\,118\,121\,100\,119 \bmod 2^{32} = 834\,470\,743$ и

$$h(T) = 98\,117\,100\,113\,104\,109\,110\,103 \bmod 2^{32} = 834\,470\,743.$$