Competidor(a): ____

Número de inscrição: ______________–______ *(opcional)*

Olimpíada Brasileira de Informática

OBI2023

Caderno de Tarefas

Modalidade Programação • Nível Júnior • Fase 3

30 de setembro de 2023

A PROVA TEM DURAÇÃO DE 3 HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

Conselho Nacional de Desenvolvimento Científico e Tecnológico

Coordenação:

INSTITUTO DE COMPUTAÇÃO UNICAMP

v1.0

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 14 páginas (não contando a folha de rosto), numeradas de 1 a 14. 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 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).
- 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.

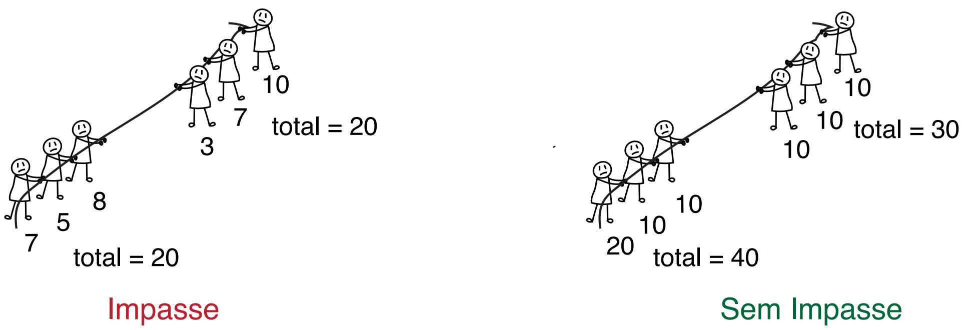
Cabo de guerra

Autor: Mateus Bezrutchka

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

Na mais nova conferência da OBI (*Organização de Brincadeiras Infantis*), os participantes estão explorando brincadeiras de competição em times, como o clássico jogo do *Cabo de Guerra*. A versão de *Cabo de Guerra* jogada na OBI é sempre disputada entre dois times, cada um com três integrantes. Cada time se reúne em uma das pontas de uma corda e então os times começam a puxar a corda em direções opostas. O time vencedor é aquele que consegue puxar a corda em sua direção, fazendo com que o time oponente a solte.

No banco de dados da OBI, cada jogador possui uma *força*, representada por um número inteiro entre 1 e 100, e a *força total* de um time é definida como a soma das forças de seus integrantes. Por exemplo, um time cujos integrantes têm forças 7, 5 e 8 tem força total $7+5+8 = 20$. Os estudos da OBI concluíram que, se os dois times tiverem forças totais diferentes, a partida de *Cabo de Guerra* sempre acaba em menos de 5 minutos. Porém, se os dois times tiverem exatamente a mesma força total, o jogo entra em um *impasse* e os participantes continuam puxando a corda por 12 horas!



A coordenação da OBI está organizando partidas de *Cabo de Guerra*, mas precisa evitar qualquer impasse. Por isso, eles pediram a sua ajuda: dadas as forças dos seis participantes de uma partida, determine se existe alguma chance de impasse – ou seja, se existe um modo de dividir os seis participantes em dois times tal que cada participante pertença a exatamente um dos dois times, cada time tenha três integrantes, e os dois times tenham a mesma força total.

Entrada

A única linha da entrada contém seis números inteiros positivos X_i ($1 \leq i \leq 6$), indicando as forças dos seis participantes.

Saída

Seu programa deverá imprimir uma única linha contendo um único caractere: 'S' (sem aspas), se existe um modo de dividir os times que causa impasse, ou 'N' (sem aspas), se não existe modo de causar impasse.

Restrições

- $1 \leq X_i \leq 100$ para $1 \leq i \leq 6$

Informações sobre a pontuação

A tarefa vale 100 pontos.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
`7 5 3 8 10 7`	`S`

Explicação do exemplo 1: Esse exemplo é representado pela figura (a). Podemos montar um dos times com os participantes de forças 7, 5 e 8 e o outro time com os participantes de forças 3, 10 e 7. Desse modo, ambos os times têm força total 20.

Exemplo de entrada 2	**Exemplo de saída 2**
`10 10 10 20 10 10`	`N`

Explicação do exemplo 2: Esse exemplo é representado pela figura (b). Em qualquer divisão válida dos participantes em dois times, um dos times terá força total $20 + 10 + 10 = 40$ e o outro terá força total $10 + 10 + 10 = 30$, logo as forças totais não podem ser iguais.

Exemplo de entrada 3	**Exemplo de saída 3**
`5 5 6 6 8 8`	`S`

Exemplo de entrada 4	**Exemplo de saída 4**
`4 8 12 24 20 16`	`N`

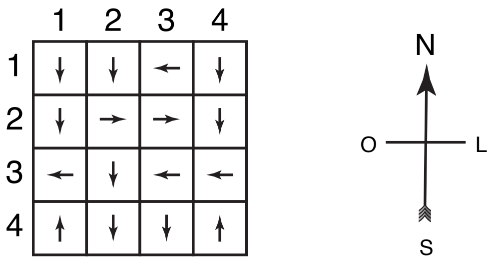
Tesouro

Autor: Mateus Bezrutchka

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

Eduardo é um estudante do departamento de Arqueologia da Unicamp que recentemente encontrou um mapa da Quadradônia, uma ilha histórica cujos habitantes adoravam quadrados. A ilha pode ser representada como um quadrado de lado M dividido em M^2 quadrados unitários com coordenadas de $(1, 1)$ a (M, M), com a primeira coordenada indicando a linha (numerada de 1 a M de Norte a Sul) e a segunda indicando a coluna (numerada de 1 a M de Oeste a Leste). Cada um desses M^2 quadrados representa uma *região* da ilha.

A Quadradônia está atualmente inabitada: seus habitantes fugiram quando a areia que cobre a ilha se tornou movediça! A areia de cada uma das regiões se move em um sentido específico, que pode ser para o Norte, Sul, Leste ou Oeste. Uma pessoa que pisa em uma das regiões da ilha é imediatamente arrastada pela areia, sendo movida da sua região atual para uma das regiões vizinhas (de acordo com o sentido de movimento da areia na região atual), com a velocidade de uma região por minuto. A figura abaixo ilustra uma representação da Quadradônia para $M = 4$, com o sentido de movimento da areia indicado em cada região.



(a)

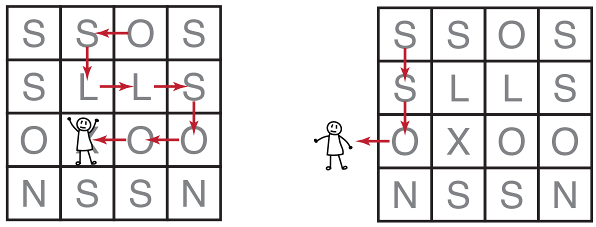
O mapa encontrado por Eduardo indica a posição de um tesouro escondido em uma das regiões da Quadradônia, que está marcada com um X. Para todas as outras regiões, o mapa indica, com um caractere, o sentido no qual a areia se move na região (N para Norte, S para Sul, L para Leste, O para Oeste). A figura (b) indica o mapa do tesouro correspondente à figura (a), assumindo que o tesouro está escondido na região $(3, 2)$.

	1	2	3	4
1	S	S	O	S
2	S	L	L	S
3	O	X	O	O
4	N	S	S	N

(b)

Interessado no tesouro, Eduardo escolheu uma das regiões da ilha para sobrevoar e saltar de paraquedas, pousando na região escolhida. Ele percebeu que a escolha dessa região inicial é muito importante: dependendo da região inicial, ele pode demorar mais ou menos tempo para chegar à região que contém o tesouro. Além disso, é possível que ele eventualmente seja arrastado para fora das bordas da ilha, caindo no mar, ou ainda que seja arrastado na areia movediça para sempre, sem nunca chegar ao tesouro.

Considere os seguintes percursos possíveis:



(c) (d)

A figura (c) indica o percurso de Eduardo se ele saltar na região $(1, 3)$; nesse caso, ele chega ao tesouro em 7 minutos. A figura (d) indica o percurso para região inicial $(1, 1)$; nesse caso, ele é jogado ao mar após chegar à região $(3, 1)$.

Eduardo pediu sua ajuda: dado o mapa do tesouro da Quadradônia e as coordenadas da região inicial dele (ou seja, a região na qual ele vai pousar), determine o tempo em minutos que ele levará para chegar à região que contém o tesouro. Caso Eduardo nunca chegue ao tesouro, indique se ele cairá no mar ou ficará preso na areia movediça para sempre.

Entrada

A primeira linha da entrada é composta por um inteiro M, indicando que a Quadradônia é composta por $M \times M$ regiões.

As próximas M linhas descrevem o mapa do tesouro e possuem M caracteres cada. R_{ij}, o j-ésimo caractere da i-ésima dessas linhas, representa a região (i, j) e pode ser:

- 'X', se a região (i, j) contém o tesouro;
- 'N', 'S', 'L' ou 'O', de acordo com o sentido de movimento da areia na região.

A última linha da entrada contém dois inteiros A e B, as coordenadas da região inicial de Eduardo (ou seja, a região na qual ele vai pousar).

Saída

Seu programa deverá produzir uma única linha contendo um único inteiro:

- Se Eduardo eventualmente chega à região que contém o tesouro, imprima o tempo em minutos que ele leva para chegar ao tesouro.
- Se Eduardo continua sendo arrastado na areia para sempre, sem nunca chegar ao tesouro, imprima 0.
- Se Eduardo sai das bordas da ilha e é jogado ao mar, imprima -1.

Restrições

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

- R_{ij} = 'N', 'S', 'L', 'O' ou 'X' para todos $1 \leq i, j \leq M$
- Existe exatamente um par (i, j) para o qual R_{ij} = 'X'
- $1 \leq A, B \leq M$
- $M_{AB} \neq$ 'X' (ou seja, a região inicial e a região do tesouro são distintas)

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (30 pontos):** $M = 2$. *(Veja os exemplos 3 e 4.)*
- **Subtarefa 2 (40 pontos):** $M \leq 50$ e é garantido que Eduardo consegue chegar à região que contém o tesouro.
- **Subtarefa 3 (30 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

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

`4`
`SSOS`
`SLLS`
`OXOO`
`NSSN`
`1 3`
	`7`

Explicação do exemplo 1: Esse é o exemplo da figura (c).

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

`4`
`SSOS`
`SLLS`
`OXOO`
`NSSN`
`1 1`
	`-1`

Explicação do exemplo 2: Esse é o exemplo da figura (d).

Exemplo de entrada 3	**Exemplo de saída 3**

`2`
`SO`
`XO`
`2 2`
	`1`

Exemplo de entrada 4	Exemplo de saída 4

`2`
`LO`
`XO`
`1 2`
	`0`

Explicação do exemplo 4: Eduardo ficará para sempre sendo arrastado entre as regiões $(1, 2)$, cuja areia se move para o Oeste, e $(1, 1)$, cuja areia se move para o Leste.

Metrônibus

Autores: André Amaral de Sousa e Mateus Bezrutchka

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

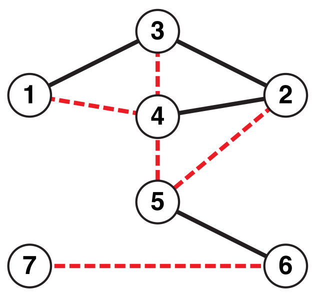
Ênia, a atual rainha da Nlogônia, é fascinada por meios de transporte, especialmente metrô e ônibus. Como consequência, a Nlogônia possui a melhor rede de transporte público do mundo inteiro, a *Metrônibus*.

Existem N estações espalhadas por todo o reino, que são conectadas por dois *sistemas de transporte*: ônibus, que utiliza rodovias, e metrô, que utiliza trilhos. Assim, no Metrônibus existe transporte direto entre duas estações se elas estão ligadas por uma rodovia ou por um trilho. Toda ligação, independente de ser rodovia ou trilho, pode ser utilizada em ambas as direções. Existe no máximo uma ligação entre cada par de estações (ou seja, um par de estações não pode estar ligado por ambos um trilho e uma rodovia).

Para utilizar o Metrônibus, o usuário precisa comprar passagens. Ênia estabeleceu as seguintes regras para estimular o transporte público:

- Toda passagem tem um preço fixo P, independente da estação e do sistema (metrô ou ônibus).
- A passagem só deve ser paga quando o usuário entrar no sistema de transporte.
- O usuário só pode entrar em um sistema por vez (ou seja, para entrar no sistema de metrô ele precisa sair do sistema de ônibus, e vice-versa).
- Após pagar o preço P para entrar em um sistema, o usuário pode se locomover o quanto desejar usando ligações daquele sistema, contanto que ele não saia do sistema. Caso o usuário troque de sistema, ele deverá pagar por uma nova passagem para retornar ao sistema original.

Por exemplo, considere a seguinte rede do Metrônibus:



Existem 7 estações e 9 ligações no total, das quais 4 são trilhos do metrô e as outras 5 são rodovias para ônibus. Suponha que o preço da passagem seja $P = 5$ e que o usuário deseja fazer uma viagem da estação 1 para a estação 7. Temos várias opções de trajeto, como por exemplo (as setas indicam a ordem em que visitamos as estações):

- **Viagem 1:** $1 \to 3 \to 2 \to 4 \to 5 \to 6 \to 7$
  - A ligação entre as estações 1 e 3 usa o metrô, então precisamos entrar no sistema de metrô pela primeira vez, pagando $P = 5$.

  - O trecho $1 \to 3 \to 2 \to 4$ é todo percorrido de metrô, logo não precisamos pagar de novo.
  - $4 \to 5$ usa ônibus, então precisamos pagar $P = 5$ para trocar de metrô para ônibus.
  - $5 \to 6$ usa o metrô, então pagamos mais $P = 5$ para reentrar no sistema de metrô.
  - $6 \to 7$ usa ônibus, então precisamos pagar mais $P = 5$ para entrar no sistema de ônibus novamente e chegar à estação 7.
  - No total, a viagem custa $5 + 5 + 5 + 5 = 20$.

- **Viagem 2:** $1 \to 4 \to 5 \to 6 \to 7$
  - A ligação entre as estações 1 e 4 usa ônibus, que estamos utilizando pela primeira vez, então é necessário pagar $P = 5$.
  - $4 \to 5$ também usa ônibus, ou seja, continuamos no mesmo sistema de transporte e não é necessário pagar outra passagem.
  - $5 \to 6$ usa o metrô, que vamos utilizar pela primeira vez, logo é necessário pagar $P = 5$.
  - $6 \to 7$ usa o sistema de ônibus, do qual saímos e precisamos pagar $P = 5$ para reentrar.
  - No total, a viagem custa $5 + 5 + 5 = 15$.

Existem outras viagens que começam na estação 1 e terminam na estação 7. Porém, o menor custo total entre todas essas viagens é 15 (correspondente à viagem 2).

Ênia deseja saber o menor custo total para ir da estação A para a estação B, e para isso te contratou. Responda corretamente, ou ela te expulsará da Nlogônia!

Entrada

A primeira linha da entrada contém quatro inteiros N, K_1, K_2, e P, representando o número de estações, o número de ligações (trilhos) no sistema de metrô, o número de ligações (rodovias) no sistema de ônibus, e o preço da passagem, respectivamente.

As próximas K_1 linhas contém a descrição das ligações do metrô. A i-ésima dessas linhas terá dois inteiros V_i e U_i, representando que há um trilho entre as estações V_i e U_i.

As próximas K_2 linhas contém a descrição das ligações de ônibus. A j-ésima dessas linhas terá dois inteiros X_j e Y_j, representando que há uma rodovia entre as estações X_j e Y_j.

A última linha da entrada contém dois números inteiros, A e B, representando a estação inicial e a estação final da viagem requisitada pela rainha.

Saída

Seu programa deverá imprimir uma única linha contendo um único inteiro, o menor custo total de uma viagem que começa na estação A e termina na estação B.

Se não for possível fazer uma viagem de A a B usando o Metrônibus, imprima -1.

Restrições

- $2 \leq N \leq 100\ 000$
- $1 \leq K_1 + K_2 \leq 100\ 000$
- $1 \leq P \leq 1\ 000$
- $1 \leq V_i, U_i \leq N$ para todo $1 \leq i \leq K_1$
- $V_i \neq U_i$ para todo $1 \leq i \leq K_1$

- $1 \leq X_j, Y_j \leq N$ para todo $1 \leq j \leq K_2$
- $X_j \neq Y_j$ para todo $1 \leq j \leq K_2$
- Cada par de estações é ligado por no máximo uma ligação no total (ou seja, considerando ambos os sistemas)
- $1 \leq A, B \leq N$
- $A \neq B$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (31 pontos):** $K_2 = 0$, ou seja, todas as ligações são do sistema de metrô. *(Veja o exemplo 2.)*
- **Subtarefa 2 (32 pontos):** $N \leq 100$.
- **Subtarefa 3 (37 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

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

`7 4 5 5`
`1 3`
`3 2`
`4 2`
`6 5`
`4 1`
`4 3`
`4 5`
`2 5`
`6 7`
`1 7`
	`15`

Explicação do exemplo 1: Este exemplo corresponde ao exemplo mostrado no enunciado.

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

`6 4 0 42`
`1 4`
`6 1`
`3 2`
`4 6`
`3 5`
	`-1`

Explicação do exemplo 2: A estação 5 não possui ligação com nenhuma outra estação, portanto é impossível chegar nela a partir da estação 3.

Oficina Mecânica

Autor: André Amaral de Sousa

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

O dono de uma oficina mecânica te contratou para criar um software de gerenciamento. Com esse software, ele espera não só conseguir controlar os estoques dos produtos utilizados na oficina, mas também auxiliar no trabalho de seus funcionários e tornar a oficina mais eficiente.

Como um(a) excelente programador(a), você já implementou quase todo o software. Falta apenas uma funcionalidade: distribuir os carros a serem consertados entre os mecânicos de modo a garantir alta satisfação dos clientes. Após analisar os dados coletados na oficina sobre clientes passados, você descobriu que o fator mais importante para a satisfação de um cliente é seu *tempo de espera* para ser atendido, ou seja, o tempo em que ele fica esperando até o seu carro começar a ser consertado. Os clientes não se importam com o tempo de conserto de seus respectivos carros, pois eles sabem que os mecânicos fazem um excelente trabalho.

O tempo de conserto varia de acordo com o carro e com o mecânico: cada carro possui um *tempo base* T_i, já determinado durante o orçamento, que mede a quantidade de trabalho necessária para consertar o carro, e cada mecânico possui um *fator de trabalho* F_j, indicando quanto tempo ele demora para completar uma unidade de trabalho. Assim, o tempo necessário para o mecânico j consertar o carro i é $T_i \cdot F_j$. Vale ressaltar que cada mecânico só trabalha em um único carro por vez, e portanto precisa finalizar o conserto de um carro antes de começar a consertar o próximo.

Por exemplo, suponha que tenhamos quatro carros cujos tempos base são 5, 10, 15 e 20, e dois mecânicos cujos fatores de trabalho são 1 e 2. Uma distribuição possível é:

- O mecânico com fator 1 conserta os carros com tempos 20 e 5, nessa ordem.
- O mecânico com fator 2 conserta os carros com tempos 10 e 15, nessa ordem.

Essa distribuição possui os seguintes tempos de espera:

- O carro com tempo base 20 é atendido imediatamente, portanto seu tempo de espera é 0.
- O carro com tempo base 5 precisa esperar o carro com tempo base 20 ser consertado, portanto seu tempo de espera é $20 \cdot 1 = 20$.
- O carro com tempo base 10 é atendido imediatamente, portanto seu tempo de espera é 0.
- O carro com tempo base 15 precisa esperar o carro com tempo base 10 ser consertado, portanto seu tempo de espera é $10 \cdot 2 = 20$.

A soma total dos tempos de espera na distribuição acima é $0+20+0+20 = 40$. Porém, existe uma distribuição (que é ótima) onde a soma total dos tempos de espera é 20 – você consegue encontrá-la?

Dados o número N de carros esperando para serem atendidos, a quantidade M de mecânicos, o tempo base T_i de cada carro, e o fator de trabalho F_j de cada mecânico, determine a menor soma total dos tempos de espera dos carros, que corresponde à melhor forma de distribuir todos os carros entre os mecânicos.

Entrada

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

A segunda linha contém N números inteiros. O i-ésimo desses números será T_i, representando o tempo base do i-ésimo carro.

A terceira e última linha da entrada contém M números inteiros. O j-ésimo desses números será F_j, representando o fator de trabalho do j-ésimo mecânico.

Saída

Seu programa deverá imprimir uma única linha contendo um único inteiro, a menor soma total dos tempos de espera em uma distribuição de todos os carros entre os mecânicos.

Restrições

- $1 \leq N \leq 100\,000$
- $1 \leq M \leq 100\,000$
- $1 \leq T_i \leq 10\,000$ para todo $1 \leq i \leq N$
- $1 \leq F_j \leq 10\,000$ para todo $1 \leq j \leq M$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (14 pontos):** $N \leq 1000$ e $M = 1$.
- **Subtarefa 2 (10 pontos):** $T_i = 1$ para todo $1 \leq i \leq N$ e $F_j = 1$ para todo $1 \leq j \leq M$.
- **Subtarefa 3 (16 pontos):** $F_j = 1$ para todo $1 \leq j \leq M$.
- **Subtarefa 4 (36 pontos):** $N \leq 1000$ e $M \leq 1000$.
- **Subtarefa 5 (24 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

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

`4 2`
`5 10 15 20`
`1 2`
	`20`

Explicação do exemplo 1: Este exemplo corresponde ao exemplo mostrado no enunciado.

Exemplo de entrada 2	Exemplo de saída 2

`4 1`
`1 1 1 1`
`3`
	`18`

Explicação do exemplo 2: Como há apenas um mecânico, ele consertará todos os carros. E como todos os tempos base são iguais, a ordem dos carros não afeta o resultado. Logo, podemos só consertar os carros em ordem. Os tempos de espera dos quatro carros são então, respectivamente, 0, 3, 6, e 9, para um total de $0 + 3 + 6 + 9 = 18$.

Exemplo de entrada 3	Exemplo de saída 3

`4 1`
`15 5 10 20`
`3`
	`150`