6^{O} Treino para Alunos da UFPR

8 de Março de 2013

Sevidor BOCA:
`http://maratona.c3sl.ufpr.br/boca/`

Organizadores:
Vinicius Kwiecien Ruoso, Ricardo Tavares de Oliveira e Flávio Zavan

Lembretes:

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova.
- A correção é automatizada, portanto, siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa. Deve-se considerar entradas e saídas padrão.
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. As soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Teste seu programa antes de submetê-lo. A cada problema detectado (erro de compilação, erro em tempo de execução, solução incorreta, formatação imprecisa, tempo excedido . . .), há penalização de 20 minutos. O tempo é critério de desempate entre duas ou mais equipes com a mesma quantidade de problemas resolvidos.
- Utilize o *clarification* para dúvidas da prova. Os juízes podem opcionalmente atendê-lo com respostas acessíveis a todos.

A: Pedido de Desculpas

File: desculpa.[c|cpp|java|pas]

Cuca saiu para jogar futebol com os amigos e esqueceu do encontro que tinha com a namorada. Ciente da mancada, Cuca deseja elaborar um pedido especial de desculpas. Resolveu então enviar flores e usar o cartão da floricultura para escrever um pedido especial de desculpas.

Cuca buscou na internet um conjunto de frases bonitas contendo a palavra "desculpe" (que pode ocorrer mais de uma vez na mesma frase). No entanto, o cartão da floricultura é pequeno, e nem todas as frases que Cuca colecionou poderão ser aproveitadas.

Cuca quer aproveitar o espaço do cartão, onde cabe um número limitado de caracteres, para escrever um sub-conjunto das frases coletadas de modo que apareça o máximo de vezes possível a palavra "desculpe".

Escreva um programa que, dados o número de caracteres que cabem no cartão e a quantidade de frases coletadas (com os respectivos comprimentos e os números de ocorrências da palavra "desculpe"), determine o número máximo de vezes que a palavra aparece, utilizando apenas as frases colecionadas, sem repetí-las.

Entrada

A entrada é constituída de vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros C e F indicando respectivamente o comprimento do cartão em caracteres ($8 \leq C \leq 1000$) e o número de frases coletadas ($1 \leq F \leq 50$). Cada uma das F linhas seguintes descreve uma frase coletada. A descrição é composta por dois inteiros N e D que indicam respectivamente o número de caracteres na frase ($8 \leq N \leq 200$) e quantas vezes a palavra "desculpe" ocorre na frase ($1 \leq D \leq 25$).

O final da entrada é indicado por $C = F = 0$.

Saída

Para cada caso de teste seu programa deve produzir três linhas na saída. A primeira identifica o conjunto de teste no formato *Teste n*, onde n é numerado a partir de 1. A segunda linha deve conter o máximo número de vezes que a palavra "desculpe" pode aparecer no cartão, considerando que apenas frases coletadas podem ser utilizadas, e cada frase não é utilizada mais de uma vez. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de entrada	**Exemplo de saída**

```
200 4
100 4
100 1
120 2
80 5
40 3
10 1
10 1
20 2
0 0
```
	```
	Teste 1
	9

	Teste 2
	4
	```

B: Unary

`File: unary.[c|cpp|java|pas]`

Unary is a minimalistic Brainfuck dialect in which programs are written using only one token.

Brainfuck programs use 8 commands: ”+”, -”, ”[”, ”]”, ≪”, ≫”, ”.”and ”,”(their meaning is not important for the purposes of this problem). Unary programs are created from Brainfuck programs using the following algorithm. First, replace each command with a corresponding binary code, using the following conversion table:

- ’<’ → 1000
- ’>’ → 1001
- ’+’ → 1010
- ’−’ → 1011
- ’.’ → 1100
- ’,’ → 1101
- ’[’ → 1110
- ’]’ → 1111

Next, concatenate the resulting binary codes into one binary number in the same order as in the program. Finally, write this number using unary numeral system - this is the Unary program equivalent to the original Brainfuck one.

You are given a Brainfuck program. Your task is to calculate the size of the equivalent Unary program, and print it modulo 1000003 ($10^6 + 3$).

Input

The first line of the input contains the integer T ($1 \leq T \leq 100$), the number of test cases. Each of the following T lines contains a test case, a program in brainfuck. Each line will contain between 1 and 100 characters, inclusive. There are no invalid characters.

Output

Output, for each test case, the size of the equivalent Unary program modulo 1000003 (10^6+3).

Sample Input	**Sample Output**
`2`	`220`
`,.`	`61425`
`++++[>,.<-]`	

Note

To write a number n in unary numeral system, one simply has to write 1 n times. For example, 5 written in unary system will be 11111.

In the first example replacing Brainfuck commands with binary code will give us 1101 1100. After we concatenate the codes, we’ll get 11011100 in binary system, or 220 in decimal. That’s exactly the number of tokens in the equivalent Unary program.

C: All Discs Considered

`File: discs.[c|cpp|java|pas]`

Operating systems are large software artefacts composed of many packages, usually distributed on several media, e.g., discs. You probably remember the time when your favourite operating system was delivered on 21 floppy discs, or, a few years later, on 6 CDs. Nowadays, it will be shipped on several DVDs, each containing tens of thousands of packages.

The installation of certain packages may require that other packages have been installed previously. Therefore, if the packages are distributed on the media in an unsuitable way, the installation of the complete system requires you to perform many media changes, provided that there is only one reading device available, e.g., one DVD-ROM drive. Since you have to start the installation somehow, there will of course be one or more packages that can be installed independently of all other packages.

Given a distribution of packages on media and a list of dependences between packages, you have to calculate the minimal number of media changes required to install all packages. For your convenience, you may assume that the operating system comes on exactly 2 DVDs.

Input

The input contains several test cases. Every test case starts with three integers N1, N2, D. You may assume that $1 \leq N_1, N_2 \leq 50000$ and $0 \leq D \leq 100000$. The first DVD contains N_1 packages, identified by the numbers $1, 2, ..., N_1$. The second DVD contains N_2 packages, identified by the numbers $N_1+1, N_1+2, ..., N_1+N_2$. Then follow D dependence specifications, each consisting of two integers x_i, y_i. You may assume that $1 \leq x_i, y_i \leq N_1+N_2$ for $1 \leq i \leq D$. The dependence specification means that the installation of package x_i requires the previous installation of package y_i. You may assume that there are no circular dependences. The last test case is followed by three zeros.

Output

For each test case output on a line the minimal number of DVD changes required to install all packages. By convention, the DVD drive is empty before the installation and the initial insertion of a disc counts as one change. Likewise, the final removal of a disc counts as one change, leaving the DVD drive empty after the installation.

Sample Input	Sample Output

3 2 1
1 2
2 2 2
1 3
4 2
2 1 1
1 3
0 0 0
	3
	4
	3

D: HQ9+

`File: hq9+.[c|cpp|java|pas]`

HQ9+ is a joke programming language which has only four one-character instructions:

- "H"prints "Hello, World!",
- "Q"prints the source code of the program itself,
- "9"prints the lyrics of "99 Bottles of Beer"song,
- "+"increments the value stored in the internal accumulator.

Instructions "H"and "Q"are case-sensitive and must be uppercase. The characters of the program which are not instructions are ignored.

You are given a program written in HQ9+. You have to figure out whether executing this program will produce any output.

Input

The first line of the input contains the integer T ($1 \leq T \leq 100$), the number of test cases. Each of the following T lines contains a test case, a program in HQ9+. Each line will contain between 1 and 100 characters, inclusive. ASCII-code of each character will be between 33 (exclamation mark) and 126 (tilde), inclusive.

Output

For each test case, output "YES", if executing the program will produce any output, and "NO"otherwise.

Sample Input	**Sample Output**

`2`
`Hi!`
`Codeforces`
	`YES`
	`NO`

Note

In the first case the program contains only one instruction - "H", which prints "Hello, World!".

In the second case none of the program characters are language instructions.

E: New Arena Password

`File: arena.[c|cpp|java|pas]`

You are a huge fan of an online programming contest called SRM (Special Round Match). To participate in an SRM contest, you must first download an applet called Arena, log in to the Arena by entering your username and password, and start competing.

Recently, to avoid hackers' attacks on the Arena, SRM imposes a new rule for the users' passwords. From now on, the first K characters of each user's password must match its last K characters. In this way, if someone enters a password with different first and last K characters repeatedly, it can be considered an attack from hackers.

However, you love your old password and do not want to change many characters from it. You are given a string S representing your old password, and an integer K. Find the minimum number of characters of S that must be changed so that the substring containing the first K characters is equal to the substring containing the last K characters.

Input

Each test case is described by a line containing an integer K and a string S ($1 \leq |S| \leq 50$, $1 \leq K \leq |S|$). S will contain only lowercase English letters (a - z).

The last test case is followed by a line containing the number zero.

Output

For each test case, print the minimum number of characters of S that must be changed.

Sample Input	Sample Output

`5 topcoderopen`
`4 puyopuyo`
`3 loool`
`5 arena`
`7 amavckdkz`
`0`
	`3`
	`0`
	`1`
	`0`
	`5`