Competidor(a): _____

Número de inscrição: _____–_____ *(opcional)*

OBI2018

Caderno de Tarefas

Modalidade **Programação** • **Nível 1** • Fase **Nacional**

25 de agosto de 2018

A PROVA TEM DURAÇÃO DE **4** HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

v1.0

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 8 páginas (não contando a folha de rosto), numeradas de 1 a 8. 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; em particular, seu programa não deve escrever frases como "Digite o dado de entrada:" ou similares.
- 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 necessariamente 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; soluções na linguagem Python 2 devem ser arquivos com sufixo *.py2*; soluções na linguagem Python 3 devem ser arquivos com sufixo *.py3*; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*.
- Na linguagem Java, **não** use o comando *package*, e note que o nome de sua classe principal deve usar somente letras minúsculas (o mesmo nome do arquivo indicado nas tarefas).
- Para tarefas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada tarefa.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou pen-drive, 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,input,print,write*
  - em Javascript: *scanf, printf*
- Procure resolver a tarefa 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.

Batalha

Nome do arquivo: "`batalha.x`", *onde* `x` *deve ser* `c|cpp|pas|java|js|py2|py3`

Todo mundo está jogando um novo game de realidade aumentada no celular, com batalhas entre monstrinhos! Nas primeiras fases do jogo as batalhas são bem simples, mas ainda assim bastante divertidas. Dois jogadores vão escolher um monstrinho cada, na sua coleção de monstrinhos. Cada monstrinho tem um tipo de ataque e um tipo de defesa, que são identificados por números naturais.

A regra da batalha, que consiste em cada monstrinho usar seu respectivo ataque ao mesmo tempo, é que se o número da defesa de um monstrinho é igual ao número do ataque do seu oponente, então ele não sofre nenhum dano; caso contrário, se o número da defesa dele é diferente do ataque do oponente, então ele sofre dano total e desmaia!

Por exemplo, o monstrinho do primeiro jogador tem o ataque 21 e a defesa 7; enquanto que o monstrinho do segundo jogador tem o ataque 7 e a defesa 12. Nesse caso, o primeiro jogador vence, pois não desmaiou, enquanto que o segundo jogador desmaiou.

Assim, o resultado da batalha, que seu programa deve determinar, pode ser:

- Jogador 1 vence: se o jogador 1 não desmaia e o jogador 2 desmaia;
- Jogador 2 vence: se o jogador 2 não desmaia e o jogador 1 desmaia;
- Empate: em qualquer caso contrário; quer dizer, os dois desmaiam, ou nenhum desmaia.

Entrada

A primeira linha da entrada contém um inteiro A_1 indicando o ataque do primeiro jogador. A segunda linha contém um inteiro D_1 indicando a defesa do primeiro jogador. A terceira linha contém um inteiro A_2 representando o ataque do segundo jogador. A quarta, e última linha, contém um inteiro D_2 representando a defesa do segundo jogador.

Saída

Imprima uma linha contendo um inteiro, 1 ou 2, indicando qual jogador ganhou a batalha. Se a batalha resultou em empate, imprima -1.

Restrições

- $1 \leq A_1 \leq 100$
- $1 \leq D_1 \leq 100$
- $1 \leq A_2 \leq 100$
- $1 \leq D_2 \leq 100$

Exemplo de entrada 1	**Exemplo de saída 1**
21 7 7 12	1

Exemplo de entrada 2	**Exemplo de saída 2**
2 5 71 18	-1

Exemplo de entrada 3	**Exemplo de saída 3**
14 8 8 14	-1

Exemplo de entrada 4	**Exemplo de saída 4**
1 19 32 1	2

Troca

Nome do arquivo: "`troca.x`", *onde* `x` *deve ser* `c|cpp|pas|java|js|py2|py3`

Um cientista especializado em bioinformática está estudando formas de simular alguns fenômenos, que ocorrem dentro das células, relacionados ao funcionamento de proteínas. Parece muito complicado, não? Só que o problema computacional básico que ele precisa resolver eficientemente é fácil de entender. Existe uma sequência de N cartas, indexadas de 1 a N, e cada carta contém dois números impressos, um de cada lado. As cartas são colocadas na mesa, na sequência, com um dos lados virado para cima. Dados dois inteiros i e j, com $i \leq j$, a operação $troca(i, j)$ consiste em virar todas as cartas da posição i até a posição j, inclusive. Por exemplo, considere a sequência de cartas abaixo.

virado para cima	31	2	45	3	8	1	32	10	4	27	12	7	7	9	63	47
virado para baixo	1	12	6	4	97	2	87	10	3	9	55	56	11	90	3	8

A operação de $troca(5, 11)$ resultaria na seguinte sequência de cartas:

virado para cima	31	2	45	3	97	2	87	10	3	9	55	7	7	9	63	47
virado para baixo	1	12	6	4	8	1	32	10	4	27	12	56	11	90	3	8

O problema do cientista é que a sequência de cartas pode ser muito grande e podem ser feitas muitas operações de troca. Ele precisa saber a sequência dos números que estarão virados para cima ao final de todas as operações. Você pode ajudá-lo?

Entrada

A primeira linha da entrada contém dois inteiros N e T, indicando respectivamente a quantidade de cartas e a quantidade de operações de troca. A segunda linha contém N inteiros, indicando os números virados para cima inicialmente. A terceira linha contém N números, indicando os virados para baixo inicialmente. As T linhas seguintes contém, cada uma, dois inteiros I e J, indicando os limites de uma operação de troca.

Saída

Imprima uma linha contendo N inteiros representando os números que estarão virados para cima após todas as operações.

Restrições

- $1 \leq N \leq 10^5$
- $1 \leq T \leq 10^5$
- $1 \leq I \leq J \leq N$
- O valor dos elementos dos vetores está entre 0 e 10^9, inclusive.

Informações sobre a pontuação

- Para um conjunto de casos de teste valendo 10 pontos, $I = J$ para todas as operações;
- Para um conjunto de casos de teste valendo 20 pontos, $N \leq 10^4$ e $T \leq 10^4$.

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

`16 1`
`31 2 45 3 8 1 32 10 4 27 12 7 7 9 63 47`
`1 12 6 4 97 2 87 10 3 9 55 56 11 90 3 8`
`5 11`
	`31 2 45 3 97 2 87 10 3 9 55 7 7 9 63 47`

Exemplo de entrada 2	**Exemplo de saída 2**
10 5 7 88 23 44 1 67 73 2 9 11 4 55 1 1 3 74 82 9 8 37 1 3 5 10 2 6 5 9 1 7	7 55 1 44 1 67 82 2 9 37

Recibo de Compra

Nome do arquivo: "`recibo.x`", *onde* `x` *deve ser* `c|cpp|pas|java|js|py2|py3`

Flavinho acabou de chegar do supermercado com K produtos na sacola, mas perdeu o recibo da compra. Ele está tentando lembrar dos preços de cada um dos produtos e precisa da sua ajuda. Por enquanto ele consegue se lembrar das seguintes informações:

- O valor total da compra foi de R reais;
- Os valores dos produtos eram números inteiros distintos.

Por exemplo, se $R = 12$ e $K = 3$, temos as seguintes possíveis combinações de preços para os três produtos: $\{1, 2, 9\}$, $\{1, 3, 8\}$, $\{1, 4, 7\}$, $\{1, 5, 6\}$, $\{2, 3, 7\}$, $\{2, 4, 6\}$ ou $\{3, 4, 5\}$.

Seu programa deve computar a quantidade de possíveis combinações de preços para os K produtos.

Entrada

A primeira linha da entrada contém dois inteiros R e K, indicando respectivamente o valor total do recibo e o número de produtos comprados.

Saída

Imprima uma linha contendo um inteiro representando a quantidade de possíveis combinações de preços para os K produtos.

Restrições

- $1 \leq R \leq 100$
- $1 \leq K \leq 20$

Informações sobre a pontuação

- Para um conjunto de casos de teste valendo 10 pontos, $R \leq 6$
- Para um conjunto de casos de teste valendo 20 pontos, $R \leq 16$

Exemplo de entrada 1	**Exemplo de saída 1**
`12 3`	`7`

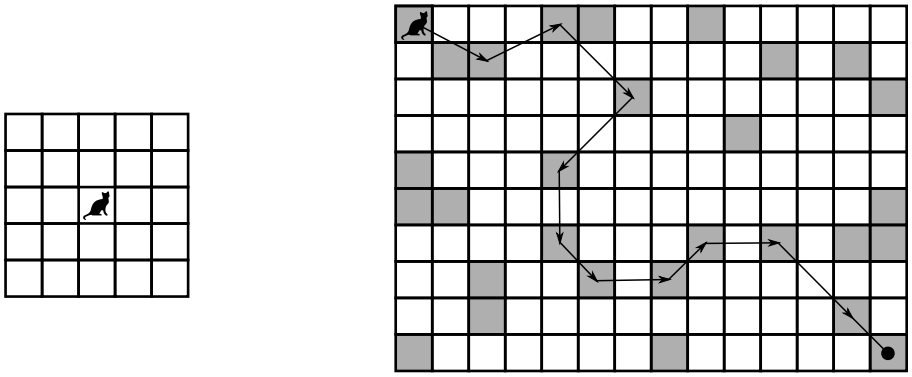
Exemplo de entrada 2	**Exemplo de saída 2**
`10 5`	`0`

Exemplo de entrada 3	**Exemplo de saída 3**
`100 8`	`116263`

Pulo do Gato

Nome do arquivo: "`gato.x`", *onde* `x` *deve ser* `c|cpp|pas|java|js|py2|py3`

O gato Obinho gosta de brincar no pátio do colégio, que tem a forma de um quadriculado de L linhas por C colunas de lajotas, que podem ser brancas ou pretas. Obinho está na lajota inicial, na linha 1, coluna 1 (canto superior esquerdo), e quer ir pulando até a lajota final, na linha L, coluna C (canto inferior direito). Mas ele só gosta de pular de uma lajota preta para outra lajota preta, nunca pisando numa lajota branca. Além disso, ele não consegue pular muito longe. A parte esquerda da figura mostra as lajotas que o Obinho pode alcançar com um pulo: qualquer lajota dentro do quadrado de 5×5 lajotas centrado na posição atual dele.



Obinho quer chegar na lajota final com o número mínimo de pulos possível. Por exemplo, na parte direita da figura, para $L = 10$ e $C = 14$, o menor número de pulos possível é 11. Seu programa deve computar o número mínimo de pulos para o Obinho chegar na lajota final!

Entrada

A primeira linha da entrada contém dois inteiros L e C, representando o número de linhas e colunas do pátio. As L linhas seguintes contêm, cada uma, C inteiros indicando a cor das lajotas: 1 para preta; 0 para branca.

Saída

Imprima uma linha contendo o número mínimo de pulos que o gato Obinho precisa dar para ir da lajota inicial até a lajota final. Se não for possível pular até a lajota final, imprima -1.

Restrições

- $1 \leq L, C \leq 500$;
- As lajotas inicial e final são sempre pretas.

Informações sobre a pontuação

- Para um conjunto de casos de teste valendo 10 pontos, $L = 1$ e todas as lajotas são pretas;
- Para um conjunto de casos de teste valendo 10 pontos, $L = 1$;
- Para um conjunto de casos de teste valendo 10 pontos, $L > 1$, $C > 1$ e todas as lajotas são pretas;
- Para um conjunto de casos de teste valendo 20 pontos, $L \leq 100$ e $C \leq 100$.

Exemplo de entrada 1	**Exemplo de saída 1**
10 14 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1	11

Exemplo de entrada 2	**Exemplo de saída 2**
10 14 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1	-1

Exemplo de entrada 3	**Exemplo de saída 3**
1 12 1 1 1 1 1 1 1 1 1 1 1 1	6