acm International Collegiate Programming Contest 2016 IBM event sponsor

Maratona de Programação da SBC 2016

Sub-Regional Brasil do ACM ICPC

10 de Setembro de 2016

Caderno de Problemas

Informações Gerais

Este caderno contém 12 problemas; as páginas estão numeradas de 1 a 16, não contando esta página de rosto. Verifique se o caderno está completo.

A) Sobre os nomes dos programas
1) Sua solução deve ser chamada *codigo_de_problema*`.c`, *codigo_de_problema*`.cpp`, *codigo_de_problema*`.pas`, *codigo_de_problema*`.java` ou *codigo_de_problema*`.py`, onde *codigo_de_problema* é a letra maiúscula que identifica o problema. Lembre que em Java o nome da classe principal deve ser igual ao nome do arquivo.

B) Sobre a entrada
1) A entrada de seu programa deve ser lida da *entrada padrão*.
2) A entrada é composta de um único caso de teste, descrito em um número de linhas que depende do problema.
3) Quando uma linha da entrada contém vários valores, estes são separados por um único espaço em branco; a entrada não contém nenhum outro espaço em branco.
4) Cada linha, incluindo a última, contém exatamente um caractere final-de-linha.
5) O final da entrada coincide com o final do arquivo.

C) Sobre a saída
1) A saída de seu programa deve ser escrita na *saída padrão*.
2) Quando uma linha da saída contém vários valores, estes devem ser separados por um único espaço em branco; a saída não deve conter nenhum outro espaço em branco.
3) Cada linha, incluindo a última, deve conter exatamente um caractere final-de-linha.

Promoção:

SBC Sociedade Brasileira de Computação

v1.0

Problema A

Andando no tempo

Imagine que você tenha uma máquina do tempo que pode ser usada no máximo três vezes, e a cada uso da máquina você pode escolher voltar para o passado ou ir para o futuro. A máquina possui três créditos fixos; cada crédito representa uma certa quantidade de anos, e pode ser usado para ir essa quantidade de anos para o passado ou para o futuro. Você pode fazer uma, duas ou três viagens, e cada um desses três créditos pode ser usado uma vez apenas. Por exemplo, se os créditos forem $5, 12$ e 9, você poderia decidir fazer duas viagens: ir 5 anos para o futuro e, depois, voltar 9 anos para o passado. Dessa forma, você terminaria quatro anos no passado, em 2012. Também poderia fazer três viagens, todas indo para o futuro, usando os créditos em qualquer ordem, terminando em 2042.

Neste problema, dados os valores dos três créditos da máquina, seu programa deve dizer se é ou não possível viajar no tempo e voltar para o presente, fazendo pelo menos uma viagem e, no máximo, três viagens; sempre usando cada um dos três créditos no máximo uma vez.

Entrada

A entrada consiste de uma linha contendo os valores dos três créditos A, B e C ($1 \leq A, B, C \leq 1000$).

Saída

Seu programa deve imprimir uma linha contendo o caracter "`S`" se é possível viajar e voltar para o presente, ou "`N`" caso contrário.

Exemplo de entrada 1	**Exemplo de saída 1**
`22 5 22`	`S`

Exemplo de entrada 2	**Exemplo de saída 2**
`31 110 79`	`S`

Exemplo de entrada 3	**Exemplo de saída 3**
`45 8 7`	`N`

Problema B

Batata quente

Batata quente é uma brincadeira bastante popular entre crianças na escola. A brincadeira é simples: a criança que está com a batata a joga para uma outra criança. Em algum momento, o professor, que não está olhando para o que está acontecendo, irá dizer que a brincadeira acabou. Quando isso acontece, a criança que está com a batata perde.

Uma variação da brincadeira, jogada na fila da cantina, é proposta por um professor. As crianças estão numeradas de 1 a N de acordo com sua posição na fila, onde a criança com o número 1 é a primeira da fila. Cada uma receberá um papel com um número, e sempre que receber a batata, deverá passá-la para a criança na posição anotada em seu papel. O jogo termina com o professor vitorioso se a batata chegar em uma posição menor ou igual a X na fila, com X definido no início da brincadeira. Se isso nunca acontecer, o jogo nunca termina, porém as crianças saem vitoriosas: no dia seguinte todas ganham um desconto na cantina.

O professor começa o jogo jogando a batata para alguma criança na fila. Como sua mira não é muito boa, ele só consegue garantir que vai jogar a batata para alguma criança em um invervalo $L \dots R$ da fila com a mesma probabilidade. Ele está considerando vários possíveis intervalos da fila para iniciar a brincadeira. Para isso, o professor gostaria de descobrir, para cada um desses intervalos, qual o valor de X que ele deve escolher para que o jogo seja o mais justo possível, ou seja, a probabilidade de o jogo terminar seja a mais próxima possível da probabilidade de o jogo não terminar.

Você deve auxiliar o professor a avaliar as propostas. Dados os papéis de cada criança da fila e vários intervalos possíveis, responda, para cada intervalo, o valor de X que torne o jogo mais justo possível. Se houver empate, responda o X mais próximo do início da fila.

Entrada

A primeira linha da entrada contém dois inteiros, N e Q ($2 \leq N \leq 50000$, $1 \leq Q \leq 10^5$). A linha seguinte contém N inteiros $p_1, p_2 \dots p_N$ ($1 \leq p_i \leq N$), os valores dos papéis recebidos por cada uma das crianças. Seguem então Q linhas, cada uma com dois inteiros L e R ($1 \leq L \leq R \leq N$), representando um intervalo considerado pelo professor.

Saída

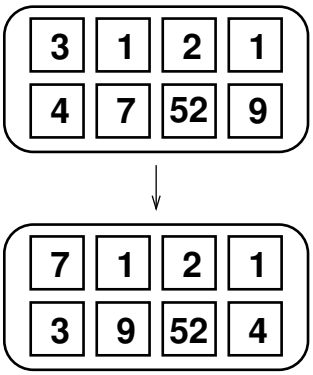
Imprima Q linhas, cada uma contendo, para cada intervalo considerado pelo professor, o número inteiro X que o professor deverá escolher para que a brincadeira seja a mais justa possível.

Exemplo de entrada 1	**Exemplo de saída 1**
9 4 2 3 4 5 6 7 4 9 5 1 3 3 5 2 8 7 9	1 3 3 1

Exemplo de entrada 2	**Exemplo de saída 2**
3 3 1 3 3 1 1 1 2 2 3	1 1 2

Problema C

Containers



O SBC–Sistema de Balanceamento de Containers precisa ser atualizado para funcionar com uma nova classe de navios, a "dois por quatro", que são navios que podem carregar oito grandes containers numa disposição de duas linhas e quatro colunas, como mostrado na figura ao lado. Esses navios possuem um guindaste fixo que é capaz de realizar um único tipo de movimentação: levantar dois containers adjacentes, na linha ou na coluna, e trocá-los de posição. Para acelerar o carregamento nos portos, os oito containers são embarcados em qualquer uma das oito posições, definindo uma configuração inicial. Depois que o navio deixa o porto, o guindaste precisa mover os containers para deixá-los numa configuração final pré-definida para a viagem.

O problema é que o custo de combustível para o guindaste realizar uma movimentação é igual à soma dos pesos dos dois containers adjacentes cujas posições foram trocadas. Dados os pesos dos containers em cada posição nas configurações inicial e final, o SBC precisa computar o custo total mínimo possível de uma sequência de movimentações que leve os containers da configuração inicial à configuração final.

Entrada

A entrada consiste de quatro linhas contendo, cada uma, quatro inteiros entre 1 e 1000, inclusive. As duas primeiras linhas definem os pesos na configuração inicial e as duas últimas linhas, os pesos na configuração final. Sempre existe uma solução, pois os containers nas configurações inicial e final são os mesmos, com as posições possivelmente trocadas.

Saída

Seu programa deve produzir uma única linha contendo um inteiro, representando o custo total mínimo de uma sequência de movimentos que leve da configuração inicial à configuração final.

Exemplo de entrada 1	**Exemplo de saída 1**
3 1 2 1 4 7 52 9 7 1 2 1 3 9 52 4	81

Exemplo de entrada 2	**Exemplo de saída 2**
1 2 3 4 5 10 7 8 1 2 3 4 5 8 7 10	50

Exemplo de entrada 3	**Exemplo de saída 3**
34 5 6 998 4 17 77 84 34 5 6 998 4 17 77 84	0

Problema D

Divisores

Pense um número positivo n. Agora me diga um divisor A de n. Agora me dê um outro número B que não seja divisor de n. Agora um múltiplo C. E um não múltiplo D. O número que você pensou é...

Parece um truque de mágica, mas é matemática! Será que, conhecendo os números A, B, C e D, você consegue descobrir qual era o número original n? Note que pode existir mais de uma solução!

Neste problema, dados os valores de A, B, C e D, você deve escrever um programa que determine qual o menor número n que pode ter sido pensado ou concluir que não existe um valor possível.

Entrada

A entrada consiste de uma única linha que contém quatro números inteiros A, B, C, e D, como descrito acima ($1 \leq A, B, C, D \leq 10^9$).

Saída

Seu programa deve produzir uma única linha. Caso exista pelo menos um número n para os quais A, B, C e D façam sentido, a linha deve conter o menor n possível. Caso contrário, a linha deve conter -1.

Exemplo de entrada 1	**Exemplo de saída 1**
`2 12 8 2`	`4`

Exemplo de entrada 2	**Exemplo de saída 2**
`3 4 60 105`	`6`

Problema E

Estatística hexa

Dada uma sequência de inteiros positivos em hexadecimal, por exemplo, $S = [\texttt{9af47c0b}, \texttt{2545557}, \texttt{ff6447979}]$, definimos $\texttt{soma}(S)$ como sendo a soma de todos os elementos de S. Considere agora uma certa permutação dos 16 dígitos hexadecimais, por exemplo, $p = [\texttt{4}, \texttt{9}, \texttt{5}, \texttt{a}, \texttt{0}, \texttt{c}, \texttt{f}, \texttt{3}, \texttt{d}, \texttt{7}, \texttt{8}, \texttt{b}, \texttt{1}, \texttt{2}, \texttt{6}, \texttt{e}]$. A partir da sequência base S, podemos definir uma sequência transformada $S^{[4]}$, que é obtida pela remoção de todas as ocorrências do dígito hexadecimal 4 de todos os inteiros em S, $S^{[4]} = [\texttt{9af7c0b}, \texttt{255557}, \texttt{ff67979}]$. Em seguida, podemos remover o dígito 9 e obter $S^{[4,9]} = [\texttt{af7c0b}, \texttt{255557}, \texttt{ff677}]$. Seguindo a ordem dos dígitos na permutação p, podemos definir dessa forma 16 sequências: $S^{[4]}, S^{[4,9]}, S^{[4,9,5]}, \ldots, S^{[4,9,5,a,0,c,f,3,d,7,8,b,1,2,6,e]}$. Estamos interessados em somar todos os elementos dessas 16 sequências:

$$\texttt{total}(S, p) = \texttt{soma}(S^{[4]}) + \texttt{soma}(S^{[4,9]}) + \texttt{soma}(S^{[4,9,5]}) + \cdots + \texttt{soma}(S^{[4,9,5,a,0,c,f,3,d,7,8,b,1,2,6,e]})$$

Claramente, esse total depende da permutação p usada na remoção sucessiva. Dada uma sequência de N inteiros positivos em hexadecimal, seu programa deve computar, considerando todas as possíveis permutações dos 16 dígitos hexadecimais: o total mínimo, o total máximo e o somatório dos totais de todas as permutações. Para o somatório dos totais de todas as permutações, imprima o resultado módulo 3b9aca07 ($10^9 + 7$ na base 10).

Entrada

A primeira linha da entrada contém um inteiro N, $\texttt{1} \leq N \leq \texttt{3f}$, representando o tamanho da sequência. As N linhas seguintes contêm, cada uma, um inteiro positivo P, $\texttt{0} \leq P \leq \texttt{fffffffff}$, definindo a sequência inicial S de inteiros. Todos os números na entrada estão em hexadecimal, com letras minúsculas.

Saída

Seu programa deve produzir uma única linha na saída contendo três inteiros positivos, em hexadecimal com letras minúsculas, representando o total mínimo, o total máximo e o somatório dos totais considerando todas as permutações possíveis dos 16 dígitos hexadecimais.

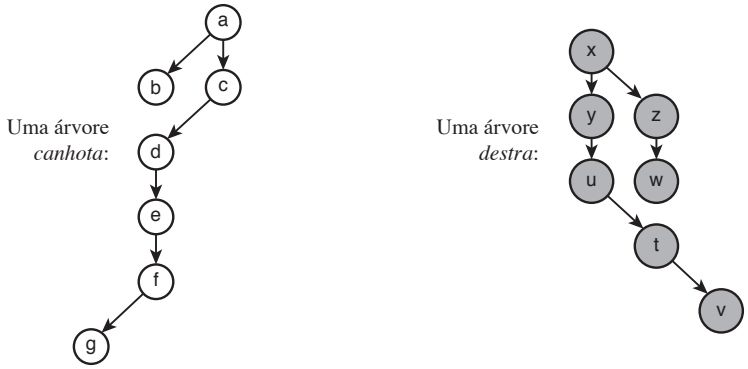
Exemplo de entrada 1	**Exemplo de saída 1**
3 9af47c0b 2545557 ff6447979	1312c99c b4e87e9387 5bb5fc

Exemplo de entrada 2	**Exemplo de saída 2**
1 fffffffff	0 effffffff1 15dac189

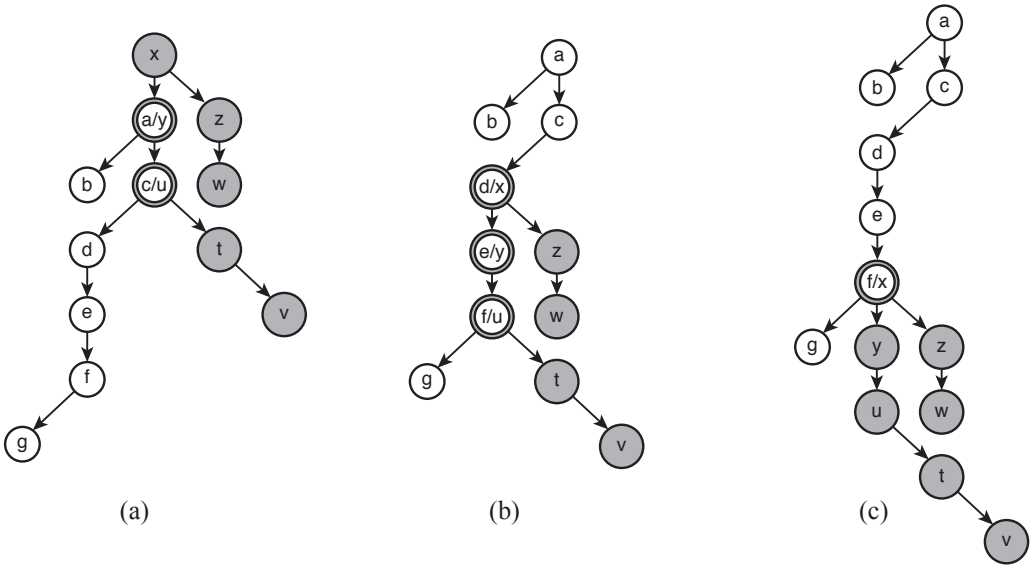
Problema F
Fundindo árvores

Em Computação árvores são objetos estranhos: a raiz está no topo e as folhas estão embaixo! Uma árvore é uma estrutura de dados composta de N vértices conectados por $N-1$ arestas de forma que é possível chegar de um vértice a qualquer outro vértice seguindo as arestas. Em uma árvore *enraizada*, cada aresta conecta um vértice *pai* a um vértice *filho*. Um único vértice não tem pai, e é chamado de *raiz*. Assim, partir da raiz é possivel chegar a qualquer outro vértice da árvore seguindo as arestas na direção de pai para filho.

Em uma árvore *ternária* cada vértice pode ter até três vértices filhos, chamados *esquerdo*, *central* e *direito*. Uma árvore ternária *canhota* é uma árvore ternária enraizada em que nenhum vértice tem filho direito. Uma árvore ternária *destra* é uma árvore ternária enraizada em que nenhum vértice tem filho esquerdo. A raiz de uma árvore ternária é sempre um vértice *central*. A figura abaixo mostra exemplos de uma árvore canhota e de uma árvore destra.



A *superposição* S de uma árvore canhota C com uma árvore destra D é uma árvore ternária enraizada em que a raiz é ou a raiz de C ou a raiz de D ou ambas as raízes, de C e de D, superpostas, e que contém a estrutura de ambas as árvores superpostas. A figura abaixo mostra algumas árvores formadas pela superposição da árvore canhota e da árvore destra da figura acima.



Note que na Figura (a) a raiz é o vértice x (da árvore destra) e os pares de vértices (a, y) e (c, u) são superpostos. Na Figura (b) a raiz é o vértice a (da árvore canhota) e os pares de vértices (d, x),

(e, y) e (f, u) são superpostos. Na Figura (c) a raiz também é o vértice a (da árvore canhota) e o par de vértices (f, x) é superposto.

Dadas uma árvore canhota e uma árvore destra, sua tarefa é determinar o número mínimo de vértices necessários para construir uma árvore ternária que é uma superposição das árvores dadas.

Entrada

A primeira linha de um caso de teste contém um inteiro N indicando o número de vértices da árvore canhota ($1 \leq N \leq 10^4$). Vértices nesta árvore são identificados por números de 1 a N, e a raiz é o vértice de número 1. Cada uma das N linhas seguintes contém três inteiros I, L e K, indicando respectivamente o identificador de um vértice I, o identificador do filho esquerdo L de I e o identificador do filho central K de I ($0 \leq I, L, K \leq N$). A linha seguinte contém um inteiro M indicando o número de vértices da árvore destra ($1 \leq M \leq 10^4$). Vértices nesta árvore são identificados por números de 1 a M, e a raiz é o vértice de número 1. Cada uma das M linhas seguintes contém três inteiros P, Q e R, indicando respectivamente o identificador de um vértice P, o identificador do filho central Q de P e o identificador do filho direito R de P ($0 \leq P, Q, R \leq N$). O valor zero indica um vértice não existente (usado quando um vértice não tem um ou ambos os seus filhos).

Saída

Imprima o número mínimo de vértices de uma árvore que é a superposição das duas árvores dadas na entrada.

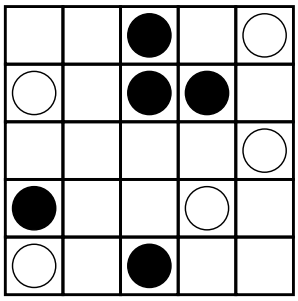
Exemplo de entrada 1	**Exemplo de saída 1**
7 1 2 3 2 0 0 3 4 0 4 0 5 5 0 6 6 7 0 7 0 0 7 1 2 3 2 4 0 3 5 0 4 0 6 5 0 0 6 0 7 7 0 0	11

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

Exemplo de entrada 3	**Exemplo de saída 3**
3 3 0 2 2 0 0 1 0 3 2 2 0 0 1 2 0	3

Problema G
Go−−

Go−− é até parecido com o tradicional jogo de Go, mas é bem mais fácil! Ele é jogado em um tabuleiro quadrado de dimensão N, inicialmente vazio, no qual dois jogadores, um jogando com as pedras pretas e o outro com as brancas, se alternam colocando uma pedra por vez dentro de qualquer célula que ainda não esteja ocupada. A partida termina depois que cada jogador colocou P pedras no tabuleiro. Considere todas as possíveis sub-áreas quadradas de dimensão de 1 a N. Uma sub-área pertence ao jogador que joga com as pedras pretas se ela contém pelo menos uma pedra preta e nenhuma pedra branca. Da mesma forma, uma sub-área quadrada pertence ao jogador que joga com as pedras brancas se contém ao menos uma pedra branca e nenhuma pedra preta. Note que as áreas que não contenham nenhuma pedra, ou que contenham tanto pedras pretas quanto brancas, não pertencem a nenhum jogador.



Neste problema, dada a posição final do tabuleiro, seu programa deve computar quantas sub-áreas quadradas pertencem a cada jogador, para descobrir quem ganhou a partida. Na figura, as pretas possuem 12 sub-áreas (cinco de dimensão 1, seis de dimensão 2 e uma de dimensão 3). As brancas, que perderam a partida, possuem apenas 10.

Entrada

A primeira linha da entrada contém dois inteiros N e P, $2 \leq N \leq 500$, $1 \leq P \leq 500$ e $P \leq N^2/2$, representando, respectivamente, a dimensão do tabuleiro e o número de pedras que cada jogador coloca. Cada uma das P linhas seguintes contém dois inteiros L e C ($1 \leq L, C \leq N$) definindo as coordenadas (linha, coluna) das pedras pretas. Depois, cada uma das próximas P linhas contém dois inteiros L e C ($1 \leq L, C \leq N$) definindo as coordenadas (linha, coluna) das pedras brancas. Todas as pedras são colocadas em células distintas.

Saída

Imprima uma linha contendo dois inteiros separados por um espaço: quantas áreas distintas pertencentes às pretas e às brancas.

Exemplo de entrada 1	**Exemplo de saída 1**
2 1 1 1 2 2	1 1

Exemplo de entrada 2	**Exemplo de saída 2**
5 5 1 3 2 3 2 4 4 1 5 3 1 5 2 1 3 5 4 4 5 1	12 10

Exemplo de entrada 3	**Exemplo de saída 3**
500 3 500 498 500 499 500 500 120 124 251 269 499 498	4 12463784

Problema H

huaauhahhuahau

Em chats, é muito comum entre jovens e adolescentes utilizar sequências de letras, que parecem muitas vezes aleatórias, para representar risadas. Alguns exemplos comuns são:

```
huaauhahhuahau
hehehehe
ahahahaha
jaisjjkasjksjjskjakijs
huehuehue
```

Cláudia é uma jovem programadora que ficou intrigada pela sonoridade das "risadas digitais". Algumas delas ela nem mesmo consegue pronunciar! Mas ela percebeu que algumas delas parecem transmitir melhor o sentimento da risada que outras. A primeira coisa que ela percebeu é que as consoantes não interferem no quanto as risadas digitais influenciam na transmissão do sentimento. A segunda coisa que ela percebeu é que as risadas digitais mais engraçadas são aquelas em que as sequências de vogais são iguais quando lidas na ordem natural (da esquerda para a direita) ou na ordem inversa (da direita para a esquerda), ignorando as consoantes. Por exemplo, "hahaha" e "huaauhahhuahau" estão entre as risadas mais engraçadas, enquanto "riajkjdhhihhjak" e "huehuehue" não estão entre as mais engraçadas.

Cláudia está muito atarefada com a análise estatística das risadas digitais e pediu sua ajuda para escrever um programa que determine, para uma risada digital, se ela é das mais engraçadas ou não.

Entrada

A entrada é composta por uma linha, contendo uma sequência de no máximo 50 caracteres, formada apenas por letras minúsculas sem acentuação. As vogais são as letras '`a`','`e`','`i`','`o`','`u`'. A sequência contém pelo menos uma vogal.

Saída

Seu programa deve produzir uma linha contendo um caractere, "`S`" caso a risada seja das mais engraçadas, ou "`N`" caso contrário.

Exemplo de entrada 1	**Exemplo de saída 1**
`hahaha`	`S`

Exemplo de entrada 2	**Exemplo de saída 2**
`riajkjdhhihhjak`	`N`

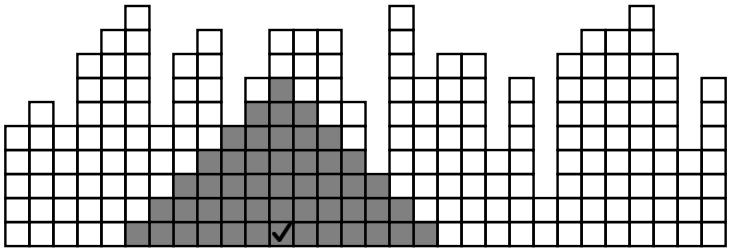
Exemplo de entrada 3	**Exemplo de saída 3**
`a`	`S`

Exemplo de entrada 4	**Exemplo de saída 4**
`huaauhahhuahau`	`S`

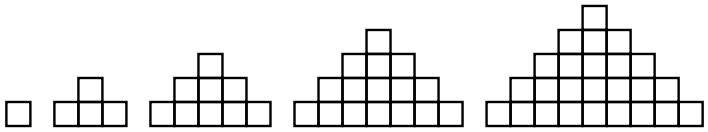
Problema I

Isósceles

Os irmãos Sérgio e Luiz estavam brincando com cubinhos de madeira e queriam construir um muro, que acabou ficando incompleto, com as colunas tendo diferentes alturas, como nessa figura.



Eles decidiram agora que a brincadeira seria retirar cubinhos, sempre de cima para baixo nas colunas, de maneira que no final restasse apenas um triângulo isósceles de cubinhos. Eles podem apenas retirar cubinhos do muro, sem recolocar em outra coluna, e os triângulos têm que ser completos. A figura abaixo ilustra os cinco primeiros triângulos isósceles de cubinhos, do tipo que eles querem, com alturas $1, 2, 3, 4$ e 5 respectivamente.



Dada a sequência de alturas das colunas do muro, seu programa deve ajudar Sérgio e Luiz a descobrir qual é a altura máxima que o triângulo poderia ter ao final. No muro da primeira figura, com 30 colunas de cubinhos, o triângulo mais alto possível teria altura igual a sete.

Entrada

A primeira linha da entrada contém um inteiro N, $1 \leq N \leq 50000$, representando o número de colunas do muro. A segunda linha contém N inteiros A_i, $1 \leq A_i \leq N$, para $1 \leq i \leq N$, indicando as alturas de cada coluna.

Saída

Seu programa deve produzir uma única linha com um inteiro H, representando a altura máxima que um triângulo poderia ter ao final.

Exemplo de entrada 1	**Exemplo de saída 1**
16 5 6 5 8 9 10 5 8 9 5 7 9 9 9 6 3	6

Exemplo de entrada 2	**Exemplo de saída 2**
8 5 1 1 1 1 1 1 3	1

Problema J

Jogos olímpicos

Um grupo de investidores está pensando em investir pesado em atletas da delegação brasileira após as olimpíadas do Rio. Para isso, eles vêm observando N atletas e perceberam que alguns estão em decadência e outros em ascensão. Em especial, o grupo está de olho em dois fatores sobre cada atleta: seu cansaço e sua habilidade. Eles anotaram os valores de habilidade e cansaço de cada atleta logo ao final das olimpíadas de 2016. Em seguida, o grupo estimou a taxa com a qual cada atleta perde ou ganha habilidade e a taxa com a qual cada atleta se cansa ao longo do tempo, e percebeu que essas taxas são constantes para os dois atributos.

Os investidores perceberam que esses dados lhes permitem definir o que resolveram chamar de atleta de ouro: um atleta que, em um determinado período de tempo, é o atleta menos cansado e o mais habilidoso. Ficou decidido que investimentos serão feitos apenas em atletas de ouro. Descubra quantos jogadores, entre os observados inicialmente, receberão algum investimento. Considere que o tempo $t = 0$ é o tempo das olimpíadas do Rio: nenhum atleta que foi de ouro antes desse tempo pode receber investimento. Considere também que qualquer tempo após as olimpíadas do Rio deve ser considerado, por maior que seja. Um atleta que é de ouro exatamente no tempo $t = 0$ deve ser contado.

Entrada

A primeira linha da entrada contém um inteiro, N ($1 \leq N \leq 10^5$), o número de atletas. Seguem N linhas, cada uma com quatro números inteiros: H_i, H_i^t, C_i, C_i^t ($-10^6 < H_i, H_i^t, C_i, C_i^t \leq 10^6$, $H_i^t, C_i^t \neq 0$), representando, respectivamente, a habilidade ao final das olimpíadas, a taxa de variação da habilidade, o cansaço ao final das olimpíadas e a taxa de variação do cansaço do i-ésimo atleta.

Saída

Seu programa deve produzir uma única linha com um inteiro O, representando o número de atletas que receberão algum investimento do grupo.

Exemplo de entrada 1	**Exemplo de saída 1**
3 3 2 1 2 2 2 2 2 1 2 3 2	1

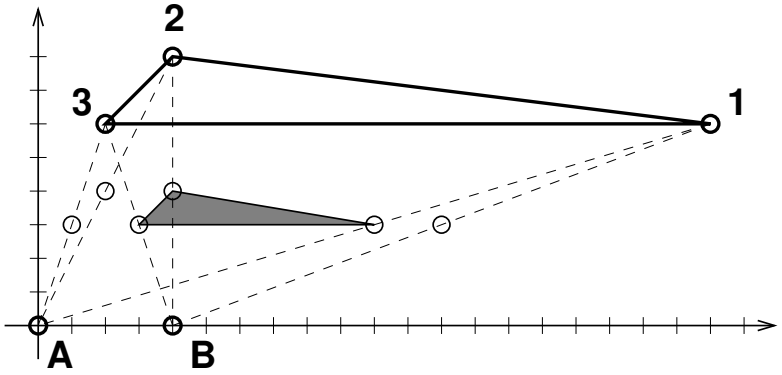
Exemplo de entrada 2	**Exemplo de saída 2**
6 1 10 5 8 8 7 12 -5 10 -2 -3 8 -3 -5 -8 -12 0 1 10 2 8 3 9 -3	0

Problema K

Kit de encolhimento de polígonos

Um *Kit de Encolhimento de Polígonos* é um material muito utilizado nas aulas de magia geométrica na Nlogônia. O *kit* consiste de dois pontos, A e B no plano cartesiano. Considere um polígono convexo dado pelos vértices $1, 2...N$, nessa ordem. Para encolher esse polígono usando o *kit*, algumas regras devem ser respeitadas. Cada vértice x do polígono deve ser movido uma vez só: para o ponto médio do segmento Ax ou para o ponto médio do segmento Bx. A operação de encolhimento deve produzir um novo polígono convexo que preserve a ordem relativa dos vértices do polígono original. Em outras palavras, considerando todas as possíveis maneiras de aplicar o kit, apenas aquelas cuja sequência final dos vértices $1, 2...N$ representa um polígono convexo são válidas. Veja que o polígono convexo original pode estar em sentido horário e uma operação de encolhimento válida produzir um polígono convexo em sentido anti-horário, na mesma ordem dos vértices. Apenas a ordem relativa dos pontos é importante, não o sentido.

É sabido que magia geométrica não é o forte da maioria dos alunos. A professora pediu que eles usassem o *kit* de encolhimento para encolher um polígono convexo fornecido por ela de forma a obter a menor área possível e um amigo seu implorou para que você resolva a questão por ele. Responda a menor área possível do polígono para ele.



A Figura acima ilustra um uso válido do *kit*, onde o polígono sombreado é o de menor área possível que preserva a sequência dos vértices. Os pontos A e B correspondem aos pontos do *kit*. Note que, apesar do nome *encolhimento*, às vezes é possível utilizar o *kit* para aumentar a área dos polígonos! Como geometria é difícil!

Observe que um único ponto ou uma reta não são considerados polígonos. Sendo assim, se um uso do *kit* produzir como resultado algo diferente de um polígono convexo, esse não é um uso válido.

Entrada

A primeira linha da entrada contém um inteiro N ($3 \leq N \leq 10^5$), o número de vértices do polígono. Seguem N linhas, cada uma com dois inteiros x, y ($-10^6 \leq x, y \leq 10^6$), os vértices do poligono. A última linha da entrada contém quatro inteiros, A_x, A_y, B_x e B_y ($-10^6 \leq A_x, A_y, B_x, B_y \leq 10^6$), as coordenadas x e y de A e as coordenadas x e y de B, respectivamente. Os pontos da entrada serão dados na ordem correta em que aparecem no polígono, no sentido horário ou anti-horário. Não haverão pontos repetidos e o polígono será convexo.

Saída

Seu programa deve produzir uma linha, contendo um número real, com 3 casas decimais de precisão, representando a menor área possível para um polígono obtido com o uso do *kit*.

Exemplo de entrada 1	**Exemplo de saída 1**
3 20 6 4 8 2 6 0 0 4 0	3.500

Exemplo de entrada 2	**Exemplo de saída 2**
3 0 4 4 4 0 0 3 -2 -3 -2	1.000

Exemplo de entrada 3	**Exemplo de saída 3**
3 0 4 4 4 0 0 2 -2 -2 -2	2.000

Problema L

Ladrilhos

Avelino tem um mosaico em uma das paredes de sua casa. É uma mosaico muito antigo, composto por pequenos ladrilhos coloridos. Como é um mosaico antigo, alguns ladrilhos se soltaram ao longo dos anos formando buracos.

Agora, Avelino quer restaurar o mosaico cobrindo os buracos com novos ladrilhos. Entretanto, para economizar, Avelino quer comprar ladrilhos de uma única cor para tapar os buracos. Em particular, quer comprar ladrilhos de uma das cores originais ou de uma cor ainda não contida no mosaico.

Por ser um mosaico, não se deseja que hajam áreas muito grandes com a mesma cor. Avelino resolveu que vai escolher a cor dos ladrilhos tentando fazer com que o tamanho da menor área monocromática seja o menor possível, para que haja mais detalhes. Veja que pode existir mais de uma cor possível. Uma área é monocromática se todos os ladrilhos nela são da mesma cor. Dois ladrilhos adjacentes fazem parte da mesma área se possuem a mesma cor, e dois ladrilhos são adjacentes se compartilham um lado.

Veja o primeiro caso de exemplo, temos três áreas da cor 1 (uma de tamanho 3 e duas de tamanho 2), uma área da cor 2 (de tamanho 3) e uma área da cor 3 de tamanho 7. Uma resposta possível seria escolher a cor 2, fazendo com que a menor área monocromática seja de tamanho 2. Se escolhermos a cor 1 a menor área seria de tamanho 3.

Crie um programa que imprima o tamanho da menor área possível.

Entrada

A primeira linha contém dois inteiros H e L, a altura e largura do mosaico, respectivamente, satisfazendo $1 \leq H \leq 200$ e $1 \leq L \leq 200$. Em seguida, H linhas conterão cada uma L inteiros, separados por espaço, correspondendo às cores dos ladrilhos. Um inteiro 0 corresponde a um buraco e um inteiro $i \neq 0$ corresponde a um ladrilho com a i-ésima cor, podendo ir de 1 até 40000 no máximo.

Saída

Seu programa deve produzir uma linha, contendo um inteiro, o tamanho da menor área possível.

Exemplo de entrada 1	**Exemplo de saída 1**
3 8 3 3 3 1 1 0 0 0 3 1 1 0 2 2 0 1 3 3 3 0 0 2 1 1	2

Exemplo de entrada 2	**Exemplo de saída 2**
3 7 1 1 0 2 2 1 1 1 1 0 2 2 1 1 1 1 0 0 3 3 3	3

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