10^o Treino para Alunos da UFPR

27 de Julho de 2013

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

Organizadores:
Vinicius Kwiecien Ruoso e Ricardo Tavares de Oliveira

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: Purification

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

You are an adventurer currently journeying inside an evil temple. After defeating a couple of weak zombies, you arrived at a square room consisting of tiles forming an $n \times n$ grid. The rows are numbered 1 through n from top to bottom, and the columns are numbered 1 through n from left to right. At the far side of the room lies a door locked with evil magical forces. The following inscriptions are written on the door:

The cleaning of all evil will awaken the door!

Being a very senior adventurer, you immediately realize what this means. You notice that every single cell in the grid are initially evil. You should purify all of these cells.

The only method of tile purification known to you is by casting the "Purification" spell. You cast this spell on a single tile – then, all cells that are located in the same row and all cells that are located in the same column as the selected tile become purified (including the selected tile)! It is allowed to purify a cell more than once.

You would like to purify all $n \times n$ cells while minimizing the number of times you cast the "Purification" spell. This sounds very easy, but you just noticed that some tiles are particularly more evil than the other tiles. You cannot cast the "Purification" spell on those particularly more evil tiles, not even after they have been purified. They can still be purified if a cell sharing the same row or the same column gets selected by the "Purification" spell.

Print the minimum number of spells cast needed to purify all the cells, or print -1 if it is impossible.

Input

The input contains one or more test cases. The first line of each test case will contain a single integer $n(1 \leq n \leq 1000)$. Then, n lines follows, each contains n characters. The $j-$th character in the $i-$th row represents the cell located at row i and column j. It will be the character E if it is a particularly more evil cell, and . otherwise. The last test case is followed by a line containing the number 0 (zero).

Output

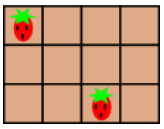
For each test case, print the minimum number of spells cast needed to purify all the cells, or print -1 if it is impossible.

Sample Input	**Sample Output**
3 .E. E.E .E. 3 EEE E.. E.E 5 EE.EE E.EE. E...E .EE.E EE.EE 0	3 -1 5

B: Cakeminator

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

You are given a rectangular cake, represented as an $r \times c$ grid. Each cell either has an evil strawberry, or is empty. For example, a 3×4 cake may look as follows:



The cakeminator is going to eat the cake! Each time he eats, he chooses a row or a column that does not contain any evil strawberries and contains at least one cake cell that has not been eaten before, and eats all the cake cells there. He may decide to eat any number of times.

Please output the maximum number of cake cells that the cakeminator can eat.

Input

The first line of each test case contains two integers r and c ($2 \leq r, c \leq 10$), denoting the number of rows and the number of columns of the cake. The next r lines each contains c characters: the $j-$th character of the $i-$th line denotes the content of the cell at row i and column j, and is either one of these:

- $'.'$ character denotes a cake cell with no evil strawberry;
- $'S'$ character denotes a cake cell with an evil strawberry.

The last test case is followed by a line containing two zeros.

Output

For each test case, output the maximum number of cake cells that the cakeminator can eat.

Sample Input	**Sample Output**
3 4 S... .... ..S. 2 2 .. .. 0 0	8 4

C: Aceitação Máxima

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

Está ocorrendo uma reforma política em Berland. O parlamento Berlandês aprovou n novas leis (identificadas por inteiros de 1 a n) que devem agora ser sancionadas pelo presidente, G.W. Boosch.

O presidente, entretanto, não irá sancionar todas as n leis, mas sim exatamente $2k$ delas ($2k \leq n$). Ele decidiu que irá escolher **exatamente dois** intervalos *disjuntos* de $[1...n]$, de tamanho k cada um, e sancionar as leis identificadas pelos números que estão nesses intervalos. Formalmente, sr. Boosch irá escolher dois inteiros $a, b (1 \leq a \leq b \leq n - k + 1, b - a \geq k)$ e sancionar as leis cujos identificadores estão nos intervalos $[a; a+k-1]$ e $[b; b+k-1]$ (incluindo os números nas extremidades dos intervalos).

O presidente irá considerar a opinião pública para escolher quais leis sancionar. Através da análise do resultado de uma pesquisa de opinião, cada lei i foi associada com um número inteiro x_i, o *valor de aceitação* da lei (quanto maior x_i, mais aceita pelo público a lei é).

Ajude o presidente a escolher quais intervalos de leis sancionar, de tal forma que a soma dos valores de aceitação das leis sancionadas seja máxima.

Entrada

A entrada consiste de vários casos de teste.

A primeira linha de cada caso de teste contém dois inteiros n e k ($2 \leq n \leq 2 \times 10^5, 0 < 2k \leq n$), o número de leis aprovadas pelo parlamento e o tamanho dos intervalos a serem usados pelo presidente, respectivamente. A próxima linha contém n inteiros $x_1, x_2, ..., x_n$: os valores de aceitação de cada lei ($1 \leq x_i \leq 10^9$).

O último caso de teste é seguido por uma linha contendo dois zeros.

Saída

Para cada caso de teste, imprima os inteiros a e b: os índices do início dos intervalos escolhidos (isto é, inteiros tais que o sr. Boosch irá sancionar as leis em $[a; a + k - 1]$ e $[b; b + k - 1]$).

Se há várias soluções, imprima aquela que contém o menor valor possível de a. Se ainda há várias soluções, imprima aquela que contém o menor valor possível de b.

Exemplo de entrada	**Exemplo de saída**
5 2 3 6 1 1 6 6 2 1 1 1 1 1 1 0 0	1 4 1 3

D: Competição de placas de carros

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

Martin e Isa são bem competitivos. A mais nova competição que eles criaram é sobre olhar as placas dos carros. Cada vez que um deles vê uma placa de carro na rua, ele ou ela manda para o outro uma mensagem SMS com o conteúdo da placa; aquele que tiver visto a placa mais recente fica na liderança. Como um escritório da Gerência de Veículos Automotores (ACM) coloca as placas seqüencialmente em ordem crescente, eles podem comparar elas e descobrir que é o vencedor.

Martin tem um olhar atento e ficou várias semanas na liderança. Talvez ele fique olhando para a rua ao invés de trabalhar, ou talvez ele fique o dia inteiro em frente a lojas de carros esperando novos carros saírem com novas placas. Isa, cansada de ficar para trás, escreveu um programa que gera uma placa aleatória, para que a próxima vez que Martin lhe mande uma mensagem, ela responda com a placa gerada. Desse jeito, ela espera dificultar as coisas para Martin.

Entretanto, Martin começou a desconfiar, e quer determinar se Isa realmente viu um carro com a placa que ela mandou ou não. Desse jeito, ele saberá se a Isa está na liderança do jogo.

Ele sabe alguns fatos sobre as placas feitas pela ACM:

- Cada placa é uma combinação de 7 caracteres, que podem ser letras maiúsculas (A-Z) ou dígitos (0-9).
- Existem dois tipos de esquema de placas: o velho, usado por vários ano, e o novo, que está em uso há alguns meses, quando as combinações do velho foram extigüidas.
- No esquema velho, os três primeiros caracteres eram letras, e os últimos quatro eram dígitos, portanto as placas vão de AAA0000 a ZZZ9999.
- No esquema novo, os primeiros cinco caracteres são letras, e os ultimos dois são dígitos. Infelizmente o chefe da ACM bagunçou o sistema de impressão enquanto ele estava tentando criar um poster para sua próxima campanha para prefeito, e a impressora não consegue imprimir as letras A, C, M, I e P. Portanto, no novo esquema, a primeira placa é BBBBB00, ao invés de AAAAA00.
- As placas são criadas em seqüência. Como caso particular, a última placa do esquema velho é seguida pela primeira placa do novo esquema. Como Isa não sabe de tudo isso, ela só garantiu que o gerador aleatório dela criasse uma combinação consistindo de sete caracteres, onde os três primeiros sempre são letras maiúsculas, os dois últimos são sempre dígitos, e o quarto e o quinto podem ser tanto uma letra maiúscula quanto um dígito (possivelmente gerando uma combinação ilegal, mas ela não tem muito tempo para se preocupar com isso).

Obviamente, Martin não considerará Isa a vencedora se ele receber uma combinação ilegal, ou se ele receber uma placa legal, mas igual ou mais velha do que a dele. Mas isso não é tudo. Como ele sabe que as placa novas não são geradas muito rápido, ele não acreditará que a Isa viu um carro com uma placa mais nova que a dele, mas sequencialmente muito distante.

Por exemplo, se Martin mandar DDDDD45, e receber ZZZZZ45, ele não acreditará que Isa viu um carro com aquela placa, porque ele sabe que a ACM não conseguiria imprimir placas suficientes para chegar a ZZZZZ45 no momento que ele recebeu a resposta.

Então, Martin decidiu considerar Isa a vencedora somente se ele receber uma placa legal, mais nova que a dele, e mais velha ou igual que a C-ésima placa consecutiva depois da que ele mandou. Ele chama de C seu número de confidência. Por exemplo, se Martin mandar ABC1234, e seu número de confidência é 6, ele considerará que Isa é a vencedora somente se ele receber uma placa mais nova que ABC1234, mas mais velha ou igual a ABC1240.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é descrito em uma única linha que contém duas strings SM e SI, e um inteiro C, separados por um único espaço cada. SM é a string de 7 caracteres enviada por Martin, que é sempre válida. SI é a string de 7 caracteres respondida por Isa, que foi gerada usando seu gerador aleatório. C é o número de confidência de Martin ($1 \leq C \leq 10^9$).

O final da entrada é dado por $SM = SI =' *'$ e $C = 0$.

Saída

Para cada caso de teste, imprima uma única linha com o caractere maiúsculo Y se, de acordo com Martin, Isa é a vencedora, e com a letra maiúscula N caso contrário.

Exemplo de entrada	**Exemplo de saída**
ABC1234 ABC1240 6	Y
ABC1234 ABC1234 6	N
ACM5932 ADM5933 260000	N
BBBBB23 BBBBC23 100	N
BBBBB23 BBBBD00 77	Y
ZZZ9997 ZZZ9999 1	N
ZZZ9998 BBBBB01 3	Y
ZZZZZ95 ZZZZZ99 10	Y
BBBBB23 BBBBB22 5	N
* * 0	

E: Loop Musical

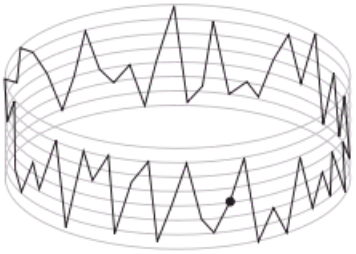
`File: loop.[c|cpp|java|pas]`

Um loop musical é um trecho de música que foi composto para repetir continuamente (ou seja, o trecho inicia novamente toda vez que chega ao final), sem que se note descontinuidade. Loops são muito usados na sonorização de jogos, especialmente jogos casuais pela internet.

Loops podem ser digitalizados por exemplo utilizando PCM. PCM, do inglês Pulse Code Modulation, é uma técnica para representação de sinais analógicos, muito utilizada em estúdio digital. Nessa técnica, a magnitude do sinal é amostrada a intervalos regulares de tempo, e os valores amostrados são armazenados em seqüência. Para reproduzir a forma de onda amostrada, o processo é invertido (demodulação).

Fernandinha trabalha para uma empresa que desenvolve jogos e compôs um bonito loop musical, codificando-o em PCM. Analisando a forma de onda do seu loop em um software de edição de estúdio, Fernandinha ficou curiosa ao notar a quantidade de "picos" existentes. Um pico em uma forma de onda é um valor de uma amostra que representa um máximo ou mínimo local, ou seja, um ponto de inflexão da forma de onda. A figura abaixo ilustra (a) um exemplo de forma de onda e (b) o loop formado com essa forma de onda, contendo 48 picos.

(a) Uma forma de onda



(b) Mesma forma de onda em loop

Fernandinha é uma amiga muito querida e pediu sua ajuda para determinar quantos picos existem no seu loop musical.

Entrada

A entrada contém varios casos de teste. A primeira linha de um caso de teste contém um inteiro N, representando o número de amostras no loop musical de Fernandinha ($2 \leq N \leq 10^4$). A segunda linha contém N inteiros H_i, separados por espaços, representando a seqüência de magnitudes das amostras ($-10^4 \leq H_i \leq 10^4$ para $1 \leq i \leq N$, $H_1 \neq H_N$ e $H_i \neq H_i + 1$ para $1 \leq i < N$). Note que H_1 segue H_N quando o loop é reproduzido.

O final da entrada é indicado por uma linha que contém apenas o número zero.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha, contendo apenas um inteiro, o número de picos existentes no loop musical de Fernandinha.

Exemplo de entrada	**Exemplo de saída**
2 1 -3 6 40 0 -41 0 41 42 4 300 450 449 450 0	2 2 4