Competidor(a): _____

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

Este Caderno de Tarefas não pode ser levado para casa após a prova. Após a prova entregue este Caderno de Tarefas para seu professor guardar. Os professores poderão devolver os Cadernos de Tarefas aos competidores após o término do período de aplicação das provas (1 a 3 de Junho de 2023).

Olimpíada Brasileira de Informática

OBI2023

Caderno de Tarefas

Modalidade Programação • Nível Sênior • Fase 1

1 a 3 de Junho de 2023

A PROVA TEM DURAÇÃO DE 2 horas

Promoção:

Sociedade Brasileira de Computação

Apoio:

FUNDAÇÃO BEHRING

Coordenação:

INSTITUTO DE COMPUTAÇÃO UNICAMP

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 10 páginas (não contando a folha de rosto), numeradas de 1 a 10. 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 Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; soluções na linguagem Python 3 devem ser arquivos com sufixo *.py*; 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).
- Você pode submeter até 50 soluções para cada tarefa. A pontuação total de cada tarefa é a melhor pontuação entre todas as submissões. Se a tarefa tem sub-tarefas, para cada sub-tarefa é considerada a melhor pontuação entre todas as submissões.
- 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 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.

Contas a pagar

Nome do arquivo: `contas.c`, `contas.cpp`, `contas.java`, `contas.js` *ou* `contas.py`

Vô João está aposentado, tem boa saúde, mas a vida não está fácil. Todo mês é um sufoco para conseguir pagar as contas! Ainda bem que ele é muito amigo dos donos das lojas do bairro, e eles permitem que ele fique devendo.

Depois de pagar aluguel, conta de luz, conta de água, conta do telefone celular e conta do mercado, Vô João ainda tem que pagar as contas do Açougue, da Farmácia e da Padaria.

Dados o valor que Vô João tem disponível e o valor das contas do Açougue, Farmácia e Padaria, escreva um programa para determinar quantas contas, entre as três que ainda não foram pagas, Vô João consegue pagar.

Entrada

A entrada contém quatro linhas. A primeira linha contém um inteiro V, o valor que Vô João tem disponível para pagar as contas. A segunda linha contém um inteiro A, o valor da conta do Açougue. A terceira linha contém um inteiro F, o valor da conta da Farmácia. A quarta linha contém um inteiro P, o valor da conta da Padaria.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o maior número de contas que Vô João consegue pagar.

Restrições

- $0 \leq V \leq 2\ 000$
- $1 \leq A \leq 1\ 000$
- $1 \leq F \leq 1\ 000$
- $1 \leq P \leq 1\ 000$

Informações sobre a pontuação

- A tarefa vale 100 pontos.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
100 30 40 30	3

Exemplo de entrada 2	**Exemplo de saída 2**
50 82 99 51	0

Exemplo de entrada 3	**Exemplo de saída 3**
200 180 100 120	1

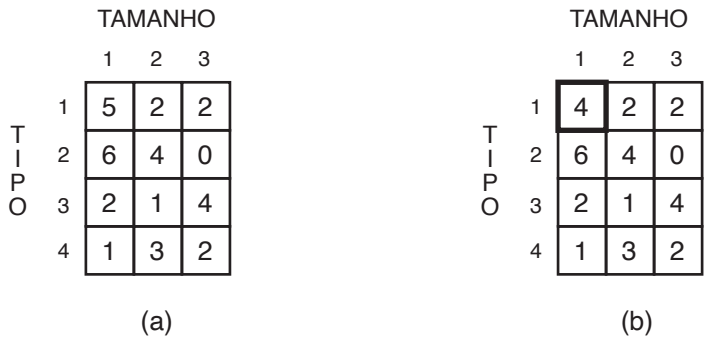
Exemplo de entrada 4	**Exemplo de saída 4**
200 100 180 90	2

Estoque

Nome do arquivo: `estoque.c`, `estoque.cpp`, `estoque.java`, `estoque.js` *ou* `estoque.py`

Você foi contratado(a) para desenvolver um programa de controle de estoque, para uma loja de roupas que está iniciando vendas online. A loja mantém um estoque de roupas, em que cada peça de roupa é identificada por um tipo (por exemplo camisa, calça, saia, vestido, ...) e um tamanho (por exemplo bebê, infantil, pequeno, médio, ...).

O estoque da loja pode ser visto como uma tabela em que cada linha representa um tipo de roupa e cada coluna representa um tamanho, como mostrado na figura (a) abaixo. Na figura, tipos de roupa são representados por números de 1 a 4 e tamanhos são representados por números de 1 a 3.



Assim, a tabela da figura (a) mostra que o estoque da peça de roupa de tipo 1 e tamanho 1 é 5 unidades, e o estoque da peça de roupa de tipo 4 e tamanho 2 é 3 unidades.

Quando uma peça de roupa é vendida, o estoque deve ser atualizado. Por exemplo, se uma peça de roupa de tipo 1 e tamanho 1 for vendida, o estoque atualizado é mostrado na figura (b). Se o estoque para um tipo e tamanho de peça de roupa tem valor zero, peças de roupa desse tipo e tamanho não podem ser vendidas (por exemplo a peça de roupa de tipo 2 e tamanho 3 na figura). Ou seja, a venda não é efetivada.

Dados o estoque inicial e a lista de pedidos de clientes, escreva um programa para determinar quantas peças de roupa são efetivamente vendidas no total. Cada pedido se refere a uma única peça de roupa. As vendas são processadas sequencialmente, na ordem em que os pedidos foram feitos. Se uma venda não é possível por falta de estoque, o pedido correspondente é ignorado.

Entrada

A primeira linha da entrada contém dois números inteiros M e N, indicando respectivamente o número de tipos e o número de tamanhos de peças de roupa no estoque. Tipos são identificados por inteiros de 1 a M e tamanhos são identificados por inteiros de 1 a N. Cada uma das M linhas seguintes contém N inteiros $X_{i,j}$, indicando a quantidade de roupas do tipo i e tamanho j, para $1 \leq i \leq M$ e $1 \leq j \leq N$. A seguir a entrada contém uma linha com um número inteiro P, o número de pedidos recebidos pela loja. Cada uma das P linhas seguintes contém dois inteiros I e J representando respectivamente o tipo e o tamanho da peça de roupa de um pedido. Os pedidos são dados na ordem em que foram feitos.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o número total de peças de roupas efetivamente vendidas.

Restrições

- $1 \leq M \leq 500$
- $1 \leq N \leq 500$
- $0 \leq X_{i,j} \leq 10$ para $1 \leq i \leq M$ e $1 \leq j \leq N$
- $1 \leq P \leq 1\,000$
- $1 \leq I \leq M$
- $1 \leq J \leq N$

Informações sobre a pontuação

- A tarefa vale 100 pontos.
- Para um conjunto de casos de testes valendo 19 pontos, há apenas um tipo de roupa, ou seja $M = 1$.
- Para um conjunto de casos de testes valendo 17 pontos, há apenas um tamanho de roupa, ou seja $N = 1$.
- para um conjunto de casos de testes valendo os 64 pontos restantes, nenhuma restrição adicional.

Exemplos

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

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

Subsequência

Nome do arquivo: `subsequencia.c`, `subsequencia.cpp`, `subsequencia.java`, `subsequencia.js` *ou* `subsequencia.py`

Você foi contratado pela Agência Extra-Espacial Brasileira, que procura indícios de vida extra-terrestre.

Um dos telescópios da Agência, para o espectro ultravioleta, gera uma sequência de valores inteiros positivos que devem ser analisados continuamente. Dadas duas sequências S_A e S_B, sua primeira missão é determinar se S_B é uma *subsequência* de S_A.

Uma *subsequência* de uma dada sequência S é um conjunto de elementos de S que não são necessariamente adjacentes mas que mantêm a mesma ordem em que aparecem em S. Por exemplo, [2], [1, 4], [1, 2, 4] e [1, 2, 3, 4] são subsequências de [1, 2, 3, 4], mas [4, 3], [3, 4, 1] e [1, 3, 5] não são.

Entrada

A primeira linha contém dois inteiros A e B, o número de elementos das sequências. A segunda linha contém A inteiros X_i, os números da sequência S_A. A seguir a entrada contém B inteiros Y_i, os números da sequência S_B.

Saída

Seu programa deve produzir uma única linha, contendo um único caractere, que deve ser a letra maiúscula '`S`' se S_B é uma subsequência da S_A ou a letra maiúscula '`N`' caso contrário.

Restrições

- $1 \leq A \leq 10^5$
- $1 \leq B \leq A$
- $-10^9 \leq X_i \leq 10^9$ para $1 \leq i \leq A$
- $-10^9 \leq Y_i \leq 10^9$ para $1 \leq i \leq B$

Informações sobre a pontuação

- A tarefa vale 100 pontos.
- Para um conjunto de casos de testes valendo 11 pontos, $A = B = 2$.
- Para um conjunto de casos de testes valendo outros 33 pontos, os números aparecem no máximo uma vez em cada sequência, $A \leq 100$, $1 \leq X_i \leq 100$ e $1 \leq Y_i \leq 100$.
- Para um conjunto de casos de testes valendo outros 56 pontos, nenhuma restrição adicional.

Exemplos

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

`5 3`
`1 2 3 4 5`
`2 3 5`
	`S`

Exemplo de entrada 2	**Exemplo de saída 2**
5 4 8 17 8 21 23 8 8 21 22	N

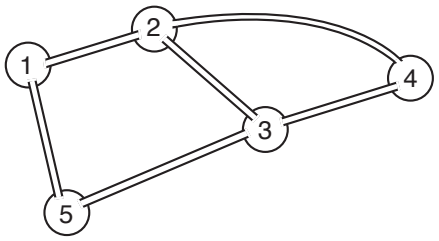
Sr. Toupeira

Nome do arquivo: `toupeira.c`, `toupeira.cpp`, `toupeira.java`, `toupeira.js` *ou* `toupeira.py`

Senhor Toupeira é o prefeito de Morro Seco e ao longo dos anos mandou construir muitos túneis embaixo da terra, conectando salões de convivência que ele também mandou construir, para alegria de sua comunidade de toupeiras. Cada túnel conecta exatamente dois salões de convivência distintos e não há dois túneis conectando o mesmo par de salões. Túneis podem ser usados em ambas direções, ou seja, o túnel que conecta os salões A e B pode ser usado para ir da A para B ou de B para A. Salões de convivência possuem identificadores únicos.

Senhor Toupeira agora quer incentivar que as toupeiras de Morro Seco façam caminhadas, para melhorar a saúde da comunidade. Para isso preparou um caderno com várias sugestões de passeio pelos túneis e salões de convivência, em que cada sugestão de passeio é descrita como uma sequência de salões de convivência, que devem ser visitados estritamente na ordem dada. No entanto, Senhor Toupeira foi alertado de que algumas das sugestões de passeio estão incorretas, pois não são possíveis.

A figura abaixo mostra um exemplo de salões de convivência e túneis, em que salões têm identificadores 1, 2, 3, 4 e 5.



Um passeio composto pela sequência de salões {5, 3, 4, 3, 2} é possível. Mas o passeio composto pela sequência de salões {2, 3, 5, 4} não é possível, pois não existe túnel entre os salões 5 e 4.

Dados o mapa de túneis e salões de convivência, e uma lista de sugestões de passeio, escreva um programa que determine quantas sugestões de passeio são possíveis.

Entrada

A primeira linha da entrada contém dois inteiros S, e T, indicando respectivamente o número de salões de convivência e o número de túneis. Salões são identificados por inteiros de 1 a S. Cada uma das T linhas seguintes descreve um túnel e contém um par de inteiros X e Y, que indicam que o túnel conecta os salões X e Y. A próxima linha da entrada contém um inteiro P que indica o número de sugestões de passeio. Cada uma das P linhas seguintes descreve uma sugestão de passeio e inicia com um inteiro N que indica o número de salões do passeio, seguido de N inteiros C_i, indicando a sequência de salões do passeio.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o número de sugestões de passeio que são possíveis.

Restrições

- $2 \leq S \leq 1\,000$

- $1 \leq T \leq S(S-1)/2$
- $1 \leq X \leq S$
- $1 \leq Y \leq S$
- $1 \leq P \leq 1\,000$
- $1 \leq N \leq 1\,000$
- $1 \leq C_i \leq S$, para $1 \leq i \leq N$
- $C_i \neq C_{i+1}$, para $1 \leq i \leq N-1$, ou seja, salões consecutivos em uma sugestão de passeio são distintos.

Informações sobre a pontuação

- A tarefa vale 100 pontos.
- Para um conjunto de casos de testes valendo 49 pontos, $S \leq 100, P \leq 100$ e $N \leq 100$.
- Para um conjunto de casos de testes valendo outros 17 pontos, $T = S - 1$ e existe um túnel entre os salões i e $i + 1$, para $1 \leq i \leq S - 1$.
- Para um conjunto de casos de testes valendo outros 34 pontos, nenhuma restrição adicional.

Exemplos

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

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

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