A Junina

2^a Seletiva Interna – 2012/1

Sevidor BOCA:

`http://10.20.107.205/boca/`
(acesso interno)

`http://200.19.107.205/boca/`
(acesso externo)

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Lucas Negri (coordenação técnica), Alexandre Gonçalves Silva (revisão técnica) , Roberto Silvio Ubertino Rosso Jr., André Luiz Guedes (Dinf/UFPr), Omir Alves, Fernando Deeke Sasse

Lembretes:

- Aos *javaneiros*: **o nome da classe deve ser o mesmo nome do arquivo a ser submetido**. Ex: **classe petrus**, nome do arquivo **petrus.java**;
- 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.

Patrocinador e Agradecimentos

- YoungArts;
- DCC/UDESC;
- Rutes, Zanatta e Weskley que tiveram muito empenho neste início de ano;
- Alguns, muitos outros anônimos.

A Junina

2^a Seletiva Interna da UDESC

01 de junho de 2012

Conteúdo

1 Problema A: Alinhamento Apocalíptico

`Arquivo: apocalipitco.[c|cpp|java]`

Abacaxis e bananas são saborosos, porém são também perigosos. Uma antiga profecia diz que quando você alinhá-los em uma determinada ordem, o mundo será destruído. Em um dia nublado, como você está cansado deste mundo, você decide experimentá-los. Você iniciou com uma sequência de abacaxis e bananas, e neste caso, há um tipo de operação permitida:

Em cada passo, qualquer quantidade de itens consecutivos pode ser selecionada, substituindo-se estes itens selecionados pela mesma quantidade de frutos de um único tipo. Você não pode esperar para destruir o mundo, então você deseja conhecer a quantidade mínima de passos a fim de atingir o seu objetivo.

Entrada

A primeira linha do arquivo de entradas contém um único valor de t ($t \leq 100$), que representa o número de casos de teste. Cada caso de teste consiste de duas linhas, onde a primeira linha indica o padrão inicial, e a segunda linha indica o padrão não desejado (*evil pattern*) que pode acabar com o mundo. Ambas linhas contêm somente os caracteres A e B , onde A representa um abacaxi e o B representa uma banana. As duas linhas terão o mesmo comprimento (menor do que 200), e não há espaços em branco à direita ou à esquerda. Por exemplo:

```
2
BB
AA
BAAAB
ABBAA
```

Saída

Para cada caso de teste é gerada uma linha de saída contendo o número mínimo de passos para destruir o mundo. Por exemplo:

```
1
2
```

Sugestão: No Segundo caso do exemplo acima, você pode primeiro transformar a linha inteira de frutas em abacaxis, e depois alterar o segundo e o terceiro caracteres para banana, totalizando dois passos a serem executados.

2 Problema B: Caixas Sem Limites

`Arquivo: caixas.[c|cpp|java]`

Você se lembra do pintor Pares das Finais Mundiais da ACM do ano de 2345[1] Pares foi um dos inventores da *monocromia*, que significa que cada uma de suas pinturas tem uma única cor, mas em diferentes tons. Ele também acreditava na utilização de formas geométricas simples.

Há vários meses atrás, Pares pintava triângulos, em uma tela, de fora para dentro. Agora que os triângulos estão em baixa e os quadrados, em alta, suas mais recentes pinturas usam quadrados concêntricos, criados de dentro para fora! Pares começa a pintar sobre uma tela retangular dividida em uma grade quadrada perfeita. Ele seleciona um número de células individuais da grade para atuar como sementes centrais, e pinta estas com a cor mais escura. A partir de cada uma das sementes quadradas, Pares pinta um quadrado maior envolvente usando um tom mais claro, e repete com quadrados maiores envolventes, até que toda a tela é coberta. Cada quadrado é exatamente uma célula a mais na grade e um tom mais claro em relação àquele que está envolvendo. Quando quadrados se sobrepõem, a célula da grade é sempre preenchida com a tonalidade mais escura.

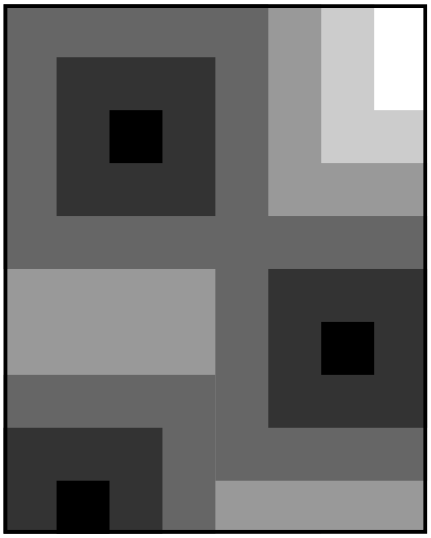

Figura 1: Exemplo de um dos mais recentes trabalhos de Pares, utilizando seis tonalidades de cor.

Depois de Pares decidir onde colocar os quadrados iniciais, a única parte difícil na criação dessas pinturas é decidir quantos tons diferentes de cor ele precisará. Para ajudar Pares, você deve escrever um programa que calcula o número de tons necessários para tal pintura, dado o tamanho da tela e os locais dos quadrados de sementes.

Entrada

O arquivo de entrada conterá múltiplos casos de teste. Cada caso de teste inicia com uma única linha contendo três números inteiros, m, n e s, separados por espaços. A tela contém exatamente $m \times n$ células de grade ($1 \leq m, n \leq 1000$), numeradas de $1, \ldots, m$ verticalmente

[1] Além do artista Pares, este problema realmente não tem nada a ver com o problema das Finais Mundiais. Então não se preocupe se você não esteve lá ou não leu esse problema antes. Na verdade, nós garantimos que qualquer conhecimento do problema mencionado não o ajudará, em nada, a resolver este!

e $1, \ldots, n$ horizontalmente. Pares começa a pintura com s ($1 \leq s \leq 1000$) células de semente, descritas nas s linhas de texto seguintes, cada uma com dois inteiros, r_i e c_i ($1 \leq r_i \leq m, 1 \leq c_i \leq n$), descrevendo a respectiva linha e coluna da grade de cada quadrado semente. Todos os quadrados sementes estão dentro dos limites da tela.

Uma linha em branco separa os casos de teste de entrada, como visto na amostra de entrada abaixo. Uma única linha com os números "`0 0 0`" marca o fim da entrada (não processe este caso).

Saída

Para cada caso de teste, seu programa deve imprimir um inteiro em uma única linha: o número de tons diferentes necessários à pintura descrita.

Exemplo de Entrada

A entrada para duas pinturas (incluindo o exemplo mostrado na figura acima), poderia ser assim:

```
10 8 3
3 3
7 7
10 2

2 2 1
1 2

0 0 0
```

Exemplo de Saída

A saída, correspondente à entrada acima, seria assim:

```
6
2
```

3 Problema C: Transporte de Carga

`Arquivo: carga.[c|cpp|java]`

Uma tarefa aparentemente simples tal como a condução de uma carga do ponto A ao ponto B, às vezes, pode ser muito complicada! Sistemas de GPS podem ser muito úteis para encontrar rotas, mas geralmente não levam em consideração todas as coisas. Por exemplo, quando buscam o percurso mais rápido, eles consideram o tempo de espera nos semáforos? E os congestionamentos? Talvez você possa escrever um software melhor? Neste problema, você receberá um mapa de ruas contendo: estradas, cruzamentos numerados (com semáforos) e dois armazéns. Sua tarefa é achar a rota mais rápida para levar um caminhão do Armazém A para o Armazém B. Mapas de ruas sempre são desenhados como uma grade, e tem esta aparência:

```
#A##0##1#
.#..#..#.
.#..#..#.
.###2#.B.
```

Figura 2: Um mapa de ruas contendo armazéns, estradas e cruzamentos.

A adjacência neste mapa é definida pela vizinhança a norte, sul, leste ou oeste. Os únicos símbolos que aparecem no mapa são os seguintes:

1. O caractere # que indica uma célula de estrada onde o caminhão pode se movimentar. As estradas são adjacentes a, no máximo outras duas estradas, interseção (cruzamento) ou células de armazéns;
2. Um único número `[0-9]` marca um cruzamento controlado por um semáforo. Cruzamentos são adjacentes a, pelo menos, três células de estradas. Os cruzamentos são numerados com números únicos e sequencialmente: nenhum número será exibido, a menos que todos os inteiros não negativos menor do que ele também apareçam no mapa. O comportamento dos semáforos será descrito a seguir:

   (a) Exatamente um caractere A marca a localização do armazém de onde seu caminhão parte;
   (b) Exatamente um caractere B marca a localização do armazém para onde você deseja enviar sua carga;
   (c) O caractere "." (ponto) é apenas grama. Você não pode trafegar neles. Você tem um caminhão de carga que parte do armazém A, e você está tentando levá-lo para o armazém B de acordo com as regras a seguir. Para simplificar, podemos também discretizar o tempo em unidades atômicas que chamamos de turno ou vez;
   (d) Em cada turno, você pode mover o caminhão para uma célula adjacente que pode ser estrada, cruzamento ou armazém, ou ainda, simplesmente ficar na mesma célula;
   (e) Um caminhão só pode se mover para um cruzamento se o semáforo estiver verde para a direção que se desloca o caminhão naquele turno. Entretanto, um caminhão numa célula de cruzamento pode sair em qualquer direção a qualquer tempo.

Os cruzamentos com semáforos permitem periodicamente o fluxo leste-oeste ou norte-sul, mas nunca ambos ao mesmo tempo. Eles são descritos por uma direção inicial e dois números que indicam respectivamente os períodos (em vezes/turnos) de direção leste-oeste e norte-sul. Por exemplo, um cruzamento inicialmente com sinal verde na direção norte e sul, descrito pelos números "2 3" será a luz verde acesa para as faces norte e sul nos turnos 1-3 inclusive, e verde para as faces leste e oeste nos turnos 4-5 e outra vez para norte e sul nos turnos 6-8, etc.

Entrada

O arquivo teste de entrada contém múltiplos casos de teste. Cada caso de teste iniciando com uma única linha contendo dois inteiros, m e n, separados por espaços. O mapa de ruas/estradas consiste em m linhas (leste-oeste) e n colunas (norte-sul) de uma grade de células ($2 \leq m, n \leq 20$). As m linhas seguintes contêm n caracteres cada, os quais descrevem o mapa usando os símbolos definidos no enunciado do problema. Para cada cruzamento numerado que aparece no mapa, em ordem ascendente iniciando em `0`, existirá uma linha de texto com o número do cruzamento seguido por um caractere '-' ou '|', e dois inteiros, a_i e b_i ($1 \leq a_i, b_i \leq 100$). A duração (em turnos/vezes) dos períodos da luz|(do sinal) respectivamente em leste-oeste e norte-sul. Um '-' indica que o semáforo está inicialmente verde na direção leste-oeste, enquanto um '|' significa que o semáforo está inicialmente verde na direção norte-sul. Um linha em branco separa os caso de teste de entrada, com mostra o exemplo a seguir. Uma linha única com os números `0 0` marca o fim da entrada. Não processe este caso.

Saída

Para cada caso de teste seu programa deve imprimir um inteiro em uma única linha: o número mínimo de turnos (vezes) que demora para levar seu caminhão do Armazém A para o Armazém B. Se não for possível chegar ao Armazém B, imprima apenas a palavra `impossible`. A saída para o exemplo de entrada deve ter a seguinte aparência:

Exemplo de Entrada

```
3 4
A##B
#..#
####

4 9
#A##0##1#
.#..#..#.
.#..#..#.
.###2#.B.
0 - 1 17
1 | 3 5
2 - 2 4

2 2
A.
.B

0 0
```

Exemplo de Saída

```
3
17
impossible
```

4 Problema D: *Deadlines* do Professor ĈĈ

`Arquivo: deadlines.[c|cpp|java]`

Ao contrário da crença popular, deixar as coisas para última nem sempre é uma boa atitude. Ao longo de seus anos como professor da UDESC, o professor ĈĈ constatou que, apesar de seus melhores esforços, seus trabalhos sempre ficavam para última hora. Era uma forma que ĈĈ se ocupava em preencher o tempo disponível com as suas muitas atividades. A fim de melhorar sua eficiência no dia–a–dia, o professor ĈĈ decidiu aprender a *arte da procrastinação*.

ĈĈ tem n atribuições a serem feitas na próxima semana. A $i-$ésima atribuição leva x_i unidades de tempo e deve ser finalizada dentro tempo t_i. ĈĈ só pode trabalhar em uma tarefa de cada vez, e uma vez que ĈĈ começa uma atividade/atribuição, ele deve trabalhar até que ela esteja terminada. Assim, o professor ĈĈ quer saber quando ele deve começar a fazer as suas atividades, afim de assegurar que todo o seu os prazos sejam cumpridos?

Entrada

O arquivo de entrada irá conter vários casos de teste. Cada caso de teste consiste de três linhas. A primeira linha de cada caso de teste contém um inteiro N ($1 \leq n \leq 1000$). A segunda linha de cada caso de teste contém N inteiros, $x_1\ x_2\ x_3\ ...\ x_n$ (duração de cada atividade i) ($1 \leq x_i \leq 10$) separados por espaços. A terceira linha de cada caso de teste contém N inteiros, $t_1\ t_2\ t_3\ ...\ t_n$ (horário de término ou *deadline* de cada atividade i) ($1 \leq t_i \leq 1000$) separados por simples espaços.

Uma linha em branco separa os casos de teste de entrada, como visto na amostra de entrada abaixo. A única linha que contém `0` marca o fim da entrada; e não é para processar este caso.

Saída

Para cada caso de teste de entrada, deve-se imprimir uma única linha contendo um inteiro indicando o último instante que o professor ĈĈ deve começar suas tarefas, e ainda assim, conseguir terminar todas x_i atividades no seus *deadlines* ou atribuições no tempo t_i. Se o último tempo necessário exigir que o professor ĈĈ inicie antes tempo "`0`", imprimir "`impossible`".

Exemplo de Entrada

```
3
1 2 1
9 9 7

2
2 2
3 3

0
```

Exemplo de Saída

```
5
impossible
```

5 Problema E: Earthquake Emendations

`Arquivo: earthquake.[c|cpp|java]`

A major earthquake has turned a beautiful stained glass window at the Farm Hill College chapel into shards of glass lying on the floor. The chapel manager is calling upon you to help him quickly put the window back together. Luckily the window only broke along its lead joints, and the individual colored pieces are all intact. Additionally, as if by divine intervention, all of the pieces landed on the floor either in their original orientation, or rotated by a multiple of 90 degrees ($\frac{\pi}{2}$ radians), and no pieces have been flipped on their face.

After much searching, a schematic of the original stained glass window as it stood before the earthquake was found in the library. You learn from the schematic that no two colored pieces of the window are identical in shape and size. With this schematic in hand, your mission is to write a program that will identify the strewn shards to help reassemble the window.

Input

Your program will be given several test cases, each comprising a description of a broken window for you to reassemble. Every test case begins with a single line containing an integer n ($1 \leq n \leq 20$), the number of individual glass pieces in the window. The next n lines each contain a description of a unique colored glass piece in the form of a simple polygon with nonzero area. The coordinates of the k ($3 \leq k \leq 100$) vertices of each polygon will be given in counter-clockwise order in the following format: $x_1\ y_1\ x_2\ y_2\ ...\ x_k\ y_k\ x_1\ y_1$. You may assume that the vertices of each polygon are distinct, i.e., $(x_i, y_i) \neq (x_j, y_j)$ whenever $i \neq j$, and consecutive vertices are not collinear. All coordinates are integers restricted to the range $0 \leq x_i\ ,\ y_i \leq 100$.

The final line of each test case is the schematic of the original window, written as a concatenation of n non-overlapping polygons in the same format described above. Each of the polygons in the schematic is guaranteed to correspond to some rotation plus translation of one of the glass pieces described above. Test cases will be separated by blank lines, and the final line of the input will contain a single integer "0", marking the end of input. For example,

```
2
1 3 4 3 1 5 1 3
4 4 4 7 2 7 4 4
0 0 2 0 2 3 0 0 2 0 4 0 2 3 2 0

4
0 0 4 0 4 2 2 2 0 0
0 0 4 0 2 2 0 2 0 0
4 4 0 4 2 3 4 4
0 4 4 4 2 6 0 4
0 0 4 0 2 2 0 0 0 0 2 2 2 4 0 4 0 0 2 2 4 0 4 4 2 4 2 2 0 4 4 4 2 5 0 4

0
```

Output

Your program should produce a single line of output for each test case. For every glass piece in the input for a test case, write a single integer indicating the position within the schematic that the piece appears. Separate the numbers with a single space in the output. For example:

```
1 2
3 2 4 1
```

6 Problema F: Estruturas em Colapso

`Arquivo: estruturas.[c|cpp|java]`

Você pode conhecer um brinquedo popular de tipo *lego* de hastes rígidas e esferas metálicas destinado a construção de figuras geométricas diversas. Este brinquedo é cobiçado por Maria que terá seu aniversário nesta semana. Este brinquedo é constituído por um número de esferas de metal que podem ser ligadas umas as outras usando elos ou hastes rígidas (todas do mesmo comprimento) com um ímã em cada extremidade. As extremidades destas hastes possuem estas esferas de metal. Estas hastes pode rodar livremente e estender desde uma esfera em qualquer direcção (essencialmente formando uma junta esférica), permitindo que você crie uma variedade de estruturas interessantes.

Maria juntou várias dessas estruturas, e agora deseja guardar suas criações, pendurando-as num canto do teto de seu quarto. Ela percebe que, para algumas de suas estruturas, quando ela pega por uma das esferas e tenta prendê-la, suspendendo-as, todas as hastes se colapsam (fundem) em uma única linha fina vertical (ver Figura 3) devido à força da gravidade e todas esferas se juntam. Maria pode pendurar suas criações, prendendo uma esfera por estrutura no teto de seu quarto, visando economizar espaço pelo colapso ou fusão de toda estrutura, pela esfera que resulta na linha mais curta entre a esfera pendurada e uma das extremidades. Ela considera que as estruturas (suas criações) que não penduram como uma única linha fina, pois estas ocupam muito espaço, e então ela descarta-as.

Figura 3: Uma das construções de Maria (à esquerda) entrou em *colapso* ou fusão, resultando em uma linha reta de comprimento 2, enquanto que a figura da direita não irá entrar em colapso quando suspensa por qualquer uma das suas esferas. No diagrama do meio, as lacunas horizontais mostradas entre as esferas são apenas para fins ilustrativos! Matematicamente, a estrutura seria uma única linha, infinitamente fina na vertical.

Para simplificar, podemos tratar as esferas de metal como pontos infinitamente pequenos e as hastes como linhas de segmentos com comprimento de uma unidade. Você pode escrever um programa para ajudar Maria descobrir quanto de espaço suas estruturas irão ocupar, e quais devem ser descartadas? Sua tarefa é encontrar o menor comprimento possível para a fixação de uma estrutura em colapso, se você fosse para pendurar até uma única esfera no teto. Ou ainda, indique que não há maneira de juntar a estrutura em uma única linha reta infinitamente fina.

Entrada

A entrada irá conter vários casos de teste para você analisar. Cada caso de teste descreve uma estrutura completamente conectada (isto é, não existem componentes soltos, esferas e

hastes estão acopladas). A primeira linha de um caso de teste consiste em dois números inteiros, n e m, separados por um espaço, indicando o número de esferas ($1 \leq n \leq 100$) e de hastes ($0 \leq m \leq 1000$) usadas na estrutura, respectivamente. As esferas são numeradas exclusivamente a partir de 1 até n. As m linhas seguintes da entrada contém dois números inteiros, a_i e b_i ($1 \leq a_i, b_i \leq n$), indicando que Maria fixou uma esfera a a uma esfera b em sua estrutura. Observe que ambas as extremidades de uma ligação ou haste, não podem ser ligadas a uma única esfera, e nem mesmo duas hastes se conectarão a duas esferas iguais.

Uma linha em branco separa cada caso de teste de entrada, como visto na amostra de entrada. Uma única linha contendo "`0 0`" marca o fim da entrada, e não se deve processar este caso.

Saída

Para cada caso de teste de entrada, imprima uma única linha que contenha o menor comprimento possível da estrutura em colapso. Afinal, Maria não quer enormes estruturas penduradas no teto de seu quarto, o comprimento das menores estruturas é o que interessa. Se não for possível para pendurar a estrutura, conforme descrito acima, por uma única esfera, formando uma única linha, escreva em uma linha contendo: "`impossible`". Veja os exemplos de entrada e saída a seguir:

Exemplo de Entrada

```
6 6
1 2
5 6
3 2
3 5
4 2
4 5

4 5
1 2
2 3
3 4
1 3
2 4

0 0
```

Exemplo de Saída

```
2
impossible
```

7 Problema G: Balas de Goma Esféricas

`Arquivo: gomas.[c|cpp|java]`

Glória comprou um saco de balas de goma. No entanto, ela não quer carregá-las num saco plástico. Em vez disso, ela as coloca num tubo cilíndrico de diâmetro d. Supondo que cada uma das balas é uma esfera perfeita de raios r_1, r_2 $\ldots, r_n$, determine o menor comprimento do tubo que Glória deve usar para guardar suas balas (Fig. 4).

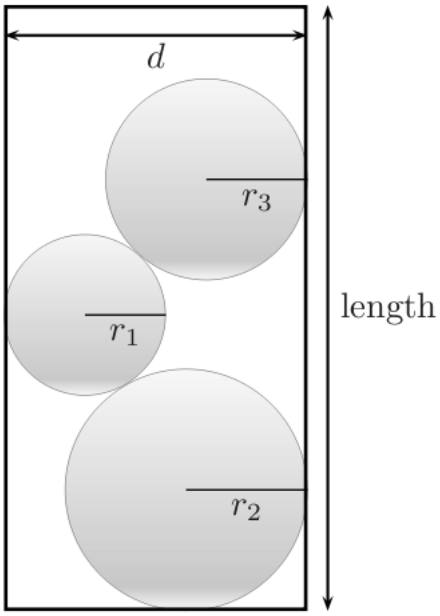

Figura 4: Tubo com balas de goma

Você deve supor que os raios das balas são suficientemente grandes para nunca três balas possam estar em contato simultaneamente uma com a outra dentro do tubo. Dada tal restrição, é útil levar em conta o fato as balas ficarão dispostas no tubo de tal modo que seus centros devem ficar sobre um plano 2-dimensional contendo o eixo de rotação do tubo.

Entrada

O arquivo de entrada consiste de múltiplos casos-teste. Cada caso-teste consiste de duas linhas. A primeira linha contém um inteiro n ($1 \leq n \leq 15$), indicando o número de de balas que Glória tem, e um valor em ponto flutuante d ($2.0 \leq d \leq 1000.0$) indicando o diâmetro do tubo cilíndrico, separados por um espaço. A segunda linha contém uma sequência de n pontos (separados por espaços) r_1, r_2, $\ldots$ r_n ($1.0 \leq r_i \leq d/2$), que denotam os raios das balas de goma. Uma linha em branco separa os casos-teste de entrada. Uma única linha com números "`0 0`" marca o fim da entrada; não processe este caso.

Saída

Para cada caso-teste de entrada, imprima o comprimento do menor tubo, arrendondado para o inteiro mais próximo.

Exemplo de Entrada

```
2 98.1789
42.8602 28.7622

3 747.702
339.687 191.953 330.811

0 0
```

Exemplo de Saída

```
138
1628
```

8 Problema H: Honed Hops

`Arquivo: hops.[c|cpp|java]`

In the Olympics, appearances do matter!

The trajectory of a long jumper is given by $h(x) = max(0, p(x))$, where $p(x) = a(x-h)^2+k$ is a quadratic polynomial describing a parabola opening downward whose vertex (h, k) lies in the upper half-plane. (That is, $a < 0$ and $k > 0$.)

Due to rigorous training, each jumper always jumps with the same trajectory, and due to corporate sponsorship and branding requirements, no two jumpers have the same trajectory.

Adoring fans who wish to preserve the moment occasionally sample their favorite athlete's coordinates at various times and write them down, such as: $(0, 0)$, $(1, 3)$, $(2, 4)$, $(3, 3)$, $(4, 0)$, $(7, 0)$.

Given two sample sets, your job is to determine whether they were taken from the same athlete or not, assuming there is enough information to do so.

Input

The input test file will contain multiple cases, each separated by a blank line. Each test case consists of three lines of text. The first line contains two integers, n_1 and n_2 ($1 \leq n_1, n_2 \leq 10$) separated by a space, indicating the number of sample points for the first and second sample sets, respectively. The second and third lines contain the sample points for the two sets, in the format x_1 y_1 x_2 y_2 .... x_n y_n. You may assume that $x_1 < x_2 < .... < x_n$; moreover, $0 \leq x_i \leq 100$, and $\leq y_i \leq 1000$ for each i. (Be careful that your calculations have sufficient precision for all input conforming to the stated bounds.)

Input is terminated by a single line containing "`0 0`"; do not process this case.

Output

For each test case, your program should output a single line containing "`same`" if the two sample sets are indeed from the same athlete, "`different`" if they are not, and "`unsure`" if there is not enough information to tell.

Sample Input

```
6 4
0 0 1 3 2 4 3 3 4 0 7 0
1 3 2 4 3 3 4 0

6 1
0 0 1 3 2 4 3 3 4 0 7 0
0 0

6 2
0 0 1 3 2 4 3 3 4 0 7 0
1 3 2 5

0 0
```

Sample Output

```
same
unsure
different
```

9 Problema I: Falta Um

`Arquivo: faltaum.[c|cpp|java]`

Um estagiário recebeu a tarefa de gerar sequencialmente todos os números pertencentes à faixa de a até b (inclusive), porém, devido a um erro no código, todas as sequencias que foram geradas tiveram exatamente um número faltante! Sua tarefa é determinar qual é o número faltante.

Tarefa

Seu programa deverá tratar vários casos de teste.

A primeira linha de cada caso de teste é composta por dois números, a e b, referentes ao valor inicial e final da faixa (inclusive) da sequencia a ser verificada (o final da entrada é marcado por `a = b = 0`).

A segunda linha de cada caso será composta pelos valores da faixa, apresentados em ordem crescente e separados por um espaço.

Limites

$0 \leq a \leq b \leq 10.000$ (0 utilizado somente para marcar o final).

Exemplo de Entrada

```
1 4
1 2 4
5 9
6 7 8 9
0 0
```

Exemplo de Saída

```
3
5
```

10 Problema J: Vaca e Frango

`Arquivo: vacafrango.[c|cpp|java]`

O canal CN (Cultura Nacional) está desenvolvendo um novo desenho, baseado em vacas, frangos (com e sem osso), doninhas e babuínos. Você, sendo o estagiário mais apto à tarefa, foi escolhido para desenvolver um programa para verificar a ordem da canção de abertura.

Tarefa

Seu programa deverá tratar vários casos de teste. Cada caso de teste é composto por dois números, a e b, referentes ao segundo inicial e final da faixa (inclusive) da canção que você deve verificar. O final da entrada é marcado por `a = b = 0`.

A cada segundo s você deve calcular a frase correta emitida pelo cantor, seguindo a seguinte regra:

1. Se s for divisível por 3, imprima "`vaca!`" e pule a linha;
2. Se s for divisível por 5, imprima "`frango!`" e pule a linha;
3. Se s for divisível por 3 e 5 simultaneamente, imprima "`vaca e frango!`" (ignore os itens 1 e 2) e pule a linha;
4. Se nenhum dos itens acima tiverem suas condições atendidas, imprima o valor de s e pule a linha;

Limites

$0 \leq a \leq b \leq 100$ (0 utilizado somente para marcar o final).

Exemplo de Entrada

```
1 6
15 16
0 0
```

Exemplo de Saída

```
1
2
vaca!
4
frango!
vaca!
vaca e frango!
16
```