OBI2011

Caderno de Tarefas

Modalidade **Programação** • Nível **1**, Fase **1**

26 de março de 2011

A PROVA TEM DURAÇÃO DE **4** HORAS

Promoção:

Sociedade Brasileira de Computação

Patrocínio:

Fundação Carlos Chagas

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 9 páginas (não contando a folha de rosto), numeradas de 1 a 9. Verifique se o caderno está completo.
- A prova deve ser feita individualmente.
- É proibido consultar a Internet, livros, anotações ou qualquer outro material durante a prova. É permitida a consulta ao *help* do ambiente de programação se este estiver disponível.
- As tarefas têm o mesmo valor na correção.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa.
- Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
- As tarefas **não** estão ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
- Preste muita atenção no nome dos arquivos fonte indicados nas tarefas. Soluções na linguagem C devem ser arquivos com sufixo *.c*; soluções na linguagem C++ devem ser arquivos com sufixo *.cc* ou *.cpp*; soluções na linguagem Pascal devem ser arquivos com sufixo *.pas*; soluções na linguagem Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; e soluções na linguagem Python devem ser arquivos com sufixo *.py*. Para problemas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada problema.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou disquete, conforme especificado pelo seu professor.
- Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (normalmente é o teclado) e escritos na saída padrão (normalmente é a tela). Utilize as funções padrão para entrada e saída de dados:
  - em Pascal: *readln, read, writeln, write*;
  - em C: *scanf, getchar, printf, putchar*;
  - em C++: as mesmas de C ou os objetos *cout* e *cin*.
  - em Java: qualquer classe ou função padrão, como por exemplo *Scanner, BufferedReader, BufferedWriter* e *System.out.println*
  - em Python: *read,readline,readlines,print,write*
- Procure resolver o problema de maneira eficiente. Na correção, eficiência também será levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo nas tarefas.

Corrida

Nome do arquivo fonte: `corrida.c`, `corrida.cpp`, `corrida.pas`, `corrida.java`, *ou* `corrida.py`

Todo ano, os habitantes da Mlogônia, apesar das crises internas, reúnem-se em torno de um esporte que é a paixão nacional: as corridas de carros. A Grande Corrida anual é um enorme evento organizado pela Associação de Corridas da Mlogônia (ACM), sendo amplamente televisionado e reportado em jornais e revistas de todo o país. Os resultados da corrida são tema principal das rodas de conversa por semanas.

Por bastante tempo, os resultados da Grande Corrida eram compilados manualmente. Observadores especializados iam à pista medir o tempo de cada um dos N carros, numerados de 1 a N, em cada uma das M voltas, anotando então os resultados em tabelas para posterior análise por parte das equipes e dos jornalistas. Muitos erros eram introduzidos nesse processo, e a organização decidiu informatizar todo o sistema.

A ACM percebeu que o esforço necessário para a construção do sistema seria grande, e optou por contar com a ajuda de uma equipe de programadores. Percival foi contratado para escrever a parte do software que determina quais foram os carros vencedores, mas está com dificuldades e pede sua ajuda. A sua tarefa, neste problema, é determinar os três carros melhor colocados, fornecidos os tempos que cada carro levou para completar cada volta da corrida.

Entrada

A primeira linha da entrada contém dois inteiros N e M representando o número de carros e o número de voltas da corrida, respectivamente.

Cada uma das N linhas seguintes representa um carro: a primeira linha representa o primeiro carro, a segunda linha representa o segundo carro, e assim por diante. Cada linha contém M inteiros representando os tempos em cada volta da corrida: o primeiro inteiro é o tempo da primeira volta, o segundo inteiro é o tempo da segunda volta, e assim por diante.

Garante-se que não houve dois carros que gastaram o mesmo tempo para completar a corrida inteira.

Saída

A saída consiste de três linhas, contendo um único inteiro cada. A primeira linha contém o número do carro que ganhou a corrida, a segunda contém o número do segundo colocado e a terceira contém o número do terceiro colocado.

Restrições

- $3 \leq N \leq 100$
- $1 \leq M \leq 100$
- $1 \leq$ qualquer número da entrada que represente o tempo de uma volta $\leq 10^6$

Informações sobre a pontuação

- Em um conjunto de casos de teste que totaliza 20 pontos, $N = 3$;
- Em um conjunto de casos de teste que totaliza 20 pontos, $M = 1$;

Exemplos

Entrada	**Saída**
3 1 1 2 3	1 2 3

Entrada	**Saída**
5 2 3 7 2 5 1 1 15 2 2 2	3 5 2

Neste exemplo, existem 5 carros numa corrida de duas voltas. Os tempos de cada carro em cada volta foram como na tabela a seguir.

	Volta 1	Volta 2	Tempo total
Carro 1	3	7	10
Carro 2	2	5	7
Carro 3	1	1	2
Carro 4	15	2	17
Carro 5	2	2	4

Sendo assim, o vencedor foi o carro 3 (com um tempo total de 2), seguido pelo carro 5 (com um tempo total de 4) e pelo carro 2 (com um tempo total de 7).

Progressões Aritméticas

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

Bob é um aluno do ensino médio que gosta muito de matemática. Na última aula ele aprendeu o que são *Progressões Aritméticas (PAs)* e ficou fascinado por elas. Pelo que Bob entendeu, Progressões Aritméticas são sequências de números nas quais a diferença entre dois elementos consecutivos é sempre igual a uma constante r, chamada de razão da PA.

Um exemplo de Progressão Aritmética de razão 2 é $-1, 1, 3, 5$. Além disso, toda sequência com um ou dois elementos é sempre uma Progressão Aritmética. Por outro lado, $5, 6, 8, 9, 10$ não é uma PA porque a diferença entre elementos consecutivos não é constante: a diferença entre os dois primeiros elementos é $6-5 = 1$, enquanto a diferença entre o terceiro e o segundo elementos é $8 - 6 = 2$.

Bob percebeu que qualquer sequência, mesmo que a mesma não seja uma Progressão Aritmética, pode ser quebrada em sequências menores que são PAs. Por exemplo, vimos que a sequência $5, 6, 8, 9, 10$ não é uma PA, mas podemos quebrar ela entre o 6 e o 8 para obtermos as sequências $5, 6$ e $8, 9, 10$, que são PAs. Note que não existe como quebrar a sequência em menos partes se quisermos ter apenas PAs no fim do procedimento.

Bob é fascinado por programação mas ainda não sabe programar muito bem, e por isso pediu sua ajuda: ele não está conseguindo descobrir como quebrar sequências muito grandes de um jeito eficiente; por isso, pediu que você escrevesse um programa para, dada uma sequência qualquer, imprimir o número mínimo de partes em que precisamos quebrar a sequência para termos apenas Progressões Aritméticas no término do processo. Caso a sequência original já seja uma PA, podemos terminar o processo com uma única parte, e portanto a resposta para esse caso é 1.

Entrada

A primeira linha da entrada é composta por um inteiro N, o número de elementos da sequência. Na segunda linha existem N inteiros a_i, os elementos da sequência.

Saída

A saída deve conter uma única linha, indicando o número mínimo de partes em que Bob precisa quebrar a sequência original para que ele termine apenas com PAs.

Restrições

- $1 \leq N \leq 10^5$
- $-10^5 \leq a_i \leq 10^5$

Exemplos

Entrada	**Saída**
5 1 2 3 4 5	1

Entrada	**Saída**
7 -2 0 2 3 3 4 6	3

É fácil verificar que a sequência $-2, 0, 2, 3, 3, 4, 6$ (do exemplo acima) não é uma PA, pois $2-0 \neq 3-2$. Verificando manualmente, você pode constatar que não é possível particionar a sequência em duas de tal forma que ambas as partes sejam PAs. Entretanto, existe uma maneira de particionar a sequência em 3 PAs: $\boxed{-2, 0, 2}$ $\boxed{3, 3}$ $\boxed{4, 6}$. Portanto, temos que a resposta para este exemplo é 3.

Entrada	**Saída**
4 -2 0 3 6	2

A sequência $-2, 0, 3, 6$ (do exemplo acima) pode ser particionada de várias formas. As únicas maneiras que resultam em PAs são as seguintes:

- Com 4 partes temos 1 possibilidade:
  $\boxed{-2}$ $\boxed{0}$ $\boxed{3}$ $\boxed{6}$
- Com 3 partes temos 3 possibilidades:
  $\boxed{-2, 0}$ $\boxed{3}$ $\boxed{6}$
  $\boxed{-2}$ $\boxed{0, 3}$ $\boxed{6}$
  $\boxed{-2}$ $\boxed{0}$ $\boxed{3, 6}$
- Com 2 partes temos 2 possibilidades:
  $\boxed{-2, 0}$ $\boxed{3, 6}$
  $\boxed{-2}$ $\boxed{0, 3, 6}$

Pulo do Sapo

Nome do arquivo fonte: `pulosapo.c`, `pulosapo.cpp`, `pulosapo.pas`, `pulosapo.java`, *ou* `pulosapo.py`

Sebastião Bueno Coelho, apelidado de SBC pelos familiares e amigos, passou as férias de janeiro de 2011 no sítio de seus avós. Durante sua estadia, uma das atividades prediletas do SBC era nadar no rio que havia no fundo da casa onde morava.

Uma das características do rio que mais impressionava SBC era um belo caminho, feito inteiramente com pedras brancas. Há muito tempo, o avô de SBC notara que os habitantes do sítio atravessavam o rio com grande frequência e, por isso, construiu um caminho no rio com pedras posicionadas em linha reta; ao fazê-lo, tomou muito cuidado para que o espaçamento das pedras fosse de exatamente um metro.

Hoje em dia, a única utilidade do caminho é servir de diversão para os sapos que vivem no rio, que pulam de uma pedra a outra agitadamente. Um certo dia, enquanto descansava e nadava nas águas, SBC assistiu atentamente às acrobacias dos bichos e notou que cada sapo sempre pulava (zero, uma ou mais vezes) uma quantidade fixa de metros.

SBC sabe que você participa da OBI todos os anos e, chegando na escola, resolveu desafiar-te com o seguinte problema: Dado o número de pedras no rio, o número de sapos, a pedra inicial sobre a qual cada sapo está (cada pedra é identificada por sua posição na sequência de pedras) e a distância que cada sapo pula, determinar as posições onde pode existir um sapo depois que SBC chega no rio.

Entrada

A primeira linha da entrada contém dois inteiros N e M representando o número de pedras no rio e o número de sapos, respectivamente. Cada uma das M linhas seguintes possui dois inteiros P e D representando a posição inicial de um sapo e a distância fixa de pulo, respectivamente.

Saída

A saída contém N linhas. A i-ésima linha indica a possibilidade ou não de ter um sapo na i-ésima pedra. Para as pedras que podem ter um sapo você deve imprimir 1, e para as pedras que com certeza não podem ter nenhum sapo você deve imprimir 0.

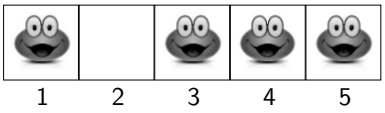
Restrições

- $1 \leq N, M \leq 100$
- Para cada sapo, $1 \leq P, D \leq N$

Exemplos

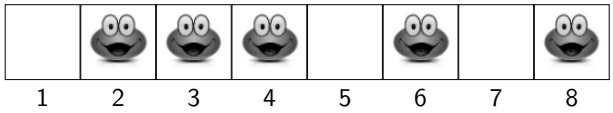
Entrada	**Saída**
5 2 3 2 4 4	1 0 1 1 1

Neste exemplo, SBC indicou a existência de 5 pedras no rio e 2 sapos. Os sapos estavam inicialmente nas pedras 3 e 4. SBC também lhe disse que o primeiro sapo da entrada sempre pula 2 metros, e o segundo sempre pula 4 metros. A figura a seguir ilustra as possíveis pedras que podem ser ocupadas pelos sapos quando eles começam a pular.



Entrada	**Saída**
8 3 3 3 2 2 6 2	0 1 1 1 0 1 0 1

Neste exemplo, SBC indicou a existência de 8 pedras no rio e 3 sapos. Os sapos estavam inicialmente nas pedras 3, 2 e 6. SBC também lhe disse que o primeiro sapo da entrada sempre pula 3 metros, o segundo e terceiro sempre pulam 2 metros. Dessa forma, o primeiro sapo pode estar nas pedras 3 ou 6; o segundo sapo pode estar nas pedras $2, 4, 6$ ou 8; e o terceiro sapo pode estar nas pedras $6, 4, 2$ e 8. A figura a seguir ilustra as possíveis pedras que podem ser ocupadas pelos sapos quando eles começam a pular.



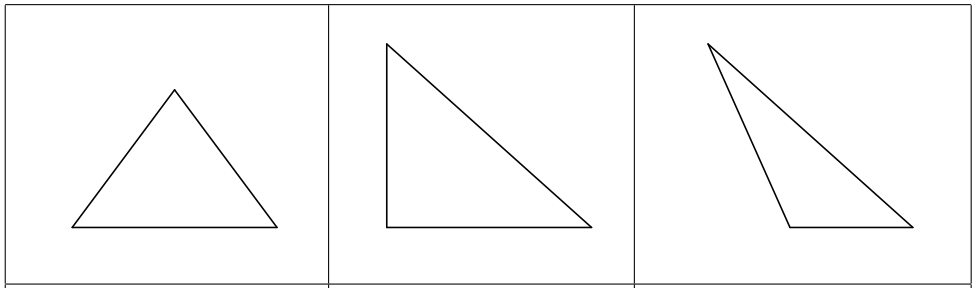
Triângulos

Nome do arquivo fonte: `triangulos.c`, `triangulos.cpp`, `triangulos.pas`, `triangulos.java`, *ou* `triangulos.py`

Caio estava brincando de construir triângulos com palitos de diferentes tamanhos. Ele fazia isso juntando as pontas de três palitos sobre uma mesa. Ele notou que podia agrupar os triângulos formados em três grupos:

- Triângulos *acutângulos*, que são aqueles em que todos os ângulos internos medem menos de 90°;
- Triângulos *retângulos*, que são aqueles que possuem um ângulo interno que mede exatamente 90°;
- Triângulos *obtusângulos*, que são aqueles que possuem um ângulo interno que mede mais de 90°.

Ele também percebeu que nem sempre é possível formar um triângulo com três palitos.



Um triângulo acutângulo	Um triângulo retângulo	Um triângulo obtusângulo

Sua tarefa é, dados os comprimentos A, B e C de três palitos, dizer se é possível formar um triângulo com esses palitos e, em caso afirmativo, dizer a qual grupo o triângulo formado pertence.

Entrada

A entrada consiste de uma única linha, contendo três inteiros A, B e C separados por espaço.

Saída

Imprima uma linha contendo apenas uma letra minúscula:

- 'n' se não for possível formar um triângulo;
- 'a' se o triângulo formado for *acutângulo*;
- 'r' se o triângulo formado for *retângulo*;
- 'o' se o triângulo formado for *obtusângulo*.

Restrições

- $1 \leq A \leq 10^4$
- $1 \leq B \leq 10^4$
- $1 \leq C \leq 10^4$

Exemplos

Entrada	**Saída**
`1 1 1`	`a`

Entrada	**Saída**
`1 2 1`	`n`

Entrada	**Saída**
`5 4 3`	`r`

Entrada	**Saída**
`6 3 4`	`o`