Encryption kit

[Due to SPOJ restrictions, this problem has been modified with respect to the original version used in the Argentinian Programming Tournament of 2014 in order to have multiple test cases per input file. The original version of this problem (in Spanish) can be found at http://dc.uba.ar/events/icpc/download/problems/tap2014-problems.pdf]

Karina is developing a powerful encryption protocol. To make it available to a wider audience, she wants to implement a tool-kit to make it easier to use. In particular, she requires a tool capable of efficiently performing a complicated operation over strings.

The positions in a string are numbered from left to right with consecutive natural numbers from **1** to the length of the string. The operation Karina needs to implement is specified by four positions in the string **i ≤ j < k ≤ l**, and consists of three steps:

1) interchange the substring that goes from position **i** to position **j** inclusive, with the substring that goes from position **k** to position **l**, inclusive;

2) reverse both substrings individually;

3) change all the characters of each substring into the next character in the alphabet. This is, every **'a'** turns into a **'b'**, every **'b'** turns into a **'c'**, etc. For the purposes of this operation we consider the alphabet to be circular, so that every **'z'** turns into an **'a'** after the operation is performed on it.

For example, let's take the string "alazareselfacil" and the positions **i = 3, j = 5, k = 8** and **l = 15**, so that the two affected substrings are "aza" and "selfacil". After the first step (interchanging) the resulting string is "al**selfacil**re**aza**". After the second step (reversing) the result is "al**licafles**re**aza**". Finally, after the third step (changing the characters to the ones following them) we obtain "al**mjdbgmft**re**bab**".

The protocol developed by Karina applies the operation described above on a string, then applies another operation on the resulting string, and so on and so forth, always applying new operations on the last obtained result. Karina needs to know what is the resulting string after all operations are performed, but unfortunately her rudimentary knowledge of the programming language R is only enough to implement a very inefficient algorithm. You are therefore required to implement a version that can efficiently handle many operations over long strings.

Input

The first line contains an integer number **T**, the number of test cases (**1 ≤ T ≤ 100**). **T** test cases follow.

The first line of each test case contains the initial string **S**, composed of between **2** and 10^5 lower-case letters of the English alphabet, and an integer **N**, representing the number of

operations that are to be performed (**1 ≤ N ≤ 10^5**).

The following **N** lines contain the description of one operation each, given by four integers **I, J, K** and **L** representing the four positions in the string for the operation explained above (**1 ≤ I ≤ J < K ≤ L ≤ |S|** with **|S|** the length of the string **S**).

Output

For each test case, print a single line containing a string representing the resulting string obtained after succesively applying all the operations in the input to the given initial string.

Example

Input:

```
4
alazareselfacil 1
3 5 8 15
alazareselfacil 2
3 5 8 15
3 5 8 15
aa 1
1 1 2 2
zabcdefghi 5
1 1 10 10
1 5 6 10
2 4 7 9
1 1 2 10
1 8 9 10
```

Output:

```
almjdbgmftrebab
alcbcfsugnbgekn
bb
defghgjklm
```