4^a Seletiva para Maratona de Programação UFPR

9 de Agosto de 2013

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

Organizadores:
André Guedes, Vinicius Ruoso e Ricardo 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: Sub-prime

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

A mais recente crise econômica foi em parte causada pela forma como os bancos faziam empréstimos para pessoas que não tinham capacidade de honrá-los e revendiam tais empréstimos para outros bancos (debêntures). Obviamente, quando as pessoas pararam de pagar os empréstimos, o sistema inteiro entrou em colapso.

A crise foi tão profunda que acabou atingindo países do mundo inteiro, inclusive a Nlogônia, onde o honrado primeiro ministro Man Dashuva ordenou que o presidente do Banco Central procurasse uma solução para o problema. Esse, por sua vez, teve uma idéia brilhante: se cada banco fosse capaz de liquidar seus empréstimos somente com suas reservas monetárias, todos os bancos sobreviveriam e a crise seria evitada. Entretanto, com o elevado número de debêntures e bancos envolvidos, essa tarefa é extremamente complicada, e portanto ele pediu a sua ajuda para escrever um programa que, dados os bancos e as debêntures emitidas, determine se é possível que todos os bancos paguem suas dívidas, utilizando suas reservas monetárias e seus créditos.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros B e N, indicando respectivamente o número de bancos ($1 \leq B \leq 20$) e o número de debêntures emitidas pelos bancos ($1 \leq N \leq 20$). Os bancos são identificados por inteiros entre 1 e B. A segunda linha contém B inteiros R_i separados por espaços, indicando as reservas monetárias de cada um dos B bancos ($0 \leq R_i \leq 10^4$, para $1 \leq i \leq B$). As N linhas seguintes contêm cada uma três inteiros separados por espaços: um inteiro D, indicando o banco devedor ($1 \leq D \leq B$), um inteiro C, indicando o banco credor ($1 \leq C \leq B$ e $D \neq C$), e um inteiro V, indicando o valor da debênture ($1 \leq V \leq 10^4$).

O final da entrada é indicado por uma linha que contém apenas dois zeros, separados por um espaço em branco.

Os dados devem ser lidos da entrada padrão.

Saída

Para caso de teste, seu programa deve imprimir uma única linha, contendo um único caractere: "S", se for possível liquidar todos as debêntures sem intervenção do Banco Central da Nlogônia, e "N", se algum banco precisar de empréstimos do governo para liquidar suas debêntures.

O resultado de seu programa deve ser escrito na saída padrão.

Exemplo de entrada	**Exemplo de saída**
3 3 1 1 1 1 2 1 2 3 2 3 1 3 3 3 1 1 1 1 2 1 2 3 2 3 1 4 3 3 1 1 1 1 2 2 2 3 2 3 1 2 0 0	S N S

B: Nested Shrubbery Boxes

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

It is a common prank to give someone a gift in a large box, in which is nested a smaller box, with another smaller box inside that one, and so forth, until the smallest box – nested within all those other boxes – contains the gift. Given a set of boxes of various sizes, your problem is to find the size (cardinality) of the largest subset of boxes that can be used to create such a nested arrangement. If no boxes can be nested, then the size of the subset is just 1.

Naturally, each box in the set from which you can choose has three dimensions. Any box can be rotated, if desired, if that would enable it to fit inside another box. For our purposes, a box A can fit inside a box B if each dimension of box A is strictly less than the corresponding dimension of box B.

For example, suppose box A has dimensions $12 \times 20 \times 60$ and box B has dimensions $42 \times 18 \times 10$. If we rotate box B appropriately – so the dimensions are $10 \times 18 \times 42$, then we will be able to nest it inside box A. However, if box B had dimensions $13 \times 12 \times 58$, then no rotations would allow it to fit inside box A.

Input

The input will consist of an unspecified number of box sets. Each set will begin with a line containing n, $0 \leq n \leq 500$, the number of boxes in the set. Each box will be described on its own line by three positive integers representing length, width and height (Each value will not exceed 1000). The end of input occurs when $n = -1$.

Output

For each set of boxes, print a line containing the largest number of boxes that can be selected from the original set to form a fully nesting subset of boxes.

Sample Input	**Sample Output**
5 145 472 812 827 133 549 381 371 900 271 389 128 718 217 491 4 432 123 139 942 844 783 481 487 577 677 581 701 -1	2 4

C: Dinner Hall

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

The University administration plans to build a new dinner hall, to replace the several small (and rather inadequate) dinner halls spread over the campus. To estimate the number of places needed in the new dinner hall, they performed an experiment to measure the maximum total number of clients inside the existing dinner halls at any time. They hired several students as pollers, and positioned one poller at each entrance and each exit of the existing dinner halls. The pollers' task was to note in small cards the time each client entered or exited the hall (one card for each event). In each card they wrote the time, in the format $HH : MM : SS$, and the associated event (letter E for an entry, letter X for and exit).

The experiment started in the morning, before breakfast, and ended in the evening, after dinner. The pollers had their watches synchronized, and the halls were empty both before and after the experiment (that is, no client was inside a hall before the experiment began, and no client remained in a hall after the experiment ended). The pollers wrote exactly one card for every client who entered a hall and for every client who exited a hall.

After the experiment, the cards were collected and sent to the administration for processing. The task, however, was not as easy as planned, because two problems were detected. Firstly, the cards were bunched together in no particular order and therefore needed sorting; that is fairly easy, but time-consuming to do by hand. But what is worse is that, although all cards had a valid time, some pollers forgot to write the letter specifying the event. The University administration decided they needed help from an expert!

Given a set of cards with times and the indication of the event (the indication of the event may be missing), write a program to determine the maximum number of clients that could possibly had been inside the dinner halls in a given instant of time.

Input

The input contains several test cases. The first line of a test case contains one integer N indicating the number of cards collected in the experiment ($2 \leq N \leq 64800$). Each of the next N lines contains the information written in a card, consisting of a time specification, followed by a single space, followed by an event specification. A time specification has the format $HH : MM : SS$, where HH represents hours ($06 \leq HH \leq 23$), MM represents minutes ($00 \leq MM \leq 59$) and SS represents seconds ($00 \leq SS \leq 59$). Within a test case, no two cards have the same time. An event specification is a single character: uppercase E for entry, uppercase X for exit and ? for unknown. Information may be missing, but the information given is always correct. That is, the times noted in all cards are valid. Also, if a card describes an entry, then a client did enter a hall at the informed time; if a card describes an exit, then a client did leave a hall at the informed time; and if a card describes an unknown event, then a client did enter or leave a hall at the informed time.

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

Output

For each test case in the input, your program must print a single line, containing one single integer, the maximum total number of clients that could have been inside the dinner halls in a given instant of time.

Sample Input	**Sample Output**
4 07:22:03 X 07:13:22 E 08:30:51 E 21:59:02 X 4 09:00:00 E 20:00:01 X 09:05:00 ? 20:00:00 ? 8 10:21:00 E 10:25:00 X 10:23:00 E 10:24:00 X 10:26:00 X 10:27:00 ? 10:22:00 ? 10:20:00 ? 0	1 2 4

D: It's Super Effective!

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

As you may know, this world is inhabited by creatures called *Dokémon*! For some people, *Dokémon* are pets. Other use them for fights. During a *Dokémon* battle, two *Dokémon* attack each other, alternately, until one of them faints. The survivor is declared the winner.

In this problem, every *Dokémon* has one (and only one) type, which indicates the creature's strength. Also, all attacks made by a *Dokémon* are physical and cause damage to its opponent.

An attack can be *regular*, *super effective* (it causes twice as much damage of a regular attack) or *not very effective* (it causes half the damage of a regular attack). This classification depends on the types of the Dokémon that are fighting. The table below indicates how the attacks are classified. The character + in the i-th row and j-th column indicates that an attack made by an i-typed *Dokémon* against a j-typed *Dokémon* is super effective. The characters − and . indicate not very effective and regular attacks, respectively. The names above the columns are abbreviated, but represent the same names in the rows, in the same order.

	Nl	Fr	Wt	El	Gs	Ic	Fg	Ps	Gr	Fl	Py	Bg	Rk	Gh	Dr
Normal	.	.	.	.	.	.	.	.	.	.	.	.	−	.	.
Fire	.	−	−	.	+	+	.	.	.	.	.	+	−	.	−
Water	.	+	−	.	−	.	.	.	+	.	.	.	+	.	−
Electric	.	.	+	−	−	.	.	.	.	+	.	.	.	.	−
Grass	.	−	+	.	−	.	.	−	+	−	.	−	+	.	−
Ice	.	.	−	.	+	−	.	.	.	+	.	.	.	.	+
Fighting	+	.	.	.	.	+	.	−	.	−	−	−	+	.	.
Poison	.	.	.	.	+	.	.	−	−	.	.	+	−	−	.
Ground	.	+	.	+	−	.	.	+	.	.	.	−	+	.	.
Flying	.	.	.	−	+	.	+	.	.	.	.	+	−	.	.
Psychic	.	.	.	.	.	.	+	+	.	.	−	.	.	.	.
Bug	.	−	.	.	+	.	−	+	.	−	+	.	.	−	.
Rock	.	+	.	.	.	+	−	.	−	+	.	+	.	.	.
Ghost	.	.	.	.	.	.	.	.	.	.	.	.	.	+	.
Dragon	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+

Bash Catchem is a young boy from Lappet Town who wishes to be a *Dokémon Master* one day. However, he is still studying tactics for his *Dokémon* battles. Your task is to help Bash to determine, given descriptions of *Dokémon* Battles, the effectiveness of the attacks during the battles.

Input

Each test case starts with a line containing an integer P, $1 \leq P \leq 151$, the number of different *Dokémon* that exist in our world. Each of the next P lines contains the description of a *Dokémon* given as two strings N T, where N is the name of the creature and T is its type. Its name contains up to 15 lowercase or uppercase letters, and its type is one of the types given in the table above, not abbreviated.

The next line contain an integer Q, $1 \leq Q \leq P^2$, the number of queries. Each of the next Q lines contains names of two *Dokémon*, $N1$ $N2$, describing a battle. It's guaranteed that both names were described in the same test case.

The last test case is followed by a line containing the number 0.

Output

For each query, print *It's super effective!* if an attack of $N1$ is super effective against $N2$. Print *It's not very effective...* if the attacks are not very effective, or *Regular Attack.* otherwise. The character ' has the decimal code 39 in the ASCII table.

Print a blank line after each test case.

Sample input	Sample output
4 Pikaxu Electric Squartle Water Dullbasaur Grass Charminder Fire 2 Pikaxu Squartle Dullbasaur Charminder 3 Miauth Normal Wizang Poison Kabrada Psychic 2 Kabrada Wizang Wizang Miauth 0	It's super effective! It's not very effective... It's super effective! Regular Attack.

E: K-ésimo pai

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

Uma árvore de P nodos é um grafo conexo e não direcionado com $P-1$ arestas. Seja R um nodo em particular de uma árvore de P nodos, dito *raiz*. Se a distância do nodo R a um nodo A é L, se a distância do nodo R a um nodo B é $L+1$, e se há uma aresta entre A e B, então o nodo A é dito o *(primeiro) pai* do nodo B.

De forma análoga, se a distância do nodo R a um nodo A é L, se a distância do nodo R a um nodo B é $L+K$, e se a distância de A a B é igual a K, então A é dito o *K-ésimo pai* de B.

Por fim, uma *folha* de uma árvore é um nodo sem filhos, isto é, um nodo que não é (primeiro) pai de nenhum outro.

Susan gosta de brincar com grafos e árvores. Ela formulou um problema e quer saber se alguém é capaz de resolvê-lo. Susan tem uma árvore inicial de P nodos, e poderá adicionar folhas ou remover folhas ao longo da brincadeira. Sua tarefa é determinar, a qualquer instante, qual é o K-ésimo pai de um dado nodo.

Entrada

A primeira linha da entrada contém um inteiro T, indicando o número de casos de testes.

A primeira linha de cada caso de teste contém um inteiro $P(1 \leq P \leq 10^5)$, o número de nodos na árvore inicial. As próximas P linhas contém dois inteiros $X\ Y$ $(1 \leq X \leq 10^5, 0 \leq Y \leq 10^5)$ cada, indicando que Y é o (primeiro) pai de X. Se $Y = 0$, então X é a raiz da árvore. A próxima linha contém um inteiro $Q(1 \leq Q \leq 10^5)$, o número de operações e consultas. As próximas Q linhas contém cada uma uma operação ou consulta, na seguinte forma:

- 0 X Y: O nodo Y é adicionado à árvore como uma folha, cujo (primeiro) pai é X; É garantido que Y não está na árvore, e X está.
- 1 X: O nodo X é removido da árvore. É garantido que X é uma folha.
- 2 X K: Deve-se determinar qual é o K-ésimo pai do nodo X $(1 \leq K \leq 10^5)$.

Os nodos são sempre numerados de 1 a 10^5. Assim, pode haver, por exemplo, uma árvore com um único nodo, de número 10^5.

Saída

Para cada consulta, imprima o K-ésimo pai de X. Se o K-ésimo pai de X não existe, ou se o nodo X não está na árvore, imprima 0.

Exemplo de entrada	**Exemplo de saída**
2 7 2 0 5 2 3 5 7 5 9 8 8 2 6 8 10 0 5 15 2 15 2 1 3 0 15 20 0 20 13 2 13 4 2 13 3 2 6 10 2 11 1 2 9 1 1 10000 0 3 0 10000 4 1 4 2 4 1	2 2 5 0 0 8 0

F: Matrizes

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

O conglomerado indiano Tutu é um conjunto de empresas que atua nos mais diversos ramos da indústria, produzindo desde sapatos até aviões e foguetes. Por ser tão diversificada, precisa de grandes e rápidos sistemas para cálculos de contabilidade.

Um dos módulos mais importantes desse sistema é o de fornecimento de produtos, onde fica a base de dados de produtos e fornecedores. Um mesmo produto pode ser fornecido por vários fornecedores diferentes.

O sistema possui duas grandes matrizes: a matriz **A**, onde cada linha representa um produto e cada coluna representa um fornecedor. O valor da matriz na linha m e coluna n representa o preço do produto m se for comprado do fornecedor n.

A outra grande matriz é a **B**, onde cada linha representa um dia do mês e cada coluna é um produto. O valor da matriz na linha m e coluna n representa a quantidade do produto n a ser adquirido no dia m.

Tal empresa tem uma política de fidelidade com seus fornecedores, e uma das práticas efetuadas pela empresa é, em um determinado dia, comprar todos os produtos necessários de um único fornecedor. Isto é, em um dia todos os produtos adquiridos serão comprados do fornecedor x, no outro dia do fornecedor y, e assim por diante.

Para auxiliar a escolha de qual fornecedor será o escolhido no dia, foi gerada outra matriz **C**, que é o resultado da multiplicação das matrizes **A** $\times$ **B**. Essa matriz diz o quanto será gasto pela empresa se adquirir todos os produtos de um determinado fornecedor em um determinado dia.

As matrizes **A** e **B** são quadradas (o número de linhas é igual ao número de colunas) e têm valores definidos pelas fórmulas: $A_{ij} = (P \times i + Q \times j)(mod\ X)$ e $B_{ij} = (R \times i + S \times j)(mod\ Y)$, onde i é o índice da linha da matriz e j é o índice da coluna da matriz (todos os índices vão de 1 até N). Os inteiros P, Q, R, S, X e Y são parâmetros constantes, que definem as duas matrizes **A** e **B**.

Escreva um programa que, dados os parâmetros das matrizes **A** e **B**, e a posição de uma das entradas as matriz C, calcula o valor daquela entrada.

Saída

A entrada consiste em vários casos de testes. A primeira linha de um caso de teste contém um inteiro N, indicando as dimensões das matrizes **A**, **B** e **C** ($2 \leq N \leq 10^5$). A linha seguinte contém seis inteiros P, Q, R, S, X e Y, indicando os parâmetros das matrizes **A** e **B** ($2 \leq X$, $Y \leq 10^4$; $0 \leq P$, $Q < X$; $0 \leq R$, $S < Y$). Finalmente, a última linha do caso de teste contém dois inteiros I e J, indicando a linha e a coluna da matriz C a serem consultados ($1 \leq I, J \leq N$).

O final da entrada é dado por $N = 0$.

Saída

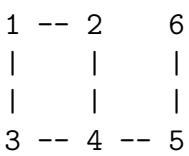
Seu programa deve imprimir uma única linha contendo o valor da matriz **C** na linha e coluna especificadas.

Exemplo de entrada	**Exemplo de saída**
3 4 3 2 3 5 6 2 2 4 3 5 1 0 6 7 4 3 2 2 2 0 1 3 2 2 1 0	18 30 2

G: Manutenção

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

Uma empresa possui vários computadores conectados em rede. Isto possibilita que os funcionários compartilhem recursos e consigam colaborar melhor para o desempenho de suas tarefas dentro da empresa. No entanto, as máquinas não estão diretamente conectadas todas entre si. Por economia de recursos, a topologia de rede adotada apresenta apenas um subconjunto das conexões possíveis, conforme o exemplo apresentado na figura abaixo. Note que as conexões são sempre bidirecionais.



Apesar de alguns computadores não estarem diretamente conectados entre si, eles ainda conseguem se comunicar porque existe um algoritmo de roteamento capaz de conectar dois computadores através de várias conexões diretas. No exemplo da figura, o computador 1 conseguiria se comunicar com o computador 5 através dos computadores 2 e 4 ou através dos computadores 3 e 4.

No entanto, freqüentemente as máquinas precisam passar por uma revisão de rotina. Quando uma máquina está em manutenção, ela precisa ser temporariamente desconectada da rede e levada para a oficina. Assim, o algoritmo de roteamento não tem mais como estabelecer conexões utilizando este computador, o que pode acabar desconectando duas ou mais partes da rede e prejudicando o trabalho na empresa. No exemplo dado, se o computador 2 fosse para a revisão não teríamos problema pois todas as outras máquinas ainda conseguiriam se comunicar entre si. No entanto, se o computador 4 fosse para a manutenção, as máquinas 1, 2 e 3 não conseguiriam se comunicar com as máquinas 5 e 6.

Se a remoção de uma máquina desconectar o restante da rede, impedindo que outros computadores se comuniquem, é necessário deixar uma máquina substituta em seu lugar durante o período de manutenção, o que representa um custo extra no orçamento da empresa. Sua tarefa é escrever um programa que identifique quais são os computadores que precisam ser substituídos durante a sua manutenção, para que os demais continuem se comunicando através da rede.

Entrada

A entrada é constituída de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros N e M, que indicam respectivamente o número de computadores na rede e o número de conexões diretas entre eles ($1 \leq N \leq 400$, e $N-1 \leq M \leq \frac{N(N-1)}{2}$). Os computadores são identificados por números de 1 a N. As M linhas seguintes contêm, cada uma, um par de números inteiros X e Y. Cada linha representa uma conexão existente na rede, indicando que os computadores X e Y possuem uma conexão direta entre si. O final da entrada é indicado por um conjunto de teste com N = M = 0. Você pode assumir que todos os conjuntos de teste representam redes conexas, onde todos os computadores conseguem se comunicar entre si através do algoritmo de roteamento.

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter a lista com os computadores que precisam ser substituídos durante sua manutenção. Esta lista deve estar ordenada de forma crescente, e cada valor deve ser seguido de um espaço em branco. Caso nenhum dos computadores da rede precise ser substituído, escreva "nenhum" na saída. A terceira linha deve ser deixada em branco. O formato mostrado no exemplo de saída abaixo deve ser seguido rigorosamente.

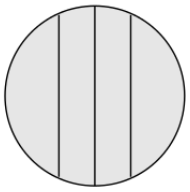
Exemplo de entrada	**Exemplo de saída**
6 6 1 2 3 1 2 4 3 4 4 5 5 6 4 6 1 2 1 3 1 4 2 3 2 4 3 4 4 3 1 2 1 3 1 4 0 0	Teste 1 4 5 Teste 2 nenhum Teste 3 1

H: Pizza Integral

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

Aberta apenas há 3 anos, a Pizzaria Integral tem mudado a história de Chapecó, atraindo olhares de renomados gastrônomos do mundo. Quando abriu, a proposta do estabelecimento era servir pizzas inovadoras, feitas com farinha de trigo integral, considerada mais saborosa e mais saudável. Passados alguns meses, o pizzaiolo começou a inovar também no menu, desenvolvendo pizzas cada vez mais exóticas, como *Lembranças duma Infância Mineira* (leva doce-de-leite, queijo minas e amoras flambadas em cachaça), *Romeu, Julieta e Um Terceiro* (leva goiabada, queijo mascarpone e camarões grandes) e *Carne-de-Onça* (leva carne moída de 1ª crua, chalota, alho chips, farofa de bala de hortelã, molho de limão e azeite de oliva).

Mas não é só nas receitas que a desbravadora pizzaria está inovando. Desde o ano passado, foi implantada também uma revolução no modo de vender as pizzas. Em pizzarias comuns, geralmente o freguês se limita a escolher o tamanho de sua pizza duma lista de tamanhos pré-definidos, como *pequeno*, *médio*, *grande* e *gigante*. Na Pizzaria Integral, contudo, o cliente pode escolher o diâmetro exato de sua pizza, e ainda pode escolher em quantos pedaços quer que ela venha cortada. O pizzaiolo garante não apenas a forma circular perfeita da pizza, atendendo o diâmetro especificado, mas também assegura que todos os pedaços terão exatamente a mesma área. Inovando ainda mais, os cortes são feitos todos verticais e paralelos. A Figura ilustra uma pizza cortada em 4 pedaços. Dado o diâmetro da pizza e a quantidade de pedaços na qual ela deve ser cortada, determine a posição dos cortes na pizza.



Entrada

A entrada é composta por vários casos de teste, cada qual numa linha. Cada caso de teste consiste de dois inteiros d ($20 \leq d \leq 100$) e k ($1 \leq k \leq 10$), separados por um espaço, representando respectivamente o diâmetro, em centímetros, solicitado para a pizza e o número de pedaços em que se deve cortá-la. A entrada é finalizada por $d = k = 0$.

Saída

Para cada caso de teste, seu programa deverá imprimir em sequência $k-1$ valores $v_1, v_2, ..., v_{k-1}$ ($0.00 < v_1 < v_2 < ... < v_{k-1} < d$), indicando onde devem ser feitos os cortes (isto é, a distância entre o corte e a extremidade esquerda da pizza), e uma quebra-de-linha. Os valores devem ser separados por um espaço e fornecidos sob precisão de duas casas decimais. Note que, quando $k = 1$, apenas a quebra-de-linha deve ser impressa.

Exemplo de entrada	**Exemplo de saída**
20 2 50 1 100 5 80 4 0 0	10.00 25.41 42.11 57.89 74.59 23.84 40.00 56.16

I: Conte os Fatores

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

Escreva um programa que computa o número de diferentes fatores primos de um inteiro positivo.

Entrada

A entrada consistirá de uma série de inteiros positivos. Cada linha possui somente um número. O valor máximo de um número é 1000000. O fim da entrada é indicado por um número igual a 0. Esse número não deve ser considerado como parte do conjunto de teste.

Saída

O programa deve imprimir cada resultado em uma linha diferente, seguindo o formado dado no exemplo de saída.

Exemplo de entrada	**Exemplo de saída**
289384 930887 692778 636916 747794 238336 885387 760493 516650 641422 0	289384 : 3 930887 : 2 692778 : 5 636916 : 4 747794 : 3 238336 : 3 885387 : 2 760493 : 2 516650 : 3 641422 : 3

J: Hangul Simplificado

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

Bruno é um garoto americano que adora o K-pop (*"Korean-Popular"*), um gênero musical bastante popular na Coréia do Sul. As músicas do K-Pop são geralmente apresentadas por *boybands* e *girlbands*. Esses grupos costumam dançar coreografias pré-definidas e cantar letras também pré-definidas em suas apresentações.

Bruno já aprendeu algumas coreografias, mas ainda não é capaz de entender as letras das músicas. Por isso, ele está estudando o idioma coreano, em particular seu alfabeto, o Hangul. Bruno tem algumas imagens contendo letras de músicas escritas em coreano, e deseja saber como essas letras soariam para um americano. Ajude-o criando um programa que lê uma imagem contendo texto coreano e imprime seu som.

Neste problema, consideramos apenas uma **versão bastante simplificada** do Hangul. Existem apenas 7 letras neste alfabeto. Seus desenhos e seus sons são dados pela tabela a seguir:

Letter	Sound
ㅏ	a
ㅗ	o
ㅓ	eo
ㅜ	u
ㅣ	i
ㄱ	g
ㄴ	n

Para formar uma palavra, as letras são escritas em sequência, da esquerda para a direita. O som de uma palavra é igual à concatenação do som das letras que a formam, em ordem. Assim, de acordo com a tabela acima, a palavra ㄱㅏㄴㄱㅡㄴㅏㄴ, por exemplo, soa como "gangnan".

Também é possível substituir uma letra de uma palavra por um *bloco*, que consiste de duas letras, uma em baixo da outra. O som de um bloco é igual à concatenação do som da letra que está acima com o som da letra que está abaixo. Assim, por exemplo, o bloco $\begin{matrix}\top\\ \vdash\end{matrix}$ soa como "ua", e a palavra $\begin{matrix}\top\\ \vdash\end{matrix}\llcorner\ \begin{matrix}\dashv\\ |\end{matrix}$ soa como "uaneoi".

É dada uma matriz representando uma imagem preto-e-branca contendo uma única palavra. Os pixels brancos representam o fundo da imagem, enquanto os pretos representam a palavra em si. Determine como a palavra contida na imagem soa.

Entrada

A primeira linha de cada caso de teste contém dois inteiros n e m ($5 \leq n \leq 40, 7 \leq m \leq 520$), o número de linhas e o número de colunas da imagem. As próximas n linhas contém m

caracteres cada, representando a imagem. O caracter . indica um pixel branco, e o caracter # indica um pixel preto.

Restrições:

- As primeiras e últimas linhas e colunas da imagem estarão em branco;
- Haverá pelo menos uma letra na imagem;
- Todas as letras na imagem serão válidas;
- O comprimento de cada *traço* pode variar, mas a grossura será sempre de 1 pixel;
- Os comprimentos dos traços verticais de "a"e "eo"serão ímpares, e suas interseções com os traços horizontais estarão em seus pontos médios;
- Os comprimentos dos traços horizontais de "o"e "u"serão ímpares, e suas interseções com os traços verticais estarão em seus pontos médios;
- As letras/blocos que formam a palavra serão separadas por pelo menos uma coluna em branco;
- Não haverá nenhuma letra escrita em baixo de outra, exceto quando um bloco é formado. Isto indica que a palavra foi escrita na imagem sem quebras-de-linha;
- Em um bloco, os pixels mais abaixo da letra de cima e os pixels mais acima da letra de baixo estarão separados por pixels brancos;
- Em um bloco, os pixels mais à esquerda de ambas as letras estarão na mesma coluna;
- As letras/blocos que formam a palavra podem não estar alinhadas horizontalmente;
- A palavra pode não fazer sentido no verdadeiro coreano.

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

Saída

Para cada caso de teste, imprima o som da palavra dada.

Exemplo de entrada

```
5 29
.............................
.###.#..#...###.#....#...#...
...#.##.#.....#.#....##..#...
...#.#..###...#.###..#...###.
.............................
19 38
......................................
..#...................................
..#...................................
..#................#...................
..###########......#...................
..#................#...................
..#................#...................
..#................#...................
.....................#...................
.....................#...................
..#................#...................
..#...................................
..#.......................#............
..#.......................#............
..#.......................#............
..####....................#............
.........................#............
....................##############......
......................................
16 34
..................................
................................#.
....#######.....................#.
.......#........................#.
.......#..............##########.
.......#........................#.
.......#........................#.
.......#........................#.
.......#.........................
..................................
....#...........#.................
....#...........#......#..........
....#####.......#......#..........
....#...........#####..#..........
....#..................#..........
..................................
0 0
```

Exemplo de saída

```
gangnan
anio
uaneoi
```