8^a Seletiva da UFPR

4 de Agosto de 2017

Sevidor BOCA:
`http://maratona.c3sl.ufpr.br/`

Flávio Zavan
Ricardo Oliveira

Instruções Importantes

- Em cada problema, cada arquivo de entrada contém apenas um caso de teste. Sua solução será executada com vários arquivos de entrada.
- Se a solução der erro ou esgotar o tempo limite para um dado arquivo de entrada, você receberá a indicação de erro (estouro de tempo, resposta errada, etc.) para aquele arquivo, e a execução terminará. O arquivo que causou o erro não é identificado. Note que pode haver outros erros, de outros tipos, para outros arquivos de entrada, mas apenas o primeiro erro encontrado é reportado.
- Sua solução será compilada com a seguinte linha de comando:
  - C: `gcc -static -O2 -lm`
  - C++: `g++ -static -O2 -lm`
  - C++11: `g++ -std=c++11 -static -O2 -lm`
  - Java: `javac`
  - Pascal: `fpc -Xt -XS -O2`
  - Python3: `python3`
- Sua solução deve processar cada arquivo de entrada no tempo máximo estipulado para cada problema, dado pela seguinte tabela:

Problema	**Nome**	**Tempo Limite (segundos)**
A	Morro da Tiririca	1
B	Fusão	1
C	Bomba	1
D	Alerta de Spoiler	1
E	Seleção Mitológica	1
F	Dracarys	1
G	BowserSort	1
H	Pastel	1
I	Cerca do Jardim	1

- Os juizes usam um sistema de 64 bits (idêntico às maquinas do DINF).
- Todas as linhas, tanto na entrada quanto na saída, terminam com o caractere de fim-de-linha ($\backslash n$), mesmo quando houver apenas uma única linha no arquivo.
- Para submissões em **JAVA**, a classe deverá ter o mesmo nome que o *basename* do problema (leia a linha entre o título e o texto do problema).

A: Morro da Tiririca

`Arquivo: tiririca.[c|cpp|java|pas|py]`

A Serra do Bar corta a Nlogônia no sentido norte-sul, divindo o país em duas regiões distintas, a leste, o litoral e a oeste, o planalto, onde está localizada a capital da nação. Com as novas obras públicas de infraestrutura, o governo nlogono irá instalar um moderno sistema de comunicação a laser, permitindo a ligação da capital com o litoral. Cada região construirá uma estação, permitindo a conexão. Para que isto seja possível, o laser precisa ser disparado com precisão de uma estação a outra. Entretando, como há uma serra entre as duas regiões impedindo a comunicação direta, foi determinado que será instalada uma placa refletora na serra, a fim de redirecionar o laser para atingir a outra estação. Para este fim, foram realizados estudos do terreno e determinou-se que o local ideal para a instalação da placa é no Morro da Tiririca, uma das montanhas mais altas do país, com um amplo campo de altitude ao redor do cume.

Com o projeto de construição definido, o governo da Nlogônia pediu para que você escreva um programa que, dado o tamanho da placa, sua localização e a posição das estações, determine se será possível uma estação se comunicar com a outra.

Entrada

Os objetos são representados no plano cartesiano 2D. A entrada é composta de exatamente quatro linhas, cada uma com dois números inteiros entre 0 e 1000. Sendo a primeira linha, o x e o y da estação da capital. Seguido de uma linha similar, representando a estação do litoral. As duas últimas linhas representam cada uma, a posição de um extremo da placa refletora. É garantido que nenhuma das estações é colinear com os pontos que formam a placa. Também garante-se que todos os pontos são distintos.

Saída

Imprima uma linha contendo `Y` se for possível disparar um laser linear de uma estação, refletir na placa e atingir a outra estação. Caso contrário, imprima `N`.

Exemplo de entrada	Exemplo de saída
2 2 4 2 2 3 4 3	Y

Exemplo de entrada	Exemplo de saída
2 2 4 4 2 3 4 3	N

Exemplo de entrada	Exemplo de saída
2 2 8 2 2 3 4 3	N

B: Fusão

`Arquivo: fusao.[c|cpp|java|pas|py]`

O grande vilão Majin Boo está destruindo o universo dos números naturais $\mathbb{N}$, e precisa ser parado o mais rápido possível! Para poder derrotar o vilão, os números mais poderosos deste universo – os números primos – estão treinando uma técnica milenar: a fusão.

Um número primo é um número natural que contém exatamente dois divisores distintos. Um número é resultado de uma fusão se pode ser escrito pelo produto de exatamente dois números primos (distintos ou não). Tal número é dito $F-$primo. Como exemplo, os números 10 ($= 2 \times 5$), 21 ($= 7 \times 3$) e 9 ($= 3 \times 3$) são $F-$primos, enquanto os números 42, 5 e 27 não são $F-$primos.

Dado um inteiro N, determine se N é um $F-$primo (e, portanto, se ele pode derrotar o malvado vilão).

Entrada

A entrada contém uma linha contendo um inteiro N ($1 \leq N \leq 2 \times 10^9$).

Saída

Imprima uma linha contendo `F-primo` se N é $F-$primo, ou `wasted` caso contrário.

Exemplo de entrada	**Exemplo de saída**
`10`	`F-primo`

Exemplo de entrada	**Exemplo de saída**
`21`	`F-primo`

Exemplo de entrada	**Exemplo de saída**
`42`	`wasted`

C: Bomba

`Arquivo: bomba.[c|cpp|java|pas|py]`

Por conta da crescense tensão na fronteira com a Nquadradônia, cientistas nlogonos desenvolveram uma nova bomba geométrica para ser usada pela ofensiva militar. Ao atingir o solo, a bomba destrói uma área perfeitamente circular. Para se adequar às necessidades militares, cada modelo fabricado conta com uma etiqueta indicando sua área de destruição. Entretanto, os generais estão insatisfeitos, uma vez que estão interessados no raio de destruição e não na área. Por isso, um deles mandou você escrever um programa que calcule o raio de destruição da bomba, dada a área destruída.

Entrada

A entrada é composta de uma única linha contendo um único número inteiro A ($1 \leq A \leq 100$), a área de destruição da bomba em quilômetros.

Saída

Imprima uma única linha contendo o raio de destruição da bomba em quilômetros com precisão de três casas decimais.

Exemplo de entrada	**Exemplo de saída**
`10`	`1.784`

Exemplo de entrada	**Exemplo de saída**
`20`	`2.523`

D: Alerta de Spoiler

`Arquivo: spoiler.[c|cpp|java|pas|py]`

Como todo bom maratonista, Vasya está "maratonando" a série mais repercutida do momento: *Game of Ice and Fire*. Como a série é muito famosa, todos os amigos de Vasya estão comentando sobre ela nas redes sociais. Vasya tem muito medo de *spoilers*, que são eventos que acontecem na série tais que Vasya ainda não assistiu o episódio em que ocorrem. Por isso, ele pediu para você (que já terminou a série) para escrever um programa para ajudá-lo a evitar *spoilers*.

São dados os eventos que ocorrem na série, relações de precedência entre eles, o evento que ocorreu no último episódio que Vasya assistiu e várias consultas. Para cada consulta, determine se um dado evento já foi assistido, é um *spoiler* (isto é, se Vasya ainda não o assistiu) ou se não é possível determinar se já foi assistido ou não.

Entrada

A primeira linha contém um inteiro N ($1 \leq N \leq 1000$), o número de eventos na série. As próximas N linhas contém uma *string* cada, de até 15 letras minúsculas, descrevendo os eventos, *em qualquer ordem*.

A próxima linha contém um inteiro M ($0 \leq M \leq \frac{N \times (N-1)}{2}$), o número de relações conhecidas. As próximas M linhas contém uma relação de precedência cada. Cada linha contém dois eventos A e B, indicando que o evento A ocorreu *antes* do evento B na série (nesta ordem). É garantido que as relações dadas são consistentes (isto é, nunca ocorre de um evento A acontecer antes de um evento B e, simultaneamente, B acontecer antes de A, direta ou indiretamente).

A próxima linha contem um evento U, o evento que ocorreu no último episódio que Vasya assistiu.

A próxima linha contém um inteiro Q ($1 \leq Q \leq 1000$), o número de consultas. Por fim, as próximas Q linhas contém uma consulta cada. Cada linha contém um evento E.

Saída

Para cada consulta, imprima uma linha contendo `spoiler` se Vasya ainda não assistiu ao evento, `safe` se ele já assistiu, ou `?` se não for possível determinar se ele já assistiu ou não.

Exemplo de entrada	Exemplo de saída
8 brancai ladymorre dragaonasce viserysmorre nedmorre lobinhos drogomorre danycasa 7 lobinhos ladymorre ladymorre viserysmorre brancai viserysmorre viserysmorre drogomorre danycasa drogomorre viserysmorre nedmorre drogomorre dragaonasce viserysmorre 4 lobinhos viserysmorre danycasa dragaonasce	safe safe ? spoiler

E: Seleção Mitológica

`Arquivo: selecaomitologica.[c|cpp|java|pas|py]`

Nova febre entre adolescentes e jovens adultos, o *game* Seleção Mitológica combina elementos de jogos de tiro em primeira pessoa e jogos de estratégia em tempo real. Seu forte apelo está nos combates alucinantes travados entre uma espécie alienígena e famosas criaturas mitológica em um ambiente virtual.

Com o grande sucesso, os desenvolvedores estão preparando um pacote de atualizações. Dentre elas, está a possibilidade de realizar o balanceamento automático dos dois times (aliens e criaturas mitológicas), baseado no nível de habilidade de cada jogador. Como você é o novo estagiário, ficou encarregado desta implementação.

Dada a quantidade de jogadores e seus níveis de habilidade, um jogo é balanceado quando a diferença no número de jogadores dos dois times é menor ou igual a 1 e a diferença da soma dos níveis de habilidade dos jogadores de cada time é mínima.

Entrada

A primeira linha de entrada é composta por um único inteiro N ($2 \leq N \leq 100$), o número de jogadores. Cada uma das N linhas seguintes contem um inteiro h_i ($1 \leq h_i \leq 10$), o nível de habilidade do i-ésimo jogador.

Saída

Imprima uma única linha contendo a diferença da soma dos níveis de habilidade dos jogadores do jogo balanceado.

Exemplo de entrada	**Exemplo de saída**
8 5 3 6 10 2 8 5 1	0

Exemplo de entrada	**Exemplo de saída**
3 1 1 9	7

F: Dracarys

`Arquivo: dracarys.[c|cpp|java|pas|py]`

Os mestres escravistas estão atacando Meereen com seus grandes navios! Para acabar com o ataque, a rainha de Meereen, Daenerys Targaryen, decidiu usar seus poderosos dragões para destruir todos os navios dos mestres.

Existem N navios inimigos, numerados de 1 a N, dispostos em uma linha reta no mar, um ao lado do outro. O navio 1 está mais à esquerda, enquanto o navio N está mais à direita. Para cada navio i, Daenerys sabe a quantidade q_i de soldados inimigos atualmente no navio.

Para atacar, Daenerys utiliza, várias vezes, a seguinte estratégia: primeiramente, ela escolhe algum navio e fica voando em seus dragões exatamente em cima do navio escolhido. Em seguida, ela fala "*Dracarys*". Isto faz com que os dragões queimem e derrotem D soldados no navio imediatamente abaixo dela, D soldados no navio imediatamente à esquerda deste (se houver), e D soldados no navio imediatamente à direita deste (se houver). Naturalmente, se houver menos de D soldados em um destes navios, todos os que estiverem nele são derrotados.

Um navio é destruído quando não tem mais soldados. Determine o número mínimo de vezes que Daenerys precisa falar "*Dracarys*" para destruir todos os navios inimigos.

Entrada

A primeira linha contém dois inteiros N e D ($1 \leq N \leq 10^5, 1 \leq D \leq 100$), o número de navios e o poder de fogo dos dragões como descrito acima, respectivamente. A segunda linha contém N inteiros $q_1, q_2, ..., q_N$ ($1 \leq q_i \leq 100$ para todo $1 \leq i \leq N$), a quantidade de soldados inicialmente em cada navio.

Saída

Imprima uma linha contendo um inteiro indicando o número mínimo de vezes que a rainha deve falar "*Dracarys*".

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

Exemplo de entrada	**Exemplo de saída**
`5 2` `3 2 5 1 3`	`4`

G: BowserSort

`Arquivo: bowser.[c|cpp|java|pas|py]`

Após estudar o método de ordenação *ShellSort*, o rei Bowser inventou sua própria versão do método, batizada de *BowserSort*.

Como sabemos, o rei Bowser não é muito esperto. O BowserSort funciona da seguinte maneira: A cada passo, Bowser troca dois elementos em posições aleatórias do vetor. Bowser executa vários passos, até se cansar. Note que o método não é muito eficiente; de fato, o vetor pode nem estar ordenado após a execução do algoritmo! Além disso, Bowser também não sabe direito como definir se o vetor está ordenado ou não – ao invés de verificar todo o vetor, ele pode verificar apenas algum intervalo no mesmo.

Como ele está muito ocupado planejando seu 42^o sequestro da princesa Peach, ele pediu que você o ajudasse a simular o BowserSort e determinar, diversas vezes durante sua execução, se e como vetor está ordenado (de acordo com seu critério).

Entrada

A primeira linha contém um inteiro N ($2 \leq N \leq 10^5$), o tamanho do vetor. A segunda linha contém N inteiros v_1, v_2, ..., v_N, o vetor inicial ($1 \leq v_i \leq 10^9$ para todo $1 \leq i \leq N$). A próxima linha contém um inteiro Q ($1 \leq Q \leq 10^5$), o número de operações. As próximas Q linhas contém uma operação cada. Cada operação pode ser do tipo:

- `S` i j ($1 \leq i, j \leq N, i \neq j$): Troque os elementos v_i e v_j entre si;
- `Q` i j ($1 \leq i < j \leq N$): Consulte o intervalo $v_{i...j}$. Note que $i < j$ e, logo, é garantido que o intervalo tem pelo menos dois elementos.

Saída

Para cada consulta, imprima uma linha contendo `A` se os elementos do intervalo estão em ordem *estritamente* crescente, `D` se tais elementos estão em ordem *estritamente* decrescente, ou `X` caso contrário.

Exemplo de entrada	**Exemplo de saída**
6 2 2 3 3 1 1 5 Q 1 3 Q 2 3 S 5 1 Q 1 3 Q 4 6	X A A D

H: Pastel

`Arquivo: pastel.[c|cpp|java|pas|py]`

Andressa pediu demissão de seu emprego e abriu uma barraca de pastel. Ao longo do dia ela realiza várias transações monetárias, algumas positivas (vendas de pastel) e outras negativas (pagamento de contas e ingredientes). Como Andressa anda muito ocupada cuidando de seu novo negócio, você, sobrinha dela, irá escrever um programa que dadas todas as transações monetárias realizadas por Andressa, indique o saldo final.

Entrada

A primeira linha contém um inteiro N ($1 \leq N \leq 100$), o número de transações realizadas. As N linhas seguintes indicam o valor v_i de cada transação em centavos ($-100000 \leq v_i \leq 100000$), sendo gastos representados por números negativos.

Saída

Imprima uma única linha contendo o saldo final de Andressa após todas as transações.

Exemplo de entrada	**Exemplo de saída**
3 -3 -5 -10	-18

Exemplo de entrada	**Exemplo de saída**
5 10 5 -3 -4 2	10

I: Cerca do Jardim

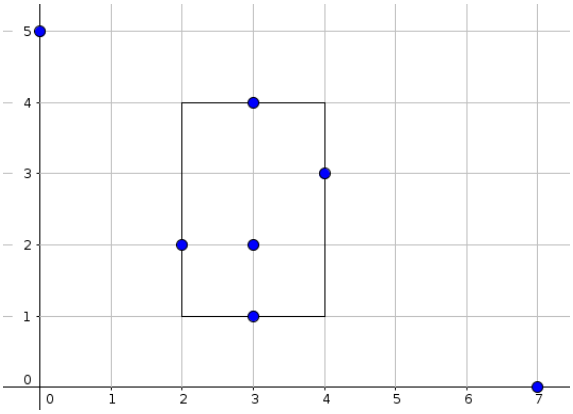
`Arquivo: cerca.[c|cpp|java|pas|py]`

O famoso Jardim de Cima é um belo jardim mantido por sua família há décadas. Há muitas flores bonitas no jardim, mas não há nenhuma cerca. Por isso, você decidiu contratar o jardineiro Jaime para construir uma cerca em volta de algumas flores do jardim.

Para lhe impressionar, Jaime não está interessado em cercar o maior número de flores possível, mas sim em construir a melhor cerca possível. Conhecidas as coordenadas (x_i, y_i) de cada flor no jardim, Jaime quer construir uma cerca tal que:

- seu formato é retangular e seus lados são paralelos aos eixos x e y;
- há pelo menos uma flor exatamente sob a cerca em cada um dos seus lados;
- seu comprimento é no mínimo 0 e no máximo C, e sua largura é a máxima possível.

Em outras palavras, se (X_1, Y_1) e (X_2, Y_2) são vértices opostos do retângulo que forma a cerca, então deve haver ao menos uma flor em cada um dos segmentos de reta que formam seus lados, $0 \leq |X_2 - X_1| \leq C$, e o valor de $|Y_2 - Y_1|$ deve ser maximizado. Se uma flor estiver em um dos cantos do retângulo, considere que ela está simultaneamente sob todos os lados que o compõe.



Sua tarefa é ajudar Jaime a determinar a largura máxima que sua cerca pode ter, isto é, o maior valor possível de $|Y_2 - Y_1|$. Note que a quantidade de flores dentro dela não é relevante. A figura ao lado demonstra o primeiro exemplo de entrada e uma possível solução.

Entrada

A primeira linha contém dois inteiros N e C ($2 \leq N \leq 10^5$, $1 \leq C \leq 10^9$), o número de flores e o comprimento máximo da cerca, respectivamente. As próximas linhas contém dois inteiros x_i e y_i cada ($0 \leq x_i, y_i \leq 10^9$), indicando as coordenadas das flores. Não há duas flores na mesma localização.

Saída

Imprima uma linha contendo um inteiro indicando a maior largura possível que a cerca pode ter.

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

Exemplo de entrada	Exemplo de saída
2 3 2 4 5 2	2

Exemplo de entrada	Exemplo de saída
2 2 2 4 5 2	0

No segundo exemplo, o retângulo contendo os dois pontos dados em suas extremidades satisfaz as restrições dadas. No terceiro exemplo, a solução consiste em um retângulo degenerado que contém apenas (qualquer) um ponto dado, que também satisfaz as restrições dadas.