MARATONA DE PROGRAMAÇÃO
InterFatecs

Final da II Maratona de Programação InterFatecs (2013)

17 de agosto de 2013

Caderno de Problemas

Este caderno contém 12 problemas.
Verifique se o caderno está completo – ele contém páginas numeradas de 1 a 23.

Informações Gerais

A) Sobre o arquivo de solução e a submissão

1. O arquivo de solução deve ter o mesmo nome que o especificado no enunciado (logo após o título).
2. Confirme se você escolheu a linguagem correta e está com o nome de arquivo correto antes de submeter a sua solução.

B) Sobre a entrada

1. A entrada de seu programa deve ser lida da entrada padrão (não use interface gráfica).
2. A entrada é composta por vários casos de teste, especificados no enunciado.
3. Em alguns problemas, o final da entrada coincide com o final do arquivo. Em outros, o final é especificado por alguma combinação de caracteres e/ou dígitos.

C) Sobre a saída

1. A saída de seu programa deve ser escrita na saída padrão.
2. Não exiba qualquer mensagem não especificada no enunciado.

Promoção:

Fatec Mogi das Cruzes — Centro Paula Souza — Governo do Estado São Paulo — Fundação FAT

Patrocínio:

SBC Sociedade Brasileira de Computação

CENGAGE Learning — Revista MundoJ — EF Englishtown

BRVANT Soluções Tecnológicas — terramar — Brasoftware

Agência de Inovação INOVA Paula Souza

Problema A

Função de espalhamento

Arquivo fonte: `midsquare.{c | cpp | java ou pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Uma das estratégias mais interessantes para armazenar dados com facilidade para posterior recuperação é por meio das chamadas tabelas de espalhamento, geralmente designadas *hash tables* em inglês. Nessa estratégia, definimos uma estrutura com espaço suficiente para guardar todos os dados que teremos e então subdividimos essa estrutura em partes menores, frequentemente chamadas de *buckets*.

Ao recebermos um registro de dados para ser guardado, usamos uma função para decidir em qual *bucket* aquele dado deve ser gravado. Quando precisarmos recuperar um registro, usamos a mesma função para descobrir onde ele deve estar armazenado, o que significa que não precisaremos procurar muito até encontrá-lo. Tipicamente o registro possui vários campos e usamos o valor de um deles para decidir onde o registro todo deve ser guardado e, por esse motivo, tal campo é chamado de campo chave do espalhamento (*hash key*).

Essa função meio mágica que mapeia *hash keys* para *buckets* é chamada de função de espalhamento (*hash function*) e precisa ser projetada cuidadosamente, para que tenha a capacidade de distribuir os valores o mais uniformemente possível entre os diversos *buckets* disponíveis na estrutura geral de armazenamento criada.

Várias abordagens são comumente usadas para construir funções de espalhamento, e uma delas é a do chamado meio do quadrado, que é um método ao mesmo tempo engenhoso e simples. Pegamos o valor da chave desejada e o elevamos ao quadrado, considerando que esse quadrado terá sempre d dígitos, assumindo zeros à esquerda quando o quadrado original tiver menos que esses d dígitos.

Em seguida, pegamos os n dígitos centrais desse quadrado e os utilizamos como número do *bucket* onde aquela chave deve ser guardada. Por exemplo, suponhamos que temos 100 *buckets*. Eles serão numerados de 0 até 99, significando que o número do *bucket* possuirá 2 dígitos, ou seja, precisaremos de $n = 2$. Para gerenciar a distribuição dos registros, basta então computar o quadrado do valor do campo chave e extrair desse resultado os 2 dígitos centrais, que indicarão o número do *bucket* correspondente ao registro.

Se, por exemplo, tivermos que decidir qual o *bucket* referente ao registro de chave = 3456, primeiro elevamos esse valor ao quadrado, o que produz o número de 8 dígitos ($d = 8$) 11943936, cujos dois dígitos centrais formam o número 43, o que significa que é nesse *bucket* onde o registro de chave 3456 deve estar armazenado.

Se, nesse exemplo, considerássemos que o quadrado deveria ser visto com 10 dígitos (ou seja, $d = 10$), o quadrado seria 0011943936, e, nesse caso, os dois dígitos centrais formariam o número 94. Se tivéssemos 1000 *buckets*, precisaríamos de 3 dígitos ($n = 3$), pois os *buckets* estariam numerados de 0 a 999. O método do meio do quadrado é então interessante para esse tipo de aplicação, pois consegue gerar valores no intervalo de 0 a 10^n-1, com um bom índice de aleatoriedade.

Sua tarefa neste problema é, dados um valor chave, a quantidade de *buckets* e um comprimento d, determinar o número do *bucket* onde aquela chave deve ser guardada segundo o método do meio do quadrado.

Entrada

A entrada é iniciada por um inteiro Q, $0 < Q \leq 200$, que indica a quantidade de casos de teste a serem processados. Seguem-se Q linhas compostas por um inteiro K, $0 < K \leq 32000$, que representa o valor de chave a ser processada; um inteiro B, que representa a quantidade de *buckets*, onde B é na forma 10^X, com $0 < X \leq 5$; e um inteiro D, $0 < D \leq 10$, que indica o comprimento a ser considerado para o quadrado da chave. Um espaço em branco separa os inteiros contidos em cada linha.

Considere que se o comprimento *D* menos o comprimento requerido da chave produzir um valor ímpar, o ‘meio do quadrado’ deverá privilegiar os dígitos mais a direita, como, por exemplo, no quadrado 901234537, se precisarmos de uma chave de 4 dígitos, o resultado deverá ser 2345, e não 1234.

Saída

Para cada caso de teste, o programa deve imprimir um inteiro representando o número do *bucket* correspondente àquela chave. Esse inteiro não deve possuir zeros iniciais.

Exemplos

Entrada:

```
5
3456 100 8
3456 100 9
3456 100 10
3456 10 8
3456 10 9
```

Saída:

```
43
43
94
3
4
```

Problem B

Password Strength

Source file: `password.{c | cpp | java` *or* `pas}`

Author: Leandro Luque (Fatec Mogi das Cruzes)

The development of software systems requires developers to know and apply some basic principles related to security, generally specified as non-functional requirements. Among these principles, are those sometimes called AAA – Authentication, Authorization and Accounting.

Authentication provides a way of identifying an user. The most common strategy used for that involves an user name and a secret password. Despite the importance of security in software development, many users choose small and easy to grasp passwords, such as abc123 or 12345.

Aiming to avoid this lack of security, Luciano decided to implement a feature that evaluates the strength of a password. This feature checks some important aspects and classify a password in one of the following: (1) too weak, (2) weak, (3) regular, (4) strong and (5) very strong.

The following rules are used to define the strength of a password:

- Five points are attributed for each character/digit/symbol in the password, limited by a maximum of sixty points;
- Five points are attributed if the password contains only one uppercase letter. Ten points are attributed if the password contains two or more uppercase letters.
- Five points are attributed if the password contains only one lowercase letter. Ten points are attributed if the password contains two or more lowercase letters.
- Five points are attributed if the password contains only one digit. Ten points are attributed if the password contains two or more digits.
- Five points are attributed if the password contains any symbol different from a letter and a digit. Ten points are attributed if the password contains two or more symbols.

After adding the points, the following table is used to define the password strength:

Interval	Strength
[0-50)	Too weak (1)
[50-60)	Weak (2)
[60-80)	Regular (3)
[80-90)	Strong (4)
[90-100]	Very strong (5)

Input

The first line contains an integer N representing the number of test cases ($N > 0$). The following N lines contain a non-empty string representing a password with a maximum length of 20.

Output

For each test case output a line with an integer representing the number associated with the password strength (1, 2, 3, 4 or 5).

Samples

Sample input

```
4
a
aA
Aa@
aPoLoNi@2013#
```

Sample output

```
1
1
1
5
```

Problema C

Pai é pai!

Arquivo fonte: `pai.{c | cpp | java ou pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Eustáquio vai ser pai e está todo eufórico com a proximidade da chegada do primeiro filho. E está preocupado também ... Sua esposa esotérica faz questão de ter o parto normal, para poder fazer um mapa astral mais preciso do pimpolho. Segundo ela, a cesariana não seria natural, pois anteciparia o processo da natureza, e a data e hora do nascimento por meios artificiais como esse invalidariam o mapa astral, o que seria uma tragédia para ela.

O médico, muito compreensivo e resignado, já fez os exames e confirmou que o parto normal não oferecerá maiores riscos, então, agora é esperar. Eustáquio foi incumbido de filmar o parto, segurar a mão da esposa, tirar fotos, anotar a hora exata do nascimento e verificar se o bebê não vai ser trocado na maternidade, mas, na verdade, ele está mesmo é preocupado em chegar rápido ao hospital quando os primeiros sinais do parto iminente aparecerem.

Precavido, conseguiu um mapa da cidade contendo as ruas e as distâncias entre todos os trechos de sua malha viária. Ele agora pretende analisar os melhores trajetos para levar sua esposa rapidamente para o hospital com segurança. Obviamente ele poderia fazer isso na web, com algum serviço de mapas, mas ele tem dificuldade com o acesso à internet no dia-a-dia. Além disso, desconfia que estamos todos sendo vigiados quando acessamos tais serviços, segundo uma teoria da conspiração que o deixou muito impressionado.

Ele tem os dados da malha viária e quer um programa que, com base neles, indique, inicialmente, a menor distância para um trajeto capaz de levar a futura mamãe de um local específico até o hospital. Como ele conhece bem sua esposa, já sabe que terá inúmeros alarmes falsos da parte dela, que é particularmente sensível e ansiosa, então, ele também quer saber qual é a extensão do caminho mais curto para retornar do hospital para o local de onde ele saiu, já que as ruas não são, necessariamente, de mão dupla. E, já que você calculou pra ele a distância da ida e da volta, aproveite e determine qual é a distância total, considerando ida e volta.

Entrada

A entrada possui vários casos de teste. Cada caso se inicia com um inteiro Q ($0 \leq Q \leq 1000$), que indica a quantidade de pontos de cruzamento das ruas que formam a malha viária da cidade. Encontramos em seguida um inteiro T ($0 \leq T \leq Q*(Q-1)$), informando a quantidade de ligações entre os diversos pontos de cruzamento das ruas. Seguem-se T linhas, cada uma contendo três inteiros X, Y e L, representando respectivamente o ponto de origem da ligação, o ponto de destino e a distância entre eles. Considere que X e Y são valores entre 1 e 1000 e que a distância é um inteiro positivo menor que 10000. Após a descrição da malha viária, aparece um inteiro N, informando as consultas que Eustáquio quer fazer. Cada uma das N linhas seguintes possui dois inteiros O e D indicando, respectivamente, qual o ponto de onde o futuro papai partirá com sua aflita esposa rumo ao hospital e qual o ponto que corresponde ao hospital escolhido para o parto. O conjunto de entradas se encerra com um valor $Q = T = 0$.

Saída

Para cada caso de teste, o programa deve imprimir uma linha com a expressão "Caso 999", onde o 999 indica o número do caso de teste, que começa em 1. Em seguida, para cada consulta da entrada, imprimir uma linha com a expressão $C_i + C_v = C_t$, onde C_i é o comprimento do menor caminho de ida, C_v é o comprimento do menor caminho de volta e C_t é a soma de C_i com C_v. Separar os elementos com um espaço em branco.

Exemplos

Entrada:

```
8 13
```

```
1 3 40
2 4 33
4 2 32
2 8 31
8 1 60
3 4 20
8 4 50
3 6 40
4 5 25
6 5 25
5 7 25
6 7 40
7 8 55
2
3 7
7 3
5 8
1 2 5
4 1 8
3 1 6
2 3 6
2 5 8
3 4 4
3 5 4
5 4 5
2
3 1
5 3
0 0
```

Saída:

```
Caso 1
70 + 155 = 225
155 + 70 = 225
Caso 2
6 + 11 = 17
24 + 4 = 28
```

Problema D

Criptografando

Arquivo fonte: `cripto.{c | cpp | java ou pas}`

Autor: Julio Fernando Lieira (Fatec Lins)

Nestes tempos de espionagem digital, Maldi e Lalu, dois grandes mestres na arte de fazer pizza, resolveram se precaver e decidiram utilizar, para troca de mensagens sobre suas receitas de pizza, um esquema simples de criptografia: a Cifra de César. A técnica consiste em substituir cada letra do texto por outra letra do alfabeto que esteja um número fixo de posições à frente. Por exemplo, com uma troca de três posições, uma letra A no texto seria substituída pela letra D, o B por E, e assim por diante, veja o exemplo:

Texto Original:	`Pizza pra comer de montao!`
Texto Cifrado:	`Slccd sud frphu gh qrqwdr!`

Desse modo, a chave para cifrar ou decifrar uma mensagem é a quantidade de posições X que as letras devem ser trocadas (no exemplo, 3). Para criptografar (ou cifrar) basta substituir cada letra do texto original pela letra correspondente X posições à frente. Para decifrar faça o contrário, substitua cada letra do texto cifrado pela letra correspondente X posições atrás. Note que as letras em maiúsculo no texto original devem ser escritas em maiúsculo no texto cifrado e vice-versa. O mesmo ocorre com as letras minúsculas. Considere ainda que não se utiliza acentuação e nem cedilha e qualquer caractere diferente de letra deve ser copiado do texto original para o texto cifrado e vice-versa. O alfabeto utilizado é o definido no acordo ortográfico de 1990 composto de 26 letras maiúsculas e minúsculas (A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z).

Precavidos, Maldi e Lalu ainda utilizam uma senha, que eles combinam sempre que se encontram pessoalmente, a partir da qual se pode determinar a quantidade de posições X a ser utilizada para cifrar e decifrar as mensagens. Essa senha, de no máximo 10 caracteres, é composta somente por letras (maiúsculas e minúsculas) e os dígitos de 1 a 9 (o zero não entra). Dada a senha, determina-se a quantidade de posições X realizando o seguinte cálculo: para cada letra minúscula encontrada na senha, soma-se o valor 2; para cada letra maiúscula soma-se o valor 5 e para cada dígito soma-se o seu próprio valor; a soma total deve ser feita e deve ser calculado o módulo 26 (resto da divisão inteira por 26). Veja o exemplo:

senha: ZeDi74eu

soma: 5 + 2 + 5 + 2 + 7 + 4 + 2 + 2 = 29

módulo 26 (resto de 29/26) = 3 ← quantidade de posições

O código para cifrar as mensagens já está pronto, mas a empresa que o desenvolveu foi acusada de corrupção e fechou sem terminar o projeto. Sua missão: desenvolver o código para decifrar as mensagens.

Entrada

A entrada é composta por vários casos de teste e termina com o fim do arquivo de entrada. Cada caso de teste é composto por duas linhas. A primeira linha contém a senha para calcular a quantidade de posições utilizada para cifrar/decifrar as mensagens. A segunda linha contém a mensagem cifrada de, no máximo, 1000 caracteres.

Saída

Para cada caso de teste, imprima na saída padrão a mensagem original.

Exemplos

Entrada:

```
zEdI74eu
Slccd sud frphu gh prqwdr!
DiLuViu
Cloob x jxppx zlj jliel ab qljxqb,
Vale23DuTo
tmuyi, nyrxi gifspe i jpeqfi gsq gsrleuyi.
EmPizza7
Dvcsb dpn nvttbsfmb f mfwf bp gpsop.
```

Saída:

```
Pizza pra comer de montao!
Forre a massa com molho de tomate,
pique, junte cebola e flambe com conhaque.
Cubra com mussarela e leve ao forno.
```

Problem E

Collision Detection

Source file: `collision.{c | cpp | java` *or* `pas}`

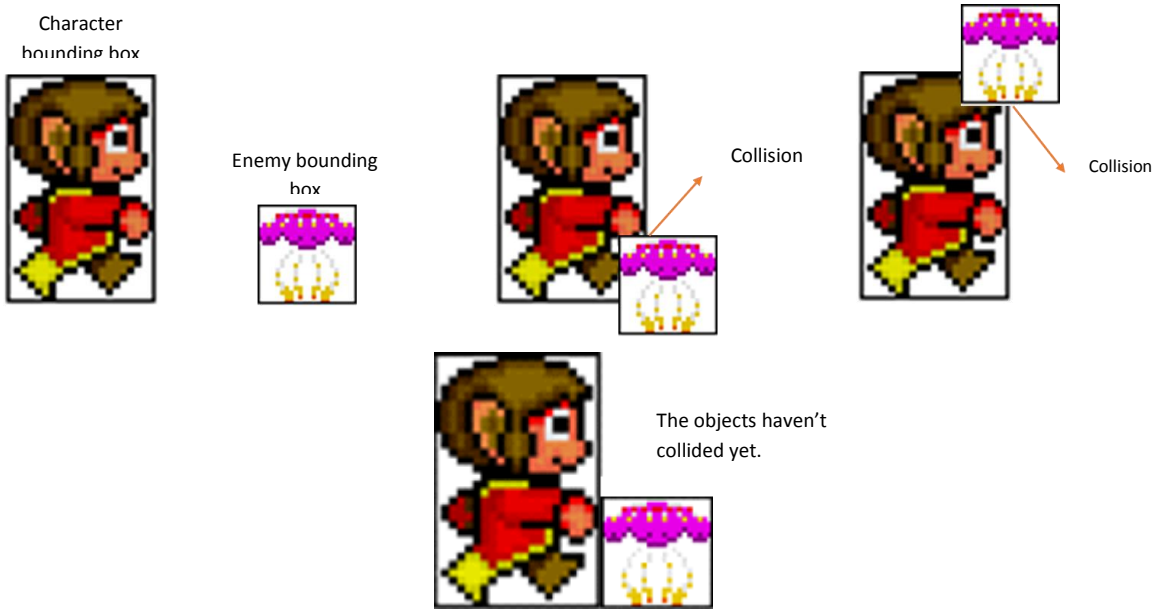
Author: Leandro Luque (Fatec Mogi das Cruzes)

Game development is one of the fields where computer professionals can work. It involves knowledge of design, programming languages and logic, physics, computer graphics, among others.

One of the essential functionalities for practically all kinds of games is collision detection. During a game, it must be continuouslly verified whether or not two objects collided, such as when the player car hits a tree and the car slows down or suffers some damage.

John Atari is developing a platform game (e.g. Alex Kidd in Miracle World – Am I getting old? Does anyone remember this game?) and he needs to implement a collision detection functionality to know when a character reaches the ground, after jumping, or touches an enemy and must lose energy. Searching for available algorithms for the collision detection, Atari found out that the simplest way is to use a technique called Bounding Box.

According to this technique, each game object that can collide to others must be surrounded by a box and, if the intersection between the bounding boxes of two objects is not empty, then they collided (see the image below).



As in John Atari's games there are objects both circular and polygonal, he needs your help to write a program that, given two objects, determines whether or not they collided using the bounding box technique.

Input

The input consists of many test cases. The first line of each test case specifies the number N ($N \geq 2$) of objects that will be informed (an object can be a character, an enemy or others). The following N lines contain a character T, which determines the type of object: 'C' (capital and without quotes) for a circle, and 'P' (capital and without quotes) for a polygon. 'C' will be followed by integers that determine the positions X and Y of the center ($-100 \leq X$ and $Y \leq 100$), and the radius R ($0 < R \leq 100$). 'P' will be followed by an integer specifying the number V of vertices ($V \geq 3$) and $2*V$ integers that determine the positions X and Y of each vertex ($-100 \leq X$ and $Y \leq 100$). The character T and the integers are separated by a single space. The last test case has N=0.

Output

For each test case, output a line containing a capital letter ‘S’ (without quotes), in case of collision between any of the specified objects, using the bounding box technique. Otherwise, output a capital letter ‘N’ (without quotes).

Samples

Sample input

```
3
C 10 15 5
C -5 -3 10
P 4 1 1 2 2 3 1 2 0
2
P 3 1 1 2 2 3 1
P 3 10 11 12 12 13 11
0
```

Sample output

```
S
N
```

Problema F

Pi

Arquivo fonte: `pi.{c | cpp | java ou pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Números irracionais como π (que, a propósito, vale aproximadamente 3.1415926535) são obtidos por meio de métodos de aproximação numérica. Existem diversos métodos capazes de produzir valores satisfatórios de π utilizando sequências numéricas relativamente curtas. Uma das sequências mais antigas foi proposta pelo matemático francês François Viète em 1593, baseada na observação de polígonos com 2*n* lados. Essa aproximação é dada por:

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots .$$

Figura 1: Sequência de Viète para o cálculo de π.

Observe que se trata de uma sequência de multiplicações de termos que pode prosseguir indefinidamente. No exemplo da figura 1, mostramos 3 termos e, se pararmos por aí, teremos o valor `3.1214451523`. Se acrescentarmos um quarto termo da sequência lógica, teremos um resultado ainda mais próximo do real valor de π, que será, no caso, `3.1365484905`. Quanto maior a quantidade de termos utilizados, mais próximo de π será o resultado obtido, e devemos parar com o processo no momento em que estivermos satisfeitos com a aproximação conseguida.

Este problema pede que você construa um programa que determina o valor de π dado pela fórmula de Viète, utilizando uma quantidade arbitrária de termos informada. Para evitar problemas com arredondamento, você deverá utilizar apenas valores reais de precisão dupla nos cálculos.

Entrada

A entrada é iniciada por um inteiro Q, $0 < Q \leq 50$, que indica a quantidade de casos de teste a serem processados. Seguem-se Q linhas, cada uma contendo um inteiro T, $0 < T \leq 50$, que indica a quantidade de termos a serem considerados na sequência de cálculo.

Saída

Para cada caso de teste, o programa deve imprimir um real de precisão dupla com o resultado produzido pelo método de Viète. O valor deverá ser exibido com 10 casas depois da vírgula.

Exemplos

Entrada:

```
3
3
4
5
```

Saída:

```
3.1214451523
3.1365484905
3.1403311570
```

Problema G

Pirocóptero

Arquivo fonte: `pirocoptero.{c | cpp | java` *ou* `pas}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Enquanto assistia pela milésima vez ao DVD da Galinha Pintadinha com a sua filha ("Pirulito que bate bate, pirulito que já bateu..."), a gerente Paula Notoba teve a ideia de produzir um pirulito que fez muito sucesso na sua época: o pirocóptero. Acompanhava o pirocóptero uma hélice de plástico que podia ser colocada junto com o cabo do pirulito para formar um brinquedo que saia voando ao ser girado. Segundo uma lenda da época em que o brinquedo foi comercializado, algumas crianças desapareceram após girar o pirocóptero. Pais e amigos disseram ter visto as crianças voando, assim como no comercial televisionado.

Lendas à parte, Paula Notoba deseja inovar e criar pirocópteros de tamanhos variados. Ela encomendou uma pesquisa de mercado e descobriu que as pessoas estão geralmente dispostas a pagar mais por pirocópteros maiores. Pesquisando fornecedores, ela descobriu que pode comprar barras de cabos de pirulitos de tamanhos variados e que pode cortar estas barras do tamanho que desejar – para formar os cabos dos pirulitos.

Ela está com uma dúvida cruel em relação ao corte das barras e pediu a sua ajuda para escrever um programa que, dado um tamanho de barra e o lucro que ela pode ter vendendo diferentes tamanhos de cabos de pirulitos, determina a estratégia ótima de corte, especificando o lucro máximo que pode ser obtido.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste contém um inteiro C ($1 \leq C \leq 5000$), que representa o comprimento da barra. As C linhas seguintes, numeradas de 1 a C, contêm um inteiro V cada ($1 \leq V \leq 10000$), indicando o preço de mercado de um pirocóptero com cabo de comprimento igual ao número da linha (ou seja, a primeira linha contém o preço de mercado de um pirocóptero com cabo de tamanho 1; a segunda linha contém o preço de mercado de um pirocóptero com cabo de tamanho 2, e assim por diante).

Saída

Para cada caso de teste, imprima uma linha contendo o lucro máximo que Paula Notoba pode obter cortando a barra.

Exemplos

Exemplo de entrada

```
4
1
2
4
4
```

Saída para o exemplo de entrada

```
5
```

Problema H

Quarter back rating

Arquivo fonte: `qbrating.{c | cpp | java ou pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Onival é um aficionado pelo futebol americano e acompanha com entusiasmo a temporada da NFL (*National Footbal League*), tanto pela internet como pela televisão. Já está, inclusive, pegando o vício tipicamente norte-americano pelas estatísticas ou *ratings*. O *rating* de um jogador é uma medida de seu desempenho, que é calculado conforme a posição em que ele atua. Jogadores com bom *rating* teoricamente possuem um retrospecto melhor que outro jogador de mesma posição com *rating* inferior no mesmo período. Então tome *rating*: é um tal de *rating* disto, *rating* daquilo, um inferno!

Como o jogador individualmente mais importante no ataque é o *quarter back*, o jogador que arma e inicia as jogadas de ataque de uma equipe, ele recebe particular atenção dos fãs, Onival incluído. E nosso amigo descobriu a fórmula de cálculo do *rating* dos *quarter-backs* da NFL em um *site* da Web e quer, agora, um programa para calcular o *rating* de um jogador em um determinado período.

A fórmula de cálculo depende de quatro indicadores:

a) Índice de passes completos por tentativa, dado por `(((Comp/Att) * 100) -30) / 20`, onde `Comp` é a quantidade de passes completos e `Att` é o número total de tentativas.

b) Índice de jardas conseguidas por tentativa, dado por `((Yards/Att) - 3) / 4`, onde `Yards` é a quantidade total de jardas obtidas e `Att` é o número total de tentativas.

c) Índice de passes para *touchdown* por tentativa, calculado por `((TDs/Att) * 100) / 5`, em que `TD` representa a quantidade de *touchdowns* conseguidos e `Att` é o número total de tentativas.

d) Índice de interceptações por tentativa, calculado por `(9.5 - ((Int/Att) * 100)) / 4`, com `Int` indicando o total de interceptações e `Att` o número total de passes tentados.

Nenhum desses índices pode ser negativo ou superior a 2,375. O *rating* do *quarter back* é então dado por `(a + b + c + d)/0.06`, onde `a`, `b`, `c` e `d` são os indicadores apresentados anteriormente.

Onival conseguiu reunir dados estatísticos com os totais referentes a `Comp`, `Att`, `Yards`, `TD` e `Int` de diversos *quarter backs* em cada temporada. Você deve fazer um programa que receba inicialmente esses dados históricos e depois responda a consultas sobre o *rating* de um jogador em um determinado período. Obviamente não é necessário que você seja um *expert* em futebol americano para dar conta desta tarefa.

Entrada

A entrada é iniciada por um inteiro E, $0 < E \leq 1000$, que indica a quantidade de linhas com estatísticas coletadas por Onival. Seguem-se E linhas compostas por uma palavra de até 20 caracteres contendo o nome do jogador, e inteiros positivos Y, C, A, J, T e I, todos eles inferiores a 10000 e que representam, respectivamente, o ano, a quantidade de passes completos, o total de tentativas, a quantidade total de jardas conseguidas, o número de *touchdowns* obtidos e a quantidade de interceptações naquela temporada.

Após as estatísticas, temos um inteiro Q, $0 < Q \leq 1000$, informando a quantidade de consultas a serem processadas pelo programa. As Q linhas seguintes são compostas por uma palavra de até 20 caracteres e dois inteiros P e U, $0 < P, U \leq 10000$, que representam, respectivamente, o nome do jogador, o primeiro ano do período e o último ano do período a ser considerado.

Saída

Para cada consulta informada, o programa deve imprimir um número real com uma casa após a vírgula, contendo o *rating* do jogador no período informado.

Exemplos

Entrada:

```
28
Marino 1991 318 549 3970 25 13
Moon 1991 404 655 4690 23 21
Elway 1991 242 451 3253 13 12
Testaverde 1991 166 326 1994 8 15
Favre 1991 0 4 0 0 2
Moon 1992 224 346 2521 18 12
Favre 1992 302 471 3227 18 13
Marino 1992 330 554 4116 24 16
Testaverde 1992 206 358 2554 14 16
Elway 1992 174 316 2242 10 17
Marino 1993 91 150 1218 8 3
Elway 1993 348 551 4030 25 10
Testaverde 1993 130 230 1797 14 9
Moon 1993 303 520 3485 21 21
Favre 1993 318 522 3303 19 24
Bledsoe 1993 214 429 2494 15 15
Favre 1994 363 582 3882 33 14
Marino 1994 385 615 4453 30 17
Elway 1994 307 494 3490 16 10
Moon 1994 371 601 4264 18 19
Bledsoe 1994 400 691 4555 25 27
Testaverde 1994 207 376 2575 16 18
Favre 1995 359 570 4413 38 13
Moon 1995 377 606 4228 33 14
Marino 1995 309 482 3668 24 15
Testaverde 1995 241 392 2883 17 10
Elway 1995 316 542 3970 26 14
Bledsoe 1995 323 636 3507 13 16
3
Elway 1993 1995
Moon 1991 1991
Favre 1990 1995
```

Saída:

```
88.4
81.7
86.8
```

Problema I

Quebra cabeças

Arquivo fonte: `quebra.{c | cpp | java ou pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Ladislau ganhou um joguinho de quebra-cabeças muito interessante. São vários blocos em formato retangular, parecidos com as peças de dominó, com sua face principal dividida em duas partes, cada uma contendo uma sequência de símbolos aleatórios, como mostra a figura 1.

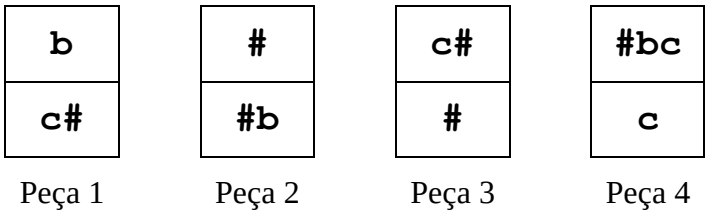


Figura 1: Exemplo de peças do jogo de Ladislau

O objetivo do jogo é criar combinações de peças em que a sequência formada pelos caracteres da parte de cima das peças é igual à sequência produzida pela parte de baixo dessas peças. Por exemplo, se colocarmos lado a lado as peças 2, 1, 3, 2, 4, como mostra a figura 2, teremos na parte de cima a sequência **`#bc###bc`**, que corresponde exatamente à sequência produzida pelos símbolos da parte de baixo. Observe que podemos usar mais de uma vez a mesma peça, pois o jogo traz várias cópias de cada uma, e que a ordem em que as peças aparecem é importante para o resultado final.

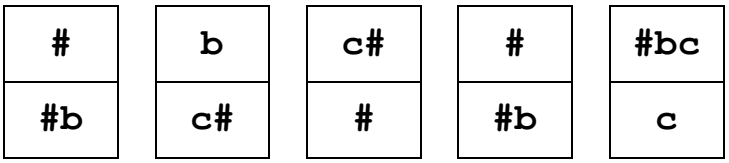


Figura 2: Exemplo de combinação de peças válida

Existem várias versões para o problema, como, por exemplo, gerar a maior ou a menor combinação de peças que produza uma configuração válida, ou ainda contar quantas são possíveis para uma determinada quantidade de peças. Sua tarefa neste problema é, dado um conjunto de peças, verificar se uma combinação é ou não válida para o jogo.

Entrada

A entrada possui vários casos de teste. Cada caso se inicia com um inteiro P, $0 \leq P \leq 100$, que indica a quantidade de peças distintas que o jogo possui. Seguem-se P linhas, cada uma contendo duas strings de até 20 caracteres de comprimento cada separadas por um espaço em branco, que indicam respectivamente o conteúdo da parte de cima e da parte de baixo de uma peça do jogo. Assuma que a primeira linha descreve a peça 1, a segunda linha descreve a peça 2, e assim sucessivamente. Após a descrição das peças, encontra-se um inteiro Q, $0 < Q \leq 50$, que indica a quantidade de combinações de peças a serem avaliadas. As Q linhas seguintes iniciam-se com um inteiro N, $0 \leq N \leq 200$, informando a quantidade de peças da combinação, após o que temos X, $0 < X \leq 100$, inteiros representando as peças na ordem em que foram colocadas. O conjunto de entradas se encerra com um inteiro $P = 0$.

Saída

Para cada caso de teste, o programa deve imprimir uma linha com a expressão “Caso 999”, onde o 999 indica o número do caso de teste, que começa em 1. Em seguida, para cada consulta da entrada, imprimir ‘Ok’ (sem aspas), se a sequência for válida, e um asterisco ‘*’ (sem aspas), em caso contrário.

Exemplos

Entrada:

```
4
b c#
# #b
c# #
#bc c
3
5 2 1 3 2 4
5 1 2 2 3 4
2 3 2
4
a ba
bba aaa
aab b
ab bba
1
3 1 2 3
3
b bbb
babbb ba
ba a
2
4 2 1 1 3
3 3 2 2
0
```

Saída:

```
Caso 1
Ok
*
*
Caso 2
*
Caso 3
Ok
*
```

Problema J

Palavras e Sistemas Numéricos

Arquivo fonte: `palavras.{c | cpp | java ou pas}`

Autor: Anderson Viçoso de Araújo (Fatec São José dos Campos)

Um sistema de numeração é um sistema em que números são representados por numerais de uma forma consistente - um numeral é um símbolo ou grupo de símbolos.

O sistema de numeração pode ser visto como o contexto que permite ao numeral "11" ser interpretado como o numeral romano para dois, o numeral binário para três ou o numeral decimal para onze.

Diferentes sistemas de numeração foram utilizados durante a evolução da humanidade. Os mais comuns encontrados na computação são: o binário (0 e 1) e o hexadecimal (0123456789ABCDEF), mas existem outros, como, por exemplo, o Hexatrigesimal, que contém todos números (0-9) e letras (A-Z).

Seu trabalho é descobrir se a sequência de entrada de caracteres de entrada é válida de acordo com um número correspondente ao sistema numérico a ser adotado. Além disso, os números da tabela seguinte podem ser considerados iguais às letras correspondentes na tabela, e vice-versa.

Número	Letra
0	O
1	I
3	E
4	A
8	B

Tabela 1: Números e letras que podem ser considerados iguais.

Por exemplo, a sequência de caracteres “abc” pode ser construída a partir dos sistemas numéricos: tridecimal, tetradecimal, pentadecimal, hexadecimal e maiores. Mas não é possível criar a mesma cadeia de caracteres através do sistema octal, decimal, duodecimal, ou menores.

Já a sequência IOIO, pode ser representada em qualquer sistema numérico, até mesmo o binário, pois a letra I pode ser considerada igual ao número 1 e a letra O igual ao número 0 (veja a tabela 1).

Aqui as letras devem ser tratadas ignorando a diferenças entre maiúsculas e minúsculas.

Entrada

Existem vários casos de teste, cada um em uma linha, onde cada linha contém: um número N ($0 < N \leq 36$), que corresponde ao número de símbolos contidos no sistema numérico que deve ser considerado. Em seguida, a entrada apresenta um número M ($0 < M < 100$), que representa o número de sequências de caracteres correspondentes ao caso de teste. Por fim, tem-se M sequências de caracteres a serem lidas. Cada sequência pode conter no máximo 100 caracteres. Cada caractere pode ser um destes:

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Entre N, M e cada sequência de caracteres, existe um espaço em branco. A leitura de um número N igual a 0 indica que não temos mais entradas.

Saída

Para cada caso de teste, você deve imprimir uma linha com vários caracteres (‘S’ ou ‘N’), separados por espaço, indicando a validade de cada uma das sequências informadas. Uma letra ‘S’ (maiúscula e sem aspas) indica que uma sequência de caracteres é válida, enquanto ‘N’ (maiúsculo e sem aspas) indica que uma sequência é inválida.

Exemplos

Entrada:

```
16 4 f4ca A8c teste CAsa
8 3 ABC teste 1010
0
```

Saída:

```
S S N N
N N S
```

Problema K

Reforma da Pirâmide

Arquivo fonte: `piramide.{c | cpp | java ou pas}`

Autor: Anderson Viçoso de Araújo (Fatec São José dos Campos)

Pirâmides construídas há milhares de anos com blocos cúbicos de calcário precisam de reformas periódicas, já que vários blocos podem apresentar trincas e entrar em colapso, o que levaria à destruição destes monumentos históricos. A técnica de prospecção sonora pode ser utilizada para classificar os blocos de uma pirâmide segundo o risco de ruptura, identificando quais devem ser removidos.

Para especificar claramente quais blocos devem ser removidos, pode ser utilizada uma numeração que identifica o nível no qual o bloco se encontra e a posição do bloco dentro do nível. Os níveis de uma pirâmide são numerados a partir da base (iniciando em 0), conforme ilustrado na figura 1.

Figura 1. Estrutura de níveis da pirâmide.

A base da pirâmide (nível 0) possui B blocos de comprimento por B de largura. Portanto, a quantidade de blocos na base é igual a B*B (se B for igual a 6, a base possuirá N=36 blocos). O nível logo acima da base possui uma fileira de blocos a menos em cada lado da pirâmide, totalizando, portanto, (B-2)*(B-2) blocos (no nosso exemplo com B=6, o nível 1 possuiria N=16 blocos). O nível logo acima deste possui uma fileira de blocos a menos em cada lado da pirâmide, e assim sucessivamente.

A posição de um bloco é determinada por três números inteiros. O primeiro e o segundo podem ser obtidos a partir da visão superior da pirâmide, o que permite associá-la a uma matriz. O terceiro deles especifica o nível no qual o bloco se encontra, conforme ilustrado pela figura 2. Nesta figura, são apresentadas as posições de três blocos, um em cada nível.

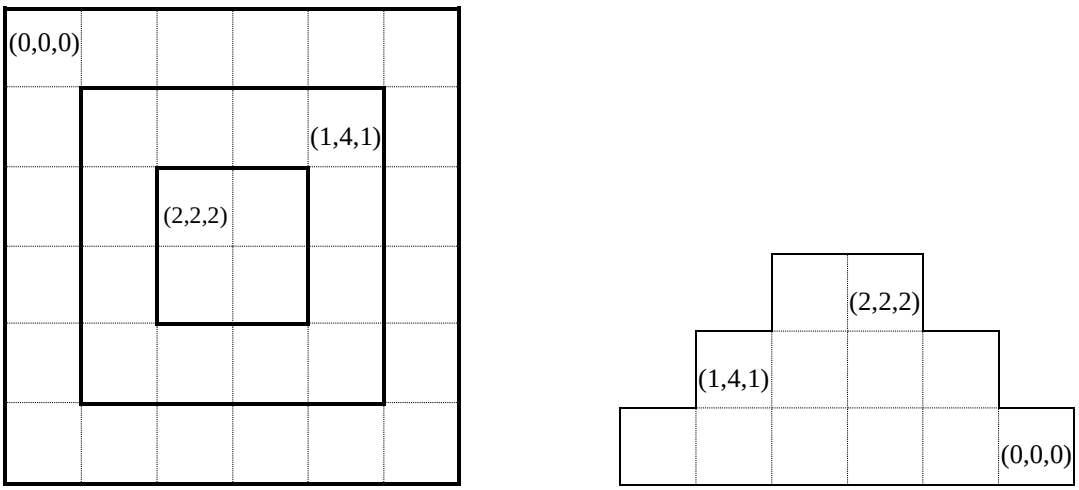


Figura 2. Visão superior e frontal da pirâmide.

Um empresa especializada em reformar pirâmides definiu algumas regras para a remoção dos blocos. Caso o bloco a ser removido esteja localizado na borda, ou seja, se não existir qualquer outro em cima dele, apenas ele deverá ser removido. Caso contrário, será necessário remover, no nível imediatamente superior ao bloco danificado, todos os blocos localizados a uma distância horizontal de até 1 bloco.

Assim sendo, caso seja necessário remover o bloco (0,0,0), apresentado na figura 2, apenas ele será removido. Caso seja necessário remover o bloco (2,2,0), deverão ser removidos, além dele, os 9 blocos localizados logo acima dele, no nível 1 e, a partir da remoção destes blocos, seguindo a mesma regra, deverão ser removidos todos os blocos do nível 2. Fica claro que, dado que um bloco foi removido no nível i, poderão ser removidos, no máximo, outros 9 blocos no nível i+1 (figura 3)

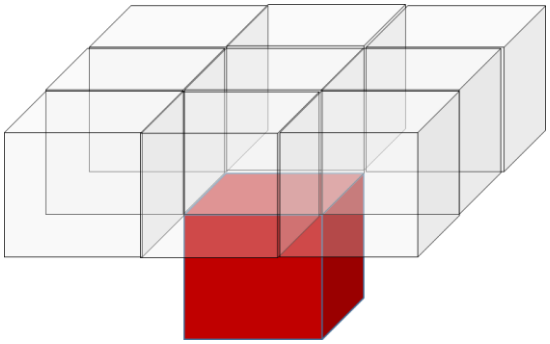


Figura 3. Exemplo de bloco a ser removido no nível i (escuro) e blocos que devem ser removidos por estarem a uma distância de até 1 no nível i+1 (claros).

A seguir, é apresentado um pequeno exemplo para a remoção do bloco na linha 3, coluna 3 e nível 0 (base):

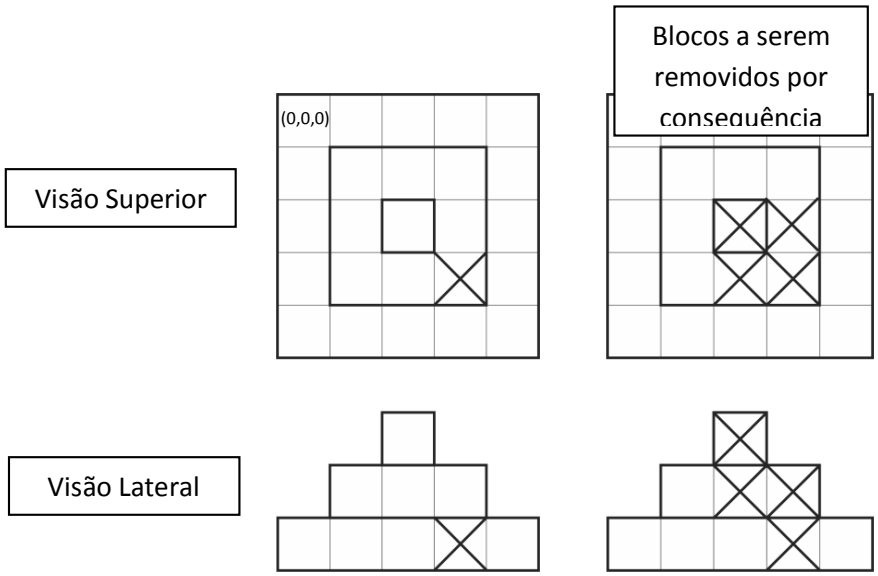


Figura 4. Exemplo de remoção do bloco na linha 3, coluna 3 e nível 0.

Para o exemplo apresentado nas imagens acima, o número correspondente ao tamanho da base da pirâmide foi 25 e a posição do bloco a ser removido foi 3,3,0. Neste caso, devem ser removidos 6 blocos no total, sendo eles: “1,1,1”, “1,2,1”, “2,1,1”, “2,2,1”, “0,0,2”, e o próprio “3,3,0”.

Escreva um programa de computador para auxiliar a empresa de reforma de pirâmides na definição do esforço necessário para a empreitada.

Entrada

Existem vários casos de teste, cada um em uma linha, onde cada linha contém: um número N ($0 < N < 10000$), que corresponde ao número de blocos da base da pirâmide. N sempre será um quadrado perfeito, ou seja, um número inteiro não negativo que pode ser expresso como o quadrado de outro número inteiro.

Depois, temos um inteiro M ($0 < M < 100$), que corresponde ao número de blocos a serem removidos, seguido das posições correspondentes dos blocos. Cada posição de um bloco possui 3 coordenadas, “X,Y,Z”, onde cada coordenada está separada por uma vírgula, sem as aspas duplas e sem espaço. A coordenada Z igual a 0 corresponde à base da pirâmide. Um nível acima da pirâmide contém a

coordenada *Z* igual a 1 e assim por diante. Leve em consideração que, em alguns casos, a técnica de prospecção sonora falha e indica um bloco inválido para remoção.

Entre *N*, *M* e cada posição de um bloco existe um espaço em branco. Assim que for lido um número *N* igual a zero, não existirão mais entradas.

Saída

Para cada caso de teste, você deve imprimir uma linha indicando a altura da pirâmide, seguido pelo número total de blocos que devem ser removidos da pirâmide. Ambos os números devem ser separados por um espaço em branco.

Exemplos

Entrada:

```
4 1 0,0,0
9 2 0,0,0 1,1,0
25 3 2,2,0 2,2,1 3,3,0
16 1 2,1,0
36 1 2,3,0
0
```

Saída:

```
1 1
2 3
3 12
2 5
3 14
```

Problema L

Decoração da Árvore de Natal

Arquivo fonte: `natal.{c | cpp | java ou pas}`

Autor: Anderson Viçoso de Araújo (Fatec São José dos Campos)

"HO HO HO", diz o bom velhinho. É quase natal. Stanislau Donetsk é um aposentado que vive no Brasil há mais de 20 anos e por ter descendência polonesa, por sorte, possui um rosto bem parecido com o conhecido papai Noel. Todo final de ano, ele consegue ganhar uma graninha fazendo apresentações e tirando fotos com as crianças em um shopping da cidade.

Antes de começar a se apresentar para as crianças, ele sempre ajuda na decoração do shopping, principalmente na decoração da árvore de natal. Como já é de praxe, o trabalho de pendurar as bolinhas e as estrelas da árvore de natal principal é de Stanislau.

Stanislau gosta de deixar estrelas e bolinhas misturadas na decoração da árvore, conforme ilustrado na figura 1. Ele utiliza uma divisão em níveis para definir quantas estrelas deve colocar na árvore. Em um nível i qualquer, exceto o primeiro, o número de estrelas deve ser igual a i-1. No primeiro nível, deve ser colocada uma única estrela.

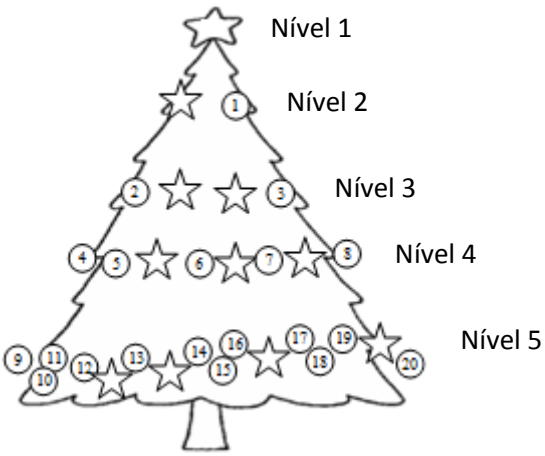


Figura 1. Exemplo de decoração da árvore de natal.

O restante das posições de cada nível deve ser preenchido com bolas de natal coloridas. As bolinhas devem ser numeradas de acordo com a sua ordem na árvore, começando em 1 e incrementando +1 conforme se aumenta o nível e se vai para a direita (veja a figura 1 como exemplo). Já as estrelas não possuem números.

Para definir a quantidade de enfeites que devem ser colocados em cada nível, deve-se imaginar que um enfeite (estrela ou bolinha) em um nível menor (no nível 1, por exemplo) está associado a M outros enfeites no nível imediatamente seguinte (nível 2, por exemplo), com exceção do maior nível da árvore (no exemplo, o nível 5), cujos enfeites não estão associados a nenhum outro abaixo. Esta associação é do tipo 1-n, de tal forma que, se o enfeite A estiver associado aos enfeites B e C, então, estes dois últimos não estarão associados a nenhum outro acima, além de A.

Aí é que entra o seu trabalho, ajudar Stanislau a descobrir em que nível ficarão certas bolas numeradas.

Entrada

Existem vários casos de teste, cada um especificado em uma linha. Para cada caso de teste, são especificados: um número N ($0 < N < 10$), que corresponde ao número de níveis que a árvore tem (lembrando que a árvore é completa); um número M ($0 < M < 10$), que representa a quantidade de enfeites de um nível i+1 com os quais um enfeite do nível i está associado; um número inteiro X ($0 < X$

< 100), que indica a quantidade de bolinhas das quais se deseja saber o nível; e *X* inteiros indicando um número de bolinha cada.

Saída

Para cada caso de teste, você deve imprimir uma linha indicando o nível na árvore de cada bolinha informada na entrada, lembrando que o topo da árvore corresponde ao nível 1. Caso a bolinha não exista para a árvore, imprima 0 (zero).

Exemplos

Entrada:

```
4 3 3 3 15 35
5 2 3 15 5 124
0
```

Saída:

```
3 4 0
5 4 0
```