acm International Collegiate Programming Contest 2008 IBM event sponsor

Maratona de Programação da SBC 2008

Sub-Regional Brasil do ACM ICPC

20 de Setembro de 2008

Caderno de Problemas

(Este caderno contém 10 problemas; as páginas estão numeradas de 1 a 17, não contando esta página de rosto)

Promoção:

Sociedade Brasileira de Computação

Patrocínio:

Fundação Carlos Chagas

v5.0

Problema A
Apagando e Ganhando

Nome do arquivo fonte: `apagando.c`, `apagando.cpp` *ou* `apagando.java`

Juliano é fã do programa de auditório *Apagando e Ganhando*, um programa no qual os participantes são selecionados atráves de um sorteio e recebem prêmios em dinheiro por participarem.

No programa, o apresentador escreve um número de N dígitos em uma lousa. O participante então deve apagar exatamente D dígitos do número que está na lousa; o número formado pelos dígitos que restaram é então o prêmio do participante.

Juliano finalmente foi selecionado para participar do programa, e pediu que você escrevesse um programa que, dados o número que o apresentador escreveu na lousa, e quantos dígitos Juliano tem que apagar, determina o valor do maior prêmio que Juliano pode ganhar.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois inteiros N e D ($1 \leq D < N \leq 10^5$), indicando a quantidade de dígitos do número que o apresentador escreveu na lousa e quantos dígitos devem ser apagados. A linha seguinte contém o número escrito pelo apresentador, que não contém zeros à esquerda.

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 cada caso de teste da entrada seu programa deve imprimir uma única linha na saída, contendo o maior prêmio que Juliano pode ganhar.

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

Exemplo de entrada	**Exemplo de saída**
4 2 3759 6 3 123123 7 4 1000000 0 0	79 323 100

Problema B
Babel

Nome do arquivo fonte: `babel.c`, `babel.cpp` *ou* `babel.java`

Joãozinho e Mariazinha são dois irmãos que estão muito empolgados com suas aulas de idiomas, cada um está fazendo vários diferentes cursinhos. Ao chegar em casa comentam sobre gramática, vocabulário, cultura dos países etc. Numa dessas conversas perceberam que algumas palavras são comuns a mais de um idioma, mesmo que não necessariamente tenham o mesmo significado. Por exemplo, "amigo" existe em português e espanhol e tem o mesmo significado, enquanto que "date" é uma palavra comum entre francês e inglês mas que pode ter significados diferentes, uma vez que "date" também se refere a um encontro em inglês, além de "data" de calendário. Já "red" em espanhol se refere a uma rede, enquanto que em inglês se refere à cor vermelha. Outro exemplo seria "actual" que, em inglês significa algo real e, em espanhol, tem o significado de presente, atual (como em português).

Empolgados com essas descobertas, resolveram escrever num caderno todas as palavras em comum que conseguiram pensar, associando cada uma a um par de idiomas. Observador como é, Joãozinho propôs um desafio a Mariazinha: dados um idioma de origem e um de destino, escrever uma série de palavras sendo que a primeira necessariamente deveria pertencer ao idioma de origem e a última ao de destino. Duas palavras adjacentes nessa seqüência deveriam necessariamente pertencer a um mesmo idioma. Por exemplo, se o idioma de origem fosse português e o de destino francês, Mariazinha poderia escrever a seqüência `amigo actual date` (português/espanhol, espanhol/inglês, inglês/francês).

Para a surpresa de Joãozinho, Mariazinha conseguiu resolver o problema com muita facilidade. Irritado com o sucesso de sua irmã, ele resolveu complicar ainda mais o problema com duas restrições: Mariazinha deve encontrar a solução que tenha o menor comprimento da seqüência total não contando os espaços entre as palavras e duas palavras consecutivas não podem ter a mesma letra inicial.

Sendo assim, a solução anterior passa a ser inválida, pois "amigo" e "actual" têm a mesma letra inicial. É possível, porém, encontrar outra solução, que no caso seria `amigo red date`, cujo comprimento total é 12.

Joãozinho fez uma extensa pesquisa na internet e compilou uma enorme lista de palavras e desafiou Mariazinha a resolver o problema. Como é possível que haja mais de uma solução, ele pediu para que ela apenas respondesse o comprimento da seqüência encontrada dadas as restrições ou se não há solução possível. Você seria capaz de ajudar Mariazinha?

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro M ($1 \leq M \leq 2000$), representando o total de palavras compiladas por Joãozinho. A segunda linha contém duas cadeias de caracteres distintas O e D, separadas por um espaço em branco, indicando os idiomas de origem e destino respectivamente. Cada uma das M linhas seguintes contém três cadeias de caracteres I_1, I_2 e P, separadas por um espaço em branco, representando dois idiomas e uma palavra comum entre ambos (I_1 e I_2 são sempre diferentes). Todas as cadeias de caracteres terão tamanho mínimo 1 e máximo 50 e conterão apenas letras minúsculas. Um mesmo par de idiomas pode ter várias palavras diferentes associadas a ele, porém uma mesma palavra P nunca será repetida.

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

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir um único inteiro, o comprimento da menor seqüência que satisfaça as restrições de Joãozinho, ou `impossivel` (em minúsculas, sem acento) caso não seja possível.

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

Exemplo de entrada	**Exemplo de saída**
4 portugues frances ingles espanhol red espanhol portugues amigo frances ingles date espanhol ingles actual 4 portugues alemao ingles espanhol red espanhol portugues amigo frances ingles date espanhol ingles actual 6 portugues frances ingles espanhol red espanhol portugues amigo frances ingles date frances espanhol la portugues ingles a espanhol ingles actual 0	12 impossivel 5

Problema C
O Salão do Clube

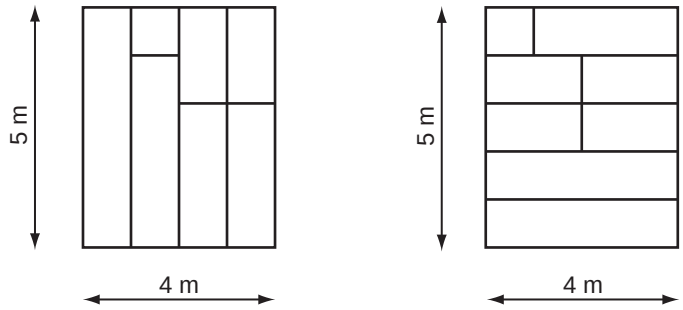
Nome do arquivo fonte: `clube.c`, `clube.cpp` *ou* `clube.java`

O Clube Recreativo de Tinguá está construindo a sua nova sede social. Os sócios desejam que o piso do salão da sede seja de tábuas de madeira, pois consideram que este é o melhor tipo de piso para os famosos bailes do clube. Uma madeireira da região doou uma grande quantidade de tábuas de boa qualidade, para serem utilizadas no piso. As tábuas doadas têm todas a mesma largura, mas têm comprimentos distintos.

O piso do salão da sede social é retangular. As tábuas devem ser colocadas justapostas, sem que qualquer parte de uma tábua seja sobreposta a outra tábua, e devem cobrir todo o piso do salão. Elas devem ser dispostas alinhadas, no sentido longitudinal, e todas devem estar no mesmo sentido (ou seja, todas as tábuas devem estar paralelas, no sentido longitudinal). Além disso, os sócios não querem muitas emendas no piso, e portanto se uma tábua não é longa o bastante para cobrir a distância entre um lado e outro do salão, ela pode ser emendada a no máximo uma outra tábua para completar a distância.

Há porém uma complicação adicional. O carpinteiro-chefe tem um grande respeito por todas as madeiras e prefere não serrar qualquer tábua. Assim, ele deseja saber se é possível forrar todo o piso com as tábuas doadas, obedecendo às restrições especificadas; caso seja possível, o carpinteiro-chefe deseja ainda saber o menor número de tábuas que será necessário utilizar.

A figura abaixo ilustra duas possíveis maneiras de forrar o piso de um salão com dimensões 4×5 metros para um conjunto de dez tábuas doadas, com 100 cm de largura, e comprimentos 1, 2, 2, 2, 2, 3, 3, 4, 4 e 5 metros.



Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros M e N indicando as dimensões, em metros, do salão ($1 \leq M, N \leq 10^4$). A segunda linha contém um inteiro L, representando a largura das tábuas, em centímetros ($1 \leq L \leq 100$). A terceira linha contém um inteiro, K, indicando o número de tábuas doadas ($1 \leq K \leq 10^5$). A quarta linha contém K inteiros X_i, separados por um espaço, cada um representando o comprimento, em metros, de uma tábua ($1 \leq X_i \leq 10^4$ para $1 \leq i \leq K$).

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 cada um dos casos de teste da entrada seu programa deve imprimir uma única linha, contendo o menor número de tábuas necessário para cobrir todo o piso do salão, obedecendo às restrições estabelecidas. Caso não seja possível cobrir todo o piso do salão obedecendo às restrições estabelecidas, imprima uma linha com a palavra '`impossivel`' (letras minúsculas, sem acento).

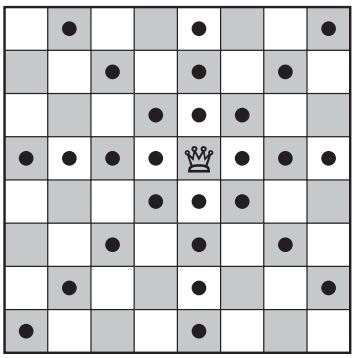
O resultado de seu programa deve ser escrito na saída padrão.

Exemplo de entrada	**Exemplo de saída**
4 5 100 10 1 2 2 2 2 3 3 4 4 5 5 4 100 7 4 5 4 4 4 4 3 4 5 99 4 4 4 4 4 3 2 100 7 2 4 1 4 2 4 4 0 0	7 5 impossivel impossivel

Problema D
Dama

Nome do arquivo fonte: `dama.c`, `dama.cpp` *ou* `dama.java`

O jogo de xadrez possui várias peças com movimentos curiosos: uma delas é a *dama*, que pode se mover qualquer quantidade de casas na mesma linha, na mesma coluna, ou em uma das duas diagonais, conforme exemplifica a figura abaixo:



O grande mestre de xadrez Kary Gasparov inventou um novo tipo de problema de xadrez: dada a posição de uma dama em um tabuleiro de xadrez vazio (ou seja, um tabuleiro 8×8, com 64 casas), de quantos movimentos, no mínimo, ela precisa para chegar em outra casa do tabuleiro?

Kary achou a solução para alguns desses problemas, mas teve dificuldade com outros, e por isso pediu que você escrevesse um programa que resolve esse tipo de problema.

Entrada

A entrada contém vários casos de teste. A primeira e única linha de cada caso de teste contém quatro inteiros X_1, Y_1, X_2 e Y_2 ($1 \leq X_1, Y_1, X_2, Y_2 \leq 8$). A dama começa na casa de coordenadas (X_1, Y_1), e a casa de destino é a casa de coordenadas (X_2, Y_2). No tabuleiro, as colunas são numeradas da esquerda para a direita de 1 a 8 e as linhas de cima para baixo também de 1 a 8. As coordenadas de uma casa na linha X e coluna Y são (X, Y).

O final da entrada é indicado por uma linha contendo quatro zeros.

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha na saída, contendo um número inteiro, indicando o menor número de movimentos necessários para a dama chegar em sua casa de destino.

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

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

Problema E
Bolhas e Baldes

Nome do arquivo fonte: `bolhas.c`, `bolhas.cpp` *ou* `bolhas.java`

Andrea, Carlos e Marcelo são muito amigos e passam todos os finais de semana à beira da piscina. Enquanto Andrea se bronzeia ao sol, os dois ficam jogando *Bolhas*. Andrea, uma cientista da computação muito esperta, já disse a eles que não entende por que passam tanto tempo jogando um jogo tão primário.

Usando o computador portátil dela, os dois geram um inteiro aleatório N e uma seqüência de inteiros, também aleatória, que é uma permutação de $1, 2, \ldots, N$.

O jogo então começa, cada jogador faz um movimento, e a jogada passa para o outro jogador. Marcelo é sempre o primeiro a começar a jogar.

Um movimento de um jogador consiste na escolha de um par de elementos consecutivos da seqüência que estejam fora de ordem e em inverter a ordem dos dois elementos. Por exemplo, dada a seqüência $1, 5, 3, 4, 2$, o jogador pode inverter as posições de 5 e 3 ou de 4 e 2, mas não pode inverter as posições de 3 e 4, nem de 5 e 2. Continuando com o exemplo, se o jogador decide inverter as posições de 5 e 3 então a nova seqüência será $1, 3, 5, 4, 2$.

Mais cedo ou mais tarde, a seqüência ficará ordenada. Perde o jogador impossibilitado de fazer um movimento.

Andrea, com algum desdém, sempre diz que seria mais simples jogar cara ou coroa, com o mesmo efeito. Sua missão, caso decida aceitá-la, é determinar quem ganha o jogo, dada a seqüência inicial.

Entrada

A entrada contém vários casos de teste. Os dados de cada caso de teste estão numa única linha, e são inteiros separados por um espaço em branco. Cada linha contém um inteiro N, $2 \leq N \leq 10^5$, seguido da seqüência inicial $P = (X_1, X_2, \ldots, X_N)$ de N inteiros distintos dois a dois, onde $1 \leq X_i \leq N$ para $1 \leq i \leq N$. O final da entrada é indicado por uma linha que contém apenas o número zero.

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha, com o nome do vencedor, igual a `Carlos` ou a `Marcelo`, sem espaços em branco.

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

Exemplo de entrada	**Exemplo de saída**

`5 1 5 3 4 2`
`5 5 1 3 4 2`
`5 1 2 3 4 5`
`6 3 5 2 1 4 6`
`5 5 4 3 2 1`
`6 6 5 4 3 2 1`
`0`
	`Marcelo`
	`Carlos`
	`Carlos`
	`Carlos`
	`Carlos`
	`Marcelo`

Problema F
Loop Musical

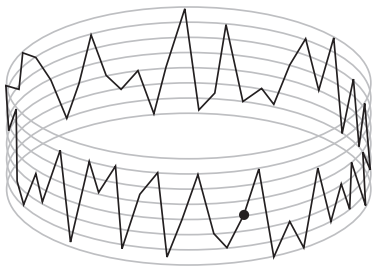
Nome do arquivo fonte: `loop.c`, `loop.cpp` *ou* `loop.java`

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

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

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

(a) Uma forma de onda



(b) Mesma forma de onda em loop

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

Entrada

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

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

Os dados devem ser lidos da entrada padrão.

Saída

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

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

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

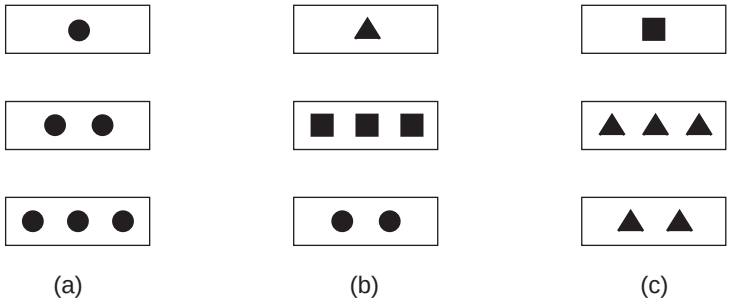
Problema G
Set

Nome do arquivo fonte: `set.c`, `set.cpp` *ou* `set.java`

Set é um jogo jogado com um baralho no qual cada carta pode ter uma, duas ou três figuras. Todas as figuras em uma carta são iguais, e podem ser círculos, quadrados ou triângulos.

Um *set* é um conjunto de três cartas em que, para cada característica (número e figura), ou as três cartas são iguais, ou as três cartas são diferentes. Por exemplo, na figura abaixo, (a) é um set válido, já que todas as cartas têm o mesmo tipo de figura e todas elas têm números diferentes de figuras.

Em (b), tanto as figuras quanto os números são diferentes para cada carta. Por outro lado, (c) não é um set, já que as duas últimas cartas têm a mesma figura, mas esta é diferente da figura da primeira carta.



O objetivo do jogo é formar o maior número de sets com as cartas que estão na mesa; cada vez que um set é formado, as três cartas correspondentes são removidas de jogo.

Quando há poucas cartas na mesa, é fácil determinar o maior número de sets que podem ser formados; no entanto, quando há muitas cartas há muitas combinações possíveis. Seu colega quer treinar para o campeonato mundial de Set, e por isso pediu que você fizesse um programa que calcula o maior número de sets que podem ser formados com um determinado conjunto de cartas.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N ($3 \leq N \leq 3 \times 10^4$), indicando o número de cartas na mesa; cada uma das N linhas seguintes contém a descrição de uma carta.

A descrição de uma carta é dada por duas palavras separadas por um espaço; a primeira, "`um`", "`dois`" ou "`tres`", indica quantas figuras aquela carta possui. A segunda, "`circulo`" (ou "`circulos`"), "`quadrado`" (ou "`quadrados`") ou "`triangulo`" (ou "`triangulos`") indica qual tipo de figura está naquela carta.

O final da entrada é indicado por uma linha contendo um zero.

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha na saída, contendo um número inteiro, indicando o maior número de sets que podem ser formados com as cartas dadas.

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

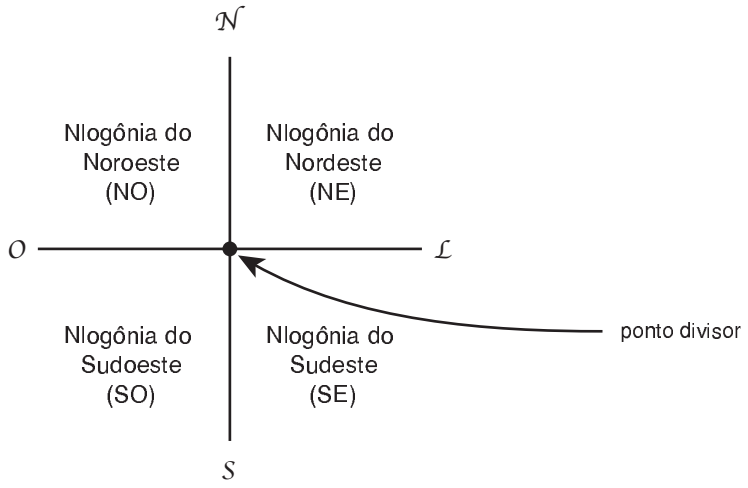
Exemplo de entrada	**Exemplo de saída**
3 um circulo dois circulos tres circulos 3 um triangulo tres quadrados dois circulos 6 dois quadrados dois quadrados dois quadrados dois quadrados dois quadrados dois quadrados 4 um quadrado tres triangulos um quadrado dois triangulos 0	1 1 2 0

Problema H
Divisão da Nlogônia

Nome do arquivo fonte: `nlog.c`, `nlog.cpp` *ou* `nlog.java`

Depois de séculos de escaramuças entre os quatro povos habitantes da Nlogônia, e de dezenas de anos de negociações envolvendo diplomatas, políticos e as forças armadas de todas as partes interessadas, com a intermediação da ONU, OTAN, G7 e SBC, foi finalmente decidida e aceita por todos a maneira de dividir o país em quatro territórios independentes.

Ficou decidido que um ponto, denominado *ponto divisor*, cujas coordenadas foram estabelecidas nas negociações, definiria a divisão do país, da seguinte maneira. Duas linhas, ambas contendo o ponto divisor, uma na direção norte-sul e uma na direção leste-oeste, seriam traçadas no mapa, dividindo o país em quatro novos países. Iniciando no quadrante mais ao norte e mais ao oeste, em sentido horário, os novos países seriam chamados de Nlogônia do Noroeste, Nlogônia do Nordeste, Nlogônia do Sudeste e Nlogônia do Sudoeste.



A ONU determinou que fosse disponibilizada uma página na Internet para que os habitantes pudessem consultar em qual dos novos países suas residências estão, e você foi contratado para ajudar a implementar o sistema.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro K indicando o número de consultas que serão realizadas ($0 < K \leq 10^3$). A segunda linha de um caso de teste contém dois números inteiros N e M representando as coordenadas do ponto divisor ($-10^4 < N, M < 10^4$). Cada uma das K linhas seguintes contém dois inteiros X e Y representando as coordenadas de uma residência ($-10^4 \leq X, Y \leq 10^4$). Em todas as coordenadas dadas, o primeiro valor corresponde à direção leste-oeste, e o segundo valor corresponde à direção norte-sul.

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

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma linha contendo:

- a palavra `divisa` se a residência encontra-se em cima de uma das linhas divisórias (norte-sul ou leste-oeste);
- `NO` se a residência encontra-se na Nlogônia do Noroeste;
- `NE` se a residência encontra-se na Nlogônia do Nordeste;
- `SE` se a residência encontra-se na Nlogônia do Sudeste;
- `SO` se a residência encontra-se na Nlogônia do Sudoeste.

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

Exemplo de entrada	**Exemplo de saída**
3 2 1 10 10 -10 1 0 33 4 -1000 -1000 -1000 -1000 0 0 -2000 -10000 -999 -1001 0	NE divisa NO divisa NE SO SE

Problema I
Maior Subseqüência Crescente

Nome do arquivo fonte: `seq.c`, `seq.cpp` *ou* `seq.java`

Determinar a subseqüência (contígua) crescente de maior comprimento em uma lista de números é um problema já clássico em competições de programação. Este é o problema que você deve resolver aqui, mas para não deixar você bocejando de tédio enquanto o soluciona, introduzimos uma pequena modificação: a lista de números é dada na forma de uma matriz bidimensional e a seqüência de comprimento máximo está "embutida" em uma submatriz da matriz original.

Vamos definir mais precisamente o problema. A linearização de uma matriz bidimensional é a justaposição de suas linhas, da primeira à última. Uma submatriz é uma região retangular (de lados paralelos aos da matriz) de uma matriz. O tamanho de uma submatriz é seu número de elementos. Você deve escrever um programa que, dada uma matriz de números inteiros, determine a maior submatriz que, quando linearizada, resulta em uma seqüência crescente.

A figura abaixo mostra alguns exemplos de submatrizes de tamanho máximo que contêm subseqüências crescentes. Note que mais de uma submatriz que contém uma subseqüência de comprimento máximo pode estar presente em uma mesma matriz. Note ainda que numa seqüência crescente não pode haver elementos repetidos: 22, 31, 33 é uma seqüência crescente, ao passo que 22, 31, 31, 33 não é.

1	**2**	**5**
4	**6**	**7**
10	8	3

1	**2**	1	2
9	**6**	7	3
8	**7**	2	8

22	2	14	22	23
16	**21**	**22**	**31**	31
57	**33**	**43**	**45**	50
46	**51**	**66**	**83**	93

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros N e M indicando as dimensões da matriz ($1 \leq N, M \leq 600$). Cada uma das N linhas seguintes contém M inteiros, separados por um espaço, descrevendo os elementos da matriz. O elemento $X_{i,j}$ da matriz é o j-ésimo inteiro da i-ésima linha da entrada ($-10^6 \leq X_{i,j} \leq 10^6$).

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 cada um dos casos de teste da entrada seu programa deve imprimir uma única linha, contendo o número de elementos da maior submatriz que, quando linearizada, resulta em uma seqüência crescente.

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

Exemplo de entrada	**Exemplo de saída**
3 3 1 2 5 4 6 7 10 8 3 3 4 1 2 1 2 9 6 7 3 8 7 2 8 4 2 -23 -12 0 2 16 15 57 33 4 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0	4 3 4 1

Problema J
Vampiros

Nome do arquivo fonte: `vampiros.c`, `vampiros.cpp` *ou* `vampiros.java`

Felipinho está empolgado com seu novo jogo de RPG sobre guerras entre clãs de vampiros. Nesse jogo ele representa um personagem de um vampiro e constantemente entra em conflito contra vampiros de outros clãs. Tais batalhas são realizadas com base nas características de cada personagem envolvido e com a ajuda de um dado comum de seis faces.

Por simplicidade, vamos considerar apenas as lutas entre dois vampiros, vampiro 1 e vampiro 2. Cada um possui uma energia vital (chamaremos de EV_1 e EV_2 respectivamente) e, além disso, são determinadas uma força de ataque AT e uma capacidade de dano D.

O combate é realizado em turnos da maneira descrita a seguir. A cada turno um dado é rolado, se o valor obtido for menor do que ou igual a AT, o vampiro 1 venceu o turno, caso contrário o vampiro 2 é quem venceu. O vencedor suga energia vital do adversário igual ao valor D, ou seja, D pontos de EV são diminuídos do perdedor e acrescentados ao vencedor. O combate segue até que um dos vampiros fique com EV igual a ou menor do que zero.

Por exemplo, suponhamos que $EV_1 = 7$, $EV_2 = 5$, $AT = 2$, $D = 4$. Rola-se o dado e o valor obtido foi 3. Nesse caso, o vampiro 2 venceu o turno e, portanto, 4 pontos de EV são diminuídos do vampiro 1 e acrescentados ao vampiro 2. Sendo assim, os novos valores seriam $EV_1 = 3$ e $EV_2 = 9$. Observe que se no próximo turno o vampiro 2 ganhar novamente, o combate irá terminar.

Os valores de AT e D são constantes durante todo o combate, apenas EV_1 e EV_2 variam.

Apesar de gostar muito do jogo, Felipinho acha que os combates estão muito demorados e e gostaria de conhecer de antemão a probabilidade de vencer, para saber se vale a pena lutar. Assim, ele pediu que você escrevesse um programa que, dados os valores iniciais de EV_1, EV_2, além de AT e D, calculasse a probabilidade de o vampiro 1 vencer o combate.

Entrada

A entrada contém vários casos de teste. Cada caso de teste ocupa uma única linha, com os quatro inteiros EV_1, EV_2, AT e D separados por um espaço em branco ($1 \leq EV_1, EV_2 \leq 10$, $1 \leq AT \leq 5$ e $1 \leq D \leq 10$).

O final da entrada é indicado por uma linha contendo quatro zeros.

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha. A linha deve conter apenas um número real, escrito com precisão de uma casa decimal, representando, em termos de percentagem, a probabilidade de o vampiro 1 vencer o combate.

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

Exemplo de entrada	**Exemplo de saída**
1 1 3 1 1 2 1 1 8 5 3 1 7 5 2 4 0 0 0 0	50.0 3.2 61.5 20.0