acm International Collegiate Programming Contest 2009 IBM event sponsor

Maratona de Programação da SBC 2009

Sub-Regional Brasil do ACM ICPC

19 de Setembro de 2009

Caderno de Problemas

(Este caderno contém 8 problemas; as páginas estão numeradas de 1 a 15, não contando esta página de rosto)

Promoção:

Sociedade Brasileira de Computação

Patrocínio:

Fundação Carlos Chagas

v4.0

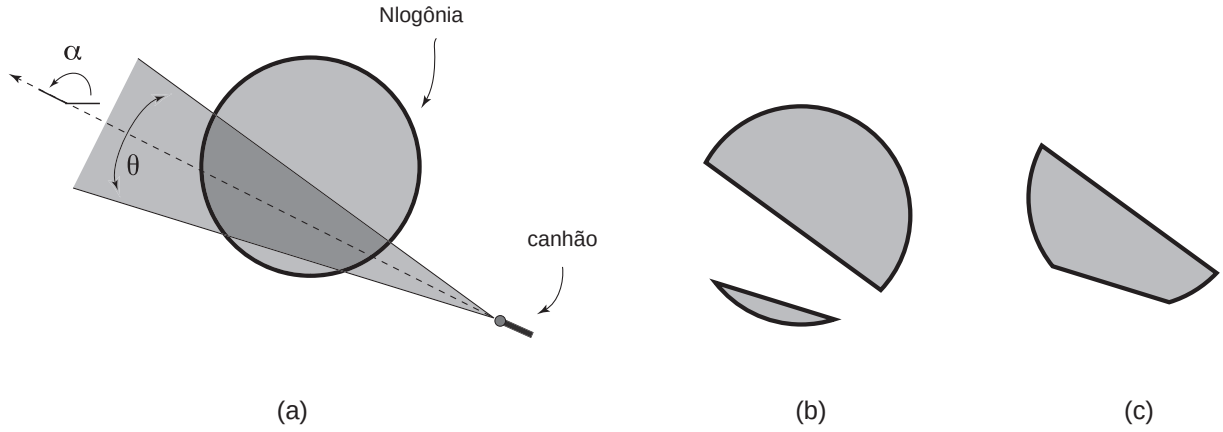
Problema A
Ataque Fulminante

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

Desde que o Rei da Nlogônia construiu, décadas atrás, um enorme muro de proteção ao redor de todo o reino, os seus habitantes vivem em segurança. O muro é imponente, extremamente reforçado, e tem o formato de um círculo que envolve todos os domínios do Rei.

No entanto, há algumas semanas os habitantes da Nlogônia estão apreensivos. Há boatos de que cientistas da Quadradônia, um povo bárbaro que habita as vizinhanças da Nlogônia, desenvolveram uma arma mortal, capaz de pulverizar tudo que esteja em sua mirada.

A nova arma é um canhão que emite um feixe de prótons que se espalha com ângulo θ a partir da boca do canhão. A direção do tiro é indicada por um ângulo α, medido a partir do eixo x, no sentido anti-horário. A figura abaixo ilustra (a) um exemplo de ataque, (b) o que restaria da Nlogônia e (c) a área que seria destruída.



Dados a coordenada do canhão, a direção do tiro e o ângulo de espalhamento do feixe de prótons, bem como a coordenada do centro e o valor do raio do muro de proteção, você deve escrever um programa para calcular a área da Nlogônia que será destruída.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha contém três números inteiros X, Y, R, com (X, Y) representando as coordenadas do centro do círculo do muro de proteção ($0 \leq X \leq 1000$ e $0 \leq Y \leq 1000$), e R o seu raio ($1 \leq R \leq 100$). A segunda linha contém quatro números inteiros P, Q, A e T, com (P, Q) representando as coordenadas da localização do canhão ($0 \leq P \leq 1000$ e $0 \leq Q \leq 1000$), A representando a direção, em graus, do tiro ($0 \leq A \leq 359$), e T representa o ângulo de espalhamento, também em graus ($1 \leq T \leq 179$). O ângulo A é medido a partir do eixo x no sentido anti-horário, e o canhão está sempre fora dos domínios da Nlogônia, ou seja, a distância entre (X, Y) e (P, Q) é maior do que R.

O final da entrada é indicado por uma linha que contém três zeros separados por espaços 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, contendo um número real, escrito com precisão de uma casa decimal, indicando a área da Nlogônia que seria destruída pelo ataque.

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

Exemplo de entrada	**Exemplo de saída**

`1 1 1`
`3 1 180 90`
`4 4 3`
`8 4 90 90`
`4 4 3`
`8 4 180 179`
`0 0 0 0`
	`3.1`
	`0.2`
	`28.3`

Problema B
Alarme Despertador

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

Daniela é enfermeira em um grande hospital, e tem os horários de trabalho muito variáveis. Para piorar, ela tem sono pesado, e uma grande dificuldade para acordar com relógios despertadores.

Recentemente ela ganhou de presente um relógio digital, com alarme com vários tons, e tem esperança que isso resolva o seu problema. No entanto, ela anda muito cansada e quer aproveitar cada momento de descanso. Por isso, carrega seu relógio digital despertador para todos os lugares, e sempre que tem um tempo de descanso procura dormir, programando o alarme despertador para a hora em que tem que acordar. No entanto, com tanta ansiedade para dormir, acaba tendo dificuldades para adormecer e aproveitar o descanso.

Um problema que a tem atormentado na hora de dormir é saber quantos minutos ela teria de sono se adormecesse imediatamente e acordasse somente quando o despertador tocasse. Mas ela realmente não é muito boa com números, e pediu sua ajuda para escrever um programa que, dada a hora corrente e a hora do alarme, determine o número de minutos que ela poderia dormir.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é descrito em uma linha, contendo quatro números inteiros H_1, M_1, H_2 e M_2, com H_1:M_1 representando a hora e minuto atuais, e H_2:M_2 representando a hora e minuto para os quais o alarme despertador foi programado ($0 \leq H_1 \leq 23$, $0 \leq M_1 \leq 59$, $0 \leq H_2 \leq 23$, $0 \leq M_2 \leq 59$).

O final da entrada é indicado por uma linha que contém apenas quatro zeros, separados por espaços em branco.

Os dados devem ser lidos da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma linha, cada uma contendo um número inteiro, indicando o número de minutos que Daniela tem para dormir.

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

Exemplo de entrada	**Exemplo de saída**
1 5 3 5 23 59 0 34 21 33 21 10 0 0 0 0	120 35 1417

Problema C
Troca de Cartas

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

Alice e Beatriz colecionam cartas de Pokémon. As cartas são produzidas para um jogo que reproduz a batalha introduzida em um dos mais bem sucedidos jogos de videogame da história, mas Alice e Beatriz são muito pequenas para jogar, e estão interessadas apenas nas cartas propriamente ditas. Para facilitar, vamos considerar que cada carta possui um identificador único, que é um número inteiro.

Cada uma das duas meninas possui um conjunto de cartas e, como a maioria das garotas de sua idade, gostam de trocar entre si as cartas que têm. Elas obviamente não têm interesse em trocar cartas idênticas, que ambas possuem, e não querem receber cartas repetidas na troca. Além disso, as cartas serão trocadas em uma única operação de troca: Alice dá para Beatriz um sub-conjunto com N cartas distintas e recebe de volta um outro sub-conjunto com N cartas distintas.

As meninas querem saber qual é o número máximo de cartas que podem ser trocadas. Por exemplo, se Alice tem o conjunto de cartas $\{1, 1, 2, 3, 5, 7, 8, 8, 9, 15\}$ e Beatriz o conjunto $\{2, 2, 2, 3, 4, 6, 10, 11, 11\}$, elas podem trocar entre si no máximo quatro cartas.

Escreva um programa que, dados os conjuntos de cartas que Alice e Beatriz possuem, determine o número máximo de cartas que podem ser trocadas.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros A e B, separados por um espaço em branco, indicando respectivamente o número de cartas que Alice e Beatriz possuem ($1 \leq A \leq 10^4$ e $1 \leq B \leq 10^4$). A segunda linha contém A números inteiros X_i, separados entre si por um espaço em branco, cada número indicando uma carta do conjunto de Alice ($1 \leq X_i \leq 10^5$). A terceira linha contém B números inteiros Y_i, separados entre si por um espaço em branco, cada número indicando uma carta do conjunto de Beatriz ($1 \leq Y_i \leq 10^5$). As cartas de Alice e Beatriz são apresentadas em ordem não decrescente.

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, contendo um numero inteiro, indicando o número máximo de cartas que Alice e Beatriz podem trocar entre si.

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

Exemplo de entrada	**Exemplo de saída**
1 1 1000 1000 3 4 1 3 5 2 4 6 8 10 9 1 1 2 3 5 7 8 8 9 15 2 2 2 3 4 6 10 11 11 0 0	0 3 4

Problema D
Sub-prime

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

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

A crise foi tão profunda que acabou atingindo países do mundo inteiro, inclusive a Nlogônia, onde o honrado primeiro ministro Man Dashuva ordenou que o presidente do Banco Central procurasse uma solução para o problema. Esse, por sua vez, teve uma ideia brilhante: se cada banco fosse capaz de liquidar seus empréstimos somente com suas reservas monetárias, todos os bancos sobreviveriam e a crise seria evitada.

Entretanto, com o elevado número de debêntures e bancos envolvidos, essa tarefa é extremamente complicada, e portanto ele pediu a sua ajuda para escrever um programa que, dados os bancos e as debêntures emitidas, determine se é possível que todos os bancos paguem suas dívidas, utilizando suas reservas monetárias e seus créditos.

Entrada

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

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

Os dados devem ser lidos da entrada padrão.

Saída

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

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

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

Problema E
Dragster

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

Embora não seja uma modalidade muito popular no Brasil, as corridas de dragsters atraem multidões nos EUA. Os fãs gostam de ver os carros velozes correndo a velocidades de até 400 km/h, mesmo que só por alguns segundos. Muitos competidores são mecânicos amadores que apenas incluiram foguetes e outros artefatos para criarem carros ultra velozes.

As competições de dragsters são disputadas em torneios de eliminação, onde cada disputa consiste de dois competidores correndo lado a lado e somente um deles sendo declarado o vencedor (o que chegar primeiro, claro). Os vencedores são então rearranjados em novas partidas, até que no final somente um competidor seja declarado o campeão.

Rubens é um piloto experiente, com carreira em diversas categorias, inclusive a Fórmula 1. Entretanto, após enfrentar alguns contratempos, resolveu dedicar-se a competições de dragsters. Aproveitando-se da larga experiência que ganhou durante a Fórmula 1, ele consegue, observando os competidores, dizer qual a probabilidade de cada um dos competidores envolvidos ser o vencedor de uma dada disputa.

Embora Rubens seja bom piloto, não é muito bom em matemática nem em programação, e pediu a sua ajuda para, dadas as probabilidades calculadas por Rubens para a disputa entre cada par de pilotos, e a descrição das corridas do torneio, determinar a probabilidade que ele tem de vencer o torneio.

Entrada

A entrada é composta de vários casos de teste. A primeira linha de um caso de teste contém inteiro N indicando o número de competidores do torneio ($2 \leq N \leq 300$). Na descrição do torneio, os competidores são identificados por inteiros de 1 a N, e as corridas são identificadas por inteiros de $N+1$ a $2 \times N - 1$. Rubens é sempre identificado pelo número 1. As N linhas seguintes descrevem a matriz M de probabilidades calculada por Rubens. A linha i contém N números reais $M[i, j]$ separados por espaços ($0 \leq M[i, j] \leq 1$, para $1 \leq i \leq N$ e $1 \leq j \leq N$). Cada elemento $M[i, j]$ da matriz indica a probabilidade de o competidor i vencer o confronto com o competidor j ($0.001 \leq M[i, j] \leq 0.999$ e $M[i, j] + M[j, i] = 1$ para $i \neq j$, e $M[i, j] = 0$ para $i = j$). As probabilidades serão sempre dadas com três casas decimais de precisão. Cada uma das $N - 1$ linhas seguintes contém dois inteiros A, B descrevendo uma corrida, sendo que A e B representam identificadores de competidores ou de corridas ($1 \leq A \leq 2 \times N - 1$ e $1 \leq B \leq 2 \times N - 1$). Note que a primeira dessas linhas descreve a corrida identificada por $N+1$, a segunda linha descreve a corrida identificada por $N+2$ e assim por diante. Quando um identificador de corrida k aparece na entrada como A, isto significa que o competidor que venceu a corrida k é quem disputará a corrida contra B. Da mesma forma, quando um identificador de corrida k aparece na entrada como B, isto significa que o competidor que venceu a corrida k é quem disputará a corrida contra A.

O final da entrada é indicado por uma linha que contém apenas um 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 um número real, escrito com precisão de seis casas decimais, indicando a probabilidade de Rubens

vencer o torneio.

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

Exemplo de entrada	**Exemplo de saída**
4 0.000 0.500 0.400 0.400 0.500 0.000 0.500 0.500 0.600 0.500 0.000 0.600 0.600 0.500 0.400 0.000 1 2 3 4 5 6 5 0.000 0.500 0.600 0.600 0.001 0.500 0.000 0.500 0.500 0.500 0.400 0.500 0.000 0.500 0.500 0.400 0.500 0.500 0.000 0.500 0.999 0.500 0.500 0.500 0.000 3 8 9 6 4 5 1 2 0	0.200000 0.225125

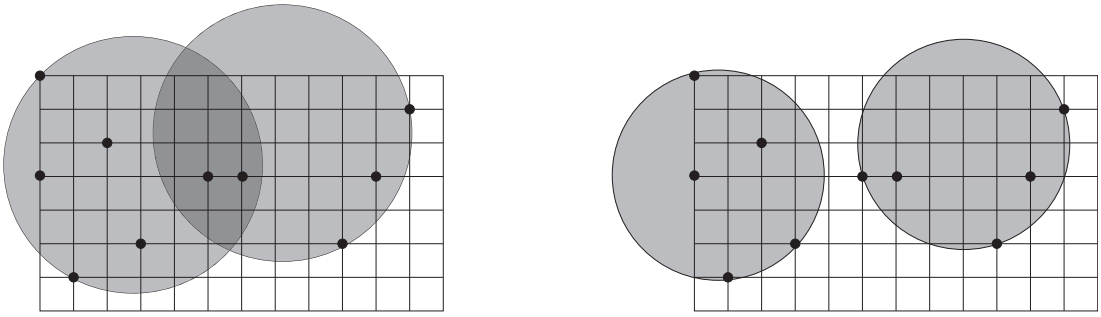
Problema F
Torres de Telefonia Celular

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

Uma nova operadora de telefonia pretende oferecer serviços de telefone residencial em sua cidade. Os telefones serão residenciais, mas a operadora vai utilizar tecnologia de telefonia celular, com torres de transmissão, para evitar os gastos de construir uma rede de cabos por toda a cidade.

A potência do transmissor/receptor colocado em uma torre define o *raio de cobertura* da torre (que por sua vez define a área de cobertura do equipamento, que é um círculo, já que a cidade é perfeitamente plana). O custo do equipamento instalado em cada torre depende de sua potência, e portanto de seu raio de cobertura.

A operadora decidiu que utilizará exatamente duas torres na cidade. O mesmo tipo de equipamento será instalado nas duas torres, ou seja, as duas torres terão o mesmo raio de cobertura. Como a operadora quer poder oferecer o seu serviço para todas as residências, a área de cobertura das duas torres em conjunto deve englobar todas as residências da cidade. Adicionalmente, o raio de cobertura das duas torres deve ser o menor possível, para miniminar o custo dos equipamentos. A figura abaixo mostra duas possíveis configurações de cobertura das duas torres para uma cidade com dez residências. Tanto (a) quanto (b) oferecem cobertura a todas as residências da cidade, mas (b) é a que utiliza o menor raio de cobertura possível.



(a) (b)

Dada a localização de cada residência na cidade, você deve escrever um programa para calcular o menor raio de cobertura das torres, de forma a garantir que todas as residências sejam cobertas.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um número inteiro N, o número de residências da cidade ($3 \leq N \leq 40$). Cada uma das N linhas seguintes contém dois inteiros X e Y, separados por um espaço em branco ($0 \leq X \leq 10^4$ e $0 \leq Y \leq 10^4$), representando a coordenada de uma residência. Cada residência tem uma localização diferente.

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 uma única linha, contendo um número real, escrito com precisão de duas casas decimais, indicando o raio de cobertura do equipamento a ser utilizado nas duas torres.

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

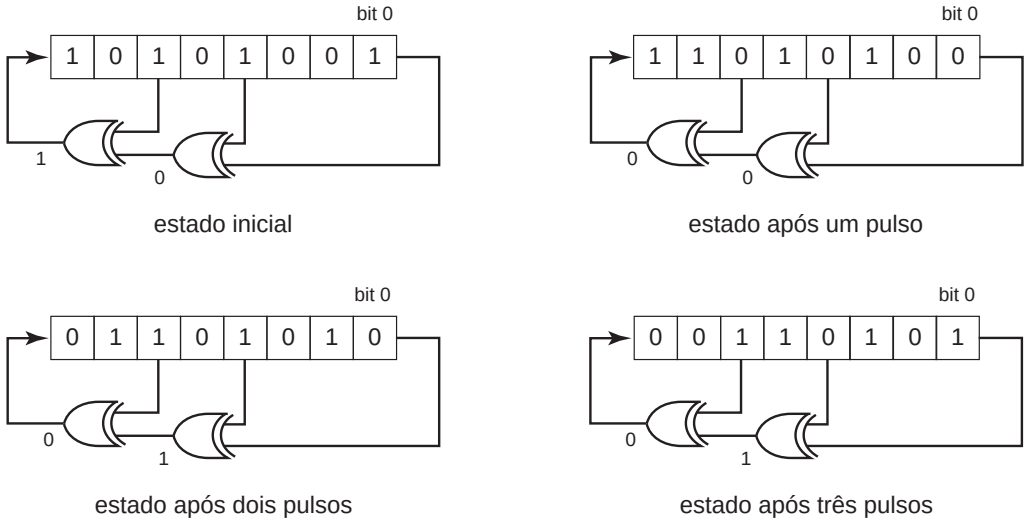
Exemplo de entrada	**Exemplo de saída**
3 0 0 1 0 0 4 10 0 0 0 3 1 6 2 2 3 5 5 3 6 3 9 5 10 5 11 3 0	0.50 3.05

Problema G
Registrador de Deslocamento

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

Um *Registrador de Deslocamento* é um circuito que desloca de uma posição os elementos de um vetor de bits. O registrador de deslocamento tem uma entrada (um bit) e uma saída (também um bit), e é comandado por um pulso de relógio. Quando o pulso ocorre, o bit de entrada se transforma no bit menos significativo do vetor, o bit mais significativo é jogado na saída do registrador, e todos os outros bits são deslocados de uma posição em direção ao bit mais significativo do vetor (em direção à saída).

Um *Registrador de Deslocamento com Retroalimentação Linear* (em inglês, LFSR) é um registrador de deslocamento no qual o bit de entrada é determinado pelo valor do OU-EXCLUSIVO de alguns dos bits do registrador antes do pulso de relógio. Os bits que são utilizados na retroalimentação do registrador são chamados de torneiras. A figura abaixo mostra um LFSR de 8 bits, com três torneiras (bits 0, 3 e 5).



Neste problema, você deve escrever um programa que, dados o número de bits de um LFSR, quais bits são utilizados na retroalimentação, um estado inicial e um estado final do LFSR, determine quantos pulsos de relógio serão necessários para que, partindo do estado inicial, o LFSR chegue ao estado final (ou determinar que isso é impossível).

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por três linhas. A primeira linha contém dois números inteiros N, T, indicando respectivamente o número de bits ($2 \leq N \leq 32$) e o número de torneiras ($2 \leq T \leq N$). Os bits são identificados por inteiros de 0 (bit menos significativo) a $N - 1$ (bit mais significativo). A segunda linha contém T inteiros, separados por espaços, apresentando os identificadores dos bits que são torneiras, em ordem crescente. O bit 0 sempre é uma torneira. A terceira linha contém dois números em notação hexadecimal I e F, separados por um espaço em branco, representando respectivamente o estado inicial e o estado final do LFSR.

O final da entrada é indicado por uma linha que contém dois zeros separados por espaços 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. Se for possível chegar ao estado final a partir do estado inicial dado, a linha da saída deve conter apenas um inteiro, o menor número de pulsos de relógio necessários para o LFSR atingir o estado final. Caso não seja possível, a linha deve conter apenas o caractere '*'.

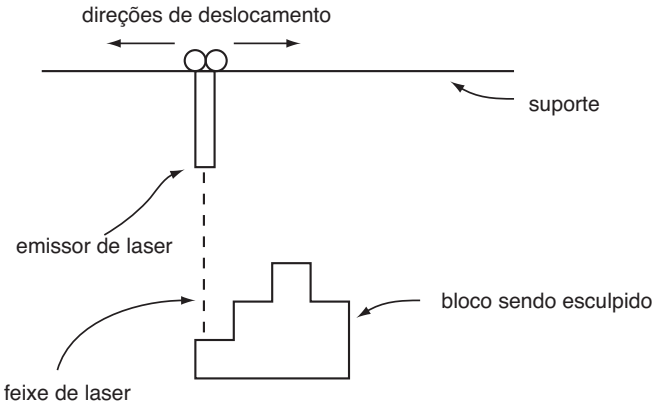
O resultado de seu programa deve ser escrito na saída padrão.

Exemplo de entrada	**Exemplo de saída**
8 3 0 3 5 a9 35 5 2 0 4 1b 2 7 3 0 2 3 4d 1a 0 0	3 * 61

Problema H
Escultura a Laser

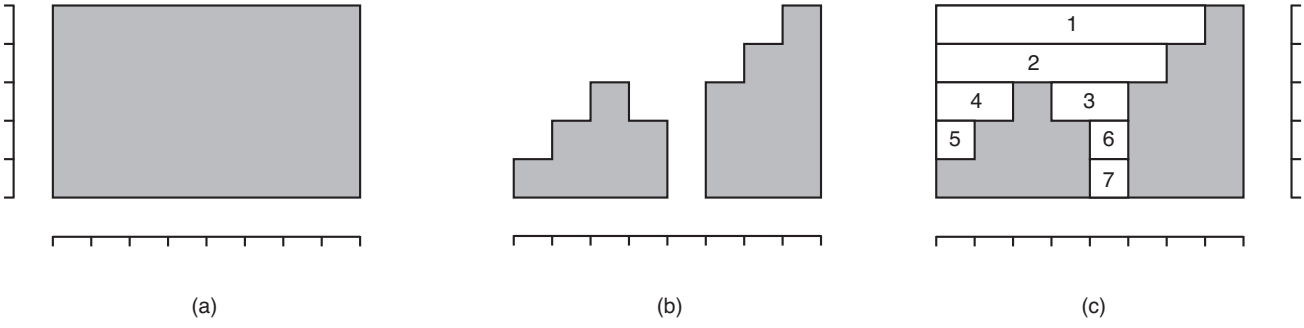
Nome do arquivo fonte: `laser.c`, `laser.cpp` *ou* `laser.java`

Desde a sua invenção, em 1958, os raios laser têm sido utilizados em uma imensa variedade de aplicações, como equipamentos eletrônicos, instrumentos cirúrgicos, armamentos, e muito mais.



A figura acima mostra um diagrama esquemático de um equipamento para esculpir, com laser, um bloco de material maciço. Na figura vemos um emissor laser que se desloca horizontalmente para a direita e para a esquerda com velocidade constante. Quando o emissor é ligado durante o deslocamento, uma camada de espessura constante é removida do bloco, sendo vaporizada pelo laser.

A figura abaixo ilustra o processo de escultura a laser, mostrando um exemplo de (a) um bloco, com 5 mm de altura por 8 mm de comprimento, no início do processo, (b) o formato que se deseja que o bloco esculpido tenha, e (c) a sequência de remoção das camadas do bloco durante o processo, considerando que a cada varredura uma camada de espessura de 1 mm é removida. Na primeira varredura, o pedaço numerado como 1 é removido; na segunda varredura, o pedaço numerado como 2 é removido, e assim por diante. Durante o processo de remoção, o laser foi ligado um total de 7 vezes, uma vez para cada pedaço de bloco removido.



Escreva um programa que, dados a altura do bloco, o comprimento do bloco, e a forma final que o bloco deve ter, determine o número total vezes de que o laser deve ser ligado para esculpir o bloco.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha de um caso de teste contém dois números inteiros A e C, separados por um espaço em branco, indicando respectivamente a altura ($1 \leq A \leq 10^4$) e o comprimento ($1 \leq C \leq 10^4$) do bloco a ser esculpido, em milímetros. A segunda linha contém C números inteiros X_i, cada um indicando a altura final, em milímetros, do bloco entre as posições i e $i + 1$ ao longo do comprimento ($0 \leq X_i \leq A$, para $0 \leq i \leq C - 1$). Considere que a cada varredura uma camada de espessura 1 milímetro é removida do bloco ao longo dos pontos onde o laser está ligado.

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, contendo um numero inteiro, indicando o número de vezes que o laser deve ser ligado para esculpir o bloco na forma indicada.

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

Exemplo de entrada	**Exemplo de saída**
5 8 1 2 3 2 0 3 4 5 3 3 1 0 2 4 3 4 4 1 0 0	7 3 3