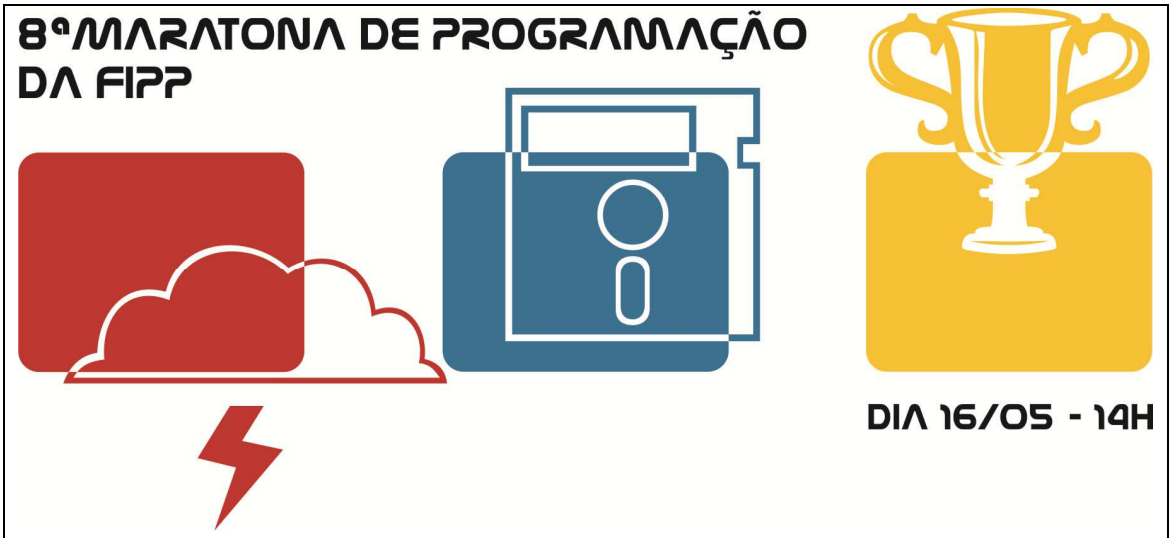
8ª MARATONA DE PROGRAMAÇÃO DA FIPP



DIA 16/05 - 14H

CADERNO DE PROBLEMAS

16/05/2012 – 14:00 às 18:00

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 6 problemas, sendo que 2 deles estão descritos em inglês.

1. É permitido o uso de material impresso, livros, apostilas e dicionários bem como lápis, caneta ou lapiseira, borracha, régua e papel para rascunho. O acesso à internet é bloqueado durante a realização da prova. A ajuda do ambiente de desenvolvimento como C, C++ ou Java pode ser utilizada.
2. A correção das resoluções dos problemas será automatizada por meio do sistema de submissão eletrônica BOCA, baseada nos resultados obtidos com uma série de execuções dos algoritmos de cada equipe.
3. Siga atentamente as exigências da tarefa quanto ao formato da entrada e saída do seu algoritmo. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
4. Os problemas não estão ordenados, neste caderno, por ordem de dificuldade. Procure resolver primeiro os problemas mais fáceis.
5. Utilize os nomes indicados nos problemas para nomear os seus arquivos-fonte, de acordo com a linguagem de programação utilizada.
6. Os problemas devem ser resolvidos utilizando o raciocínio entrada-processamento-saída, ou seja, não é necessário armazenar toda a saída para exibi-la.

Observações:

Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (teclado) e escritos na saída padrão (tela). Utilize as funções padrão para entrada e saída de dados:

- em C: `scanf`, `getchar`, `printf`, `putchar`;
- em C++: as mesmas de C ou os objetos `cin` e `cout`;
- em Java:
```
Scanner sc = new Scanner(); int i = sc.nextInt();
String s = sc.next(); float f = sc.nextFloat();
System.out.println(); System.out.printf("%d",i);
```

Em C ou C++: use `sprintf(str,"%d",num);` ao invés de `itoa(num,str,10);`

Faculdade de Informática de Presidente Prudente

http://www.unoeste.br/fipp/maratona

Robô Colecionador (1 – vermelho)

Arquivo fonte: *robo.c*, *robo.cpp* ou *robo.java*

Um dos esportes favoritos na Robolândia é o Rali dos Robôs. Este rali é praticado em uma arena retangular gigante de N linhas por M colunas de células quadradas. Algumas das células estão vazias, algumas contêm figurinhas da Copa (muito apreciadas pelas inteligências artificiais da Robolândia) e algumas são ocupadas por pilastras que sustentam o teto da arena. Em seu percurso os robôs podem ocupar qualquer célula da arena, exceto as que contêm pilastras, que bloqueiam o seu movimento. O percurso do robô na arena durante o rali é determinado por uma sequência de instruções. Cada instrução é representada por um dos seguintes caracteres: 'D', 'E' e 'F', significando, respectivamente, "gire 90 graus para a direita", "gire 90 graus para a esquerda" e "ande uma célula para frente". O robô começa o rali em uma posição inicial na arena e segue fielmente a sequência de instruções dada (afinal, eles são robôs!). Sempre que o robô ocupa uma célula que contém uma figurinha da Copa ele a coleta. As figurinhas da Copa não são repostas, ou seja, cada figurinha pode ser coletada uma única vez. Quando um robô tenta andar para uma célula onde existe uma pilastra ele patina, permanecendo na célula onde estava com a mesma orientação. O mesmo também acontece quando um robô tenta sair da arena.

Dados o mapa da arena, descrevendo a posição de pilastras e figurinhas, e a sequência de instruções de um robô, você deve escrever um programa para determinar o número de figurinhas coletadas pelo robô.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém três números inteiros N, M e S ($1 \leq N, M \leq 100$, $1 \leq S \leq 5 \times 10^4$), separados por espaços em branco, indicando respectivamente o número de linhas e o número de colunas da arena e o número de instruções para o robô. Cada uma das N linhas seguintes da entrada descreve uma linha de células da arena e contém uma cadeia com M caracteres. A primeira linha que aparece na descrição da arena é a que está mais ao Norte; a primeira coluna que aparece na descrição de uma linha de células da arena é a que está mais a Oeste.

Cada célula da arena pode conter um dos seguintes caracteres:

- '.' — célula normal;
- '*' — célula que contém uma figurinha da Copa;
- '#' — célula que contém uma pilastra;
- 'N', 'S', 'L', 'O' — célula onde o robô inicia o percurso (única na arena). A letra representa a orientação inicial do robô (Norte, Sul, Leste e Oeste, respectivamente).

A última linha da entrada contém uma sequência de S caracteres dentre 'D', 'E' e 'F', representando as instruções do robô.

O último caso de teste é seguido por uma linha que contém apenas três números zero separados por um espaço em branco.

A entrada deve ser lida da entrada padrão.

Saída

Para cada rali descrito na entrada seu programa deve imprimir uma única linha contendo um único inteiro, indicando o número de figurinhas que o robô colecionou durante o rali.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 3 2 *** *N* *** DE 4 4 5 ...# *#O. *.*. *.#. FFEFF 10 10 20 ....*..... ....*.. .....*.... ..*.#..... ...#N.*..* ...*...... .... FDFFFFFFEEFFFFFFEFDF 0 0 0	0 1 3

Onde Estão as Bolhas? (2 – azul)

Arquivo fonte: *bolhas.c*, *bolhas.cpp* ou *bolhas.java*

Uma das operações mais frequentes em computação é ordenar uma sequência de objetos. Portanto, não é surpreendente que essa operação seja também uma das mais estudadas.

Um algoritmo bem simples para ordenação é chamado *Bubblesort*. Ele consiste de vários turnos. A cada turno o algoritmo simplesmente itera sobre a sequência trocando de posição dois elementos consecutivos se eles estiverem fora de ordem. O algoritmo termina quando nenhum elemento trocou de posição em um turno.

O nome *Bubblesort* (ordenação das bolhas) deriva do fato de que elementos menores ("mais leves") movem-se na direção de suas posições finais na sequência ordenada (movem-se na direção do início da sequência) durante os turnos, como bolhas na água. Abaixo é mostrada uma implementação do algoritmo em pseudocódigo:

```
Para i variando de 1 a N faça
    Para j variando de N − 1 a i faça
        Se seq[ j − 1] > seq[ j] então
            Troque os elementos seq[ j − 1] e seq[ j]
        Fim-Se
    Fim-Para
    Se nenhum elemento trocou de lugar então
        Final do algoritmo
    Fim-Se
Fim-Para
```

Por exemplo, ao ordenar a sequência [5, 4, 3, 2, 1] usando o algoritmo acima, quatro turnos são necessários. No primeiro turno ocorrem quatro intercâmbios: 1×2, 1×3, 1×4 e 1×5; no segundo turno ocorrem três intercâmbios: 2 × 3, 2 × 4 e 2 × 5; no terceiro turno ocorrem dois intercâmbios: 3 × 4 e 3 × 5; no quarto turno ocorre um intercâmbio: 4 × 5; no quinto turno nenhum intercâmbio ocorre e o algoritmo termina.

Embora simples de entender, provar correto e implementar, o algoritmo *bubblesort* é muito ineficiente: o número de comparações entre elementos durante sua execução é, em média, diretamente proporcional a N^2, sendo N o número de elementos na sequência.

Você foi requisitado para fazer uma "engenharia reversa" no *bubblesort*, ou seja, dados o comprimento da sequência, o número de turnos necessários para a ordenação e o número de intercâmbios ocorridos em cada turno, seu programa deve descobrir uma possível sequência que, quando ordenada, produza exatamente o mesmo número de intercâmbios nos turnos.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros N e M que indicam, respectivamente, o número de elementos ($1 \leq N \leq 100.000$) na sequência que está sendo ordenada, e o número de turnos ($0 \leq M \leq 100.000$) necessários para ordenar a sequência usando *bubblesort*. A segunda linha de um caso de teste contém M inteiros X_i, indicando o número de intercâmbios em cada turno i ($1 \leq X_i \leq N - 1$, para $1 \leq i \leq M$).

O final da entrada é indicado por $N = M = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma linha na saída, contendo uma permutação dos números $\{1, 2, \ldots, N\}$, que quando ordenada usando *bubblesort* produz o mesmo número de intercâmbios no mesmo número de turnos especificados na entrada. Ao imprimir a permutação, deixe um espaço em branco entre dois elementos consecutivos. Se mais de uma permutação existir, imprima a maior na ordem lexicográfica padrão para sequências de números (a ordem lexicográfica da permutação $a_1, a_2, \ldots a_N$ é maior do que a da permutação $b_1, b_2, \ldots b_N$ se para algum $1 \leq i \leq N$ temos $a_i > b_i$ e o prefixo $a_1, a_2, \ldots a_{i-1}$ é igual ao prefixo $b_1, b_2, \ldots b_{i-1}$).

Em outras palavras, caso exista mais de uma solução, imprima aquela em que o primeiro elemento da permutação é o maior possível. Caso exista mais de uma solução satisfazendo essa restrição, imprima, dentre estas, aquela em que o segundo elemento é o maior possível. Caso exista mais de uma solução satisfazendo as duas restrições anteriores, imprima, dentre estas, a solução em que o terceiro elemento é o maior possível e, assim, sucessivamente.

Para toda entrada haverá pelo menos uma permutação solução.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 1 1 5 4 4 3 2 1 6 5 2 2 2 2 1 0 0	2 1 3 5 4 3 2 1 6 5 1 2 3 4

Car Plates Competition (3 – verde)

Source file: *carplates.c*, *carplates.cpp* ou *carplates.java*

Martin and Isa are very competitive. The newest competition they have created is about looking at the plates of the cars. Each time one of them sees a car plate in the streets, he or she sends to the other an SMS message with the content of that plate; the one who has seen the newest plate is in the lead of the game. As the Automobile Car Management (ACM) office assigns the plates sequentially in increasing order, they can compare them and find out who is the winner.

Martin has a very smart eye and he has stayed on the lead for several weeks. Maybe he keeps looking at the streets instead of working, or maybe he stays all day in front of car selling companies waiting for new cars to go out with new plates. Isa, tired of being always behind, has written a program that generates a random plate, so the next time Martin sends a message to her, she will respond with this generated plate. In this way, she hopes to give Martin a hard time trying to beat her.

However, Martin has grown suspicious, and he wants to determine if Isa actually saw a car with the plate she sent or not. This way, he will know if Isa is in the lead of the game.

He knows some facts about the plates assigned by the ACM:

- Each plate is a combination of 7 characters, which may be uppercase letters (A-Z), or digits (0-9).
- There are two kinds of plate schemes: the old one, used for several years, and the new one which has been in use for some months, when the combinations of the old one were exhausted.
- In the old scheme, the first three characters were letters, and the last four were digits, so the plates run from AAA0000 to ZZZ9999.
- In the new scheme, the first five characters are letters, and the last two are digits. Unfortunately the chief of ACM messed up with the printing system while he was trying to create a poster for his next campaign for mayor, and the printer is not able to write the letters A, C, M, I, and P. Therefore, in this new scheme, the first plate is BBBBB00, instead of AAAAA00.
- The plates are assigned following a sequential order. As a particular case, the last plate from the old scheme is followed by the first plate from the new scheme.

As Isa is not aware of all of this, she has just made sure that her random generator creates a combination consisting of seven characters, where the first three characters are always uppercase letters, the last two characters are always digits, and each one of the fourth and fifth characters may be an uppercase letter or a digit (possibly generating an illegal combination, but she has not much time to worry about that).

Of course, Martin will not consider Isa the winner if he receives an illegal combination, or if he receives a legal plate, but equal to or older than his. But that's not all of it. Since Martin knows that new plates are not generated too fast, he will not believe that Isa saw a car with a plate newer than the one he sent, but sequentially too far. For example, if Martin sends DDDDD45, and receives ZZZZZ45, he will not believe that Isa saw a car

with that plate, because he knows that the ACM couldn't have printed enough plates to get to ZZZZZ45 in the time he received that answer.

So, Martin has decided to consider Isa the winner only if he receives a legal plate, newer than his, and older than or equal to the *C*-th consecutive plate after the one he sent. He calls *C* his confidence number. For example, if Martin sends ABC1234, and his *confidence number* is 6, he will think that Isa is the winner only if he receives any plate newer than ABC1234, but older than or equal to ABC1240.

Input

The input contains several test cases. Each test case is described in a single line that contains two strings S_M and S_I, and an integer *C*, separated by single spaces. S_M is the 7-character string sent by Martin, which is always a legal plate. S_I is the 7-character string answered by Isa, which was generated using her random generator. *C* is Martin's confidence number ($1 \le C \le 10^9$).

The end of input is indicated by $S_M = S_I =$ "*" and $C = 0$.

The input must be read from standard input.

Output

For each test case, output a single line with the uppercase character "Y" if, according to Martin, Isa is the winner, and with the uppercase character "N" otherwise.

The output must be written to standard output.

Sample Input	**Output for the sample Input**
ABC1234 ABC1240 6 ABC1234 ABC1234 6 ACM5932 ADM5933 260000 BBBBB23 BBBBC23 100 BBBBB23 BBBBD00 77 ZZZ9997 ZZZ9999 1 ZZZ9998 BBBBB01 3 ZZZZZ95 ZZZZZ99 10 BBBBB23 BBBBB22 5 * * 0	Y N N N Y N Y Y N

Jingle Composing (4 – rosa)

Source file: *jingle.c*, *jingle.cpp* ou *jingle.java*

A. C. Marcos is taking his first steps in the direction of jingle composition. He is having some troubles, but at least he is achieving pleasant melodies and attractive rhythms.

In music, a note has a pitch (its frequency, resulting in how high or low is the sound) and a duration (for how long the note should sound). In this problem we are interested only in the duration of the notes.

A jingle is divided into a sequence of measures, and a measure is formed by a series of notes.

The duration of a note is indicated by its shape. In this problem, we will use uppercase letters to indicate a note's duration. The following table lists all the available notes:

Notes	𝅝	𝅗𝅥	♩	♪	𝅘𝅥𝅯	𝅘𝅥𝅰	𝅘𝅥𝅱
Identifier	W	H	Q	E	S	T	X
Duration	1	1/2	1/4	1/8	1/16	1/32	1/64

The duration of a measure is the sum of the durations of its notes. In Marcos' jingles, each measure has the same duration. As Marcos is just a beginner, his famous teacher Johann Sebastian III taught him that the duration of a measure must always be 1.

For example, Marcos wrote a composition containing five measures, of which the first four have the correct duration and the last one is wrong. In the example below, each measure is surrounded with slashes and each note is represented as in the table above.

/HH/QQQQ/XXXTXTEQH/W/HW/

Marcos likes computers as much as music. He wants you to write a program that determines, for each one of his compositions, how many measures have the right duration.

Input

The input contains several test cases. Each test case is described in a single line containing a string whose length is between 3 and 200 characters, inclusive, representing a composition.

A composition begins and ends with a slash '/'. Measures in a composition are separated by a slash '/'. Each note in a measure is represented by the corresponding uppercase letter, as described above. You may assume that each composition contains at least one measure and that each measure contains at least one note. All characters in the input will be either slashes or one of the seven uppercase letters used to represent notes, as described above.

The last test case is followed by a line containing a single asterisk.

The input must be read from standard input.

Output

For each test case your program must output a single line, containing a single integer, the number of measures that have the right duration.

The output must be written to standard output.

Sample input	**Output for the sample input**
/HH/QQQQ/XXXTXTEQH/W/HW/ /W/W/SQHES/ /WE/TEX/THES/ *	4 3 0

Revisão de Contrato (5 – amarelo)

Arquivo fonte: *contrato.c*, *contrato.cpp* ou *contrato.java*

Durante anos, todos os contratos da Associação de Contratos da Modernolândia (ACM) foram datilografados em uma velha máquina de datilografia.

Recentemente Sr. Miranda, um dos contadores da ACM, percebeu que a máquina apresentava falha em um, e apenas um, dos dígitos numéricos. Mais especificamente, o dígito falho, quando datilografado, não é impresso na folha, como se a tecla correspondente não tivesse sido pressionada. Ele percebeu que isso poderia ter alterado os valores numéricos representados nos contratos e, preocupado com a contabilidade, quer saber, a partir dos valores originais negociados nos contratos, que ele mantinha em anotações manuscritas, quais os valores de fato representados nos contratos. Por exemplo, se a máquina apresenta falha no dígito 5, o valor 1500 seria datilografado no contrato como 100, pois o 5 não seria impresso. Note que o Sr. Miranda quer saber o valor numérico representado no contrato, ou seja, nessa mesma máquina, o número 5000 corresponde ao valor numérico 0, e não 000 (como ele de fato aparece impresso).

Entrada

A entrada consiste de diversos casos de teste, cada um em uma linha. Cada linha contém dois inteiros D e N ($1 \leq D \leq 9$, $1 \leq N < 10^{100}$), representando, respectivamente, o dígito que está apresentando problema na máquina e o número que foi negociado originalmente no contrato (que podem ser grande, pois Modernolândia tem sido acometida por hiperinflação nas últimas décadas). O último caso de teste é seguido por uma linha que contém apenas dois zeros separados por espaços em branco.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada o seu programa deve imprimir uma linha contendo um único inteiro V, o valor numérico representado de fato no contrato.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
5 5000000 3 123456 9 23454324543423 9 99999999991999999 7 777 0 0	0 12456 23454324543423 1 0

Mário (6 – branco)

Arquivo fonte: *mario.c*, *mario.cpp* ou *mario.java*

Mário é dono de uma empresa de guarda-volumes, a Armários a Custos Moderados (ACM). Mário conquistou sua clientela graças à rapidez no processo de armazenar os volumes. Para isso, ele tem duas técnicas:

- Todos os armários estão dispostos numa fila e são numerados com inteiros positivos a partir de 1. Isso permite a Mário economizar tempo na hora de procurar um armário;
- Todos os armários têm rodinhas, o que lhe dá grande flexibilidade na hora de rearranjar seus armários (naturalmente, quando Mário troca dois armários de posição, ele também troca suas numerações, para que eles continuem numerados sequencialmente a partir de 1).

Para alugar armários para um novo cliente, Mário gosta de utilizar armários contíguos, pois no início da locação um novo cliente em geral faz muitas requisições para acessar o conteúdo armazenado, e o fato de os armários estarem contíguos facilita o acesso para o cliente e para Mário.

Desde que Mário tenha armários livres em quantidade suficiente, ele sempre pode conseguir isso. Por exemplo, se a requisição de um novo cliente necessita de quatro armários, mas apenas os armários de número 1, 3, 5, 6, 8 estiverem disponíveis, Mário pode trocar os armários 5 e 2 e os armários 6 e 4 de posição: assim, ele pode alugar o intervalo de armários de 1 até 4.

No entanto, para minimizar o tempo de atendimento a um novo cliente, Mário quer fazer o menor número de trocas possível para armazenar cada volume. No exemplo acima, ele poderia simplesmente trocar os armários 1 e 4 de posição, e alugar o intervalo de 3 até 6.

Mário está muito ocupado com seus clientes e pediu que você fizesse um programa para determinar o número mínimo de trocas necessário para satisfazer o pedido de locação de um novo cliente.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois números inteiros N e L ($1 \leq N \leq L \leq 100.000$), indicando quantos armários são necessários para acomodar o pedido de locação do novo cliente e quantos armários estão disponíveis, respectivamente. A linha seguinte contém L números inteiros positivos separados por espaços em branco, nenhum deles maior do que 1.000.000.000, indicando as posições dos armários disponíveis. Os números dos armários livres são dados em ordem crescente.

O final da entrada é indicado por um caso em que $N = L = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, imprima uma linha contendo um único número inteiro, indicando o número mínimo de trocas que Mário precisa efetuar para satisfazer o pedido do novo cliente (ou seja, ter *N* armários consecutivos livres).

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
5 6 1 3 4 5 6 8 5 5 1 3 5 6 8 5 6 1 4 5 6 7 8 0 0	1 2 0