Competidor(a): _____

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

OBI2019

Caderno de Tarefas

Modalidade **Programação** • **Nível Sênior** • Fase **Estadual**

14 de agosto de 2019

A PROVA TEM DURAÇÃO DE **2** HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 7 páginas (não contando a folha de rosto), numeradas de 1 a 7. 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 tarefas 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.py*; soluções na linguagem Python 3 devem ser arquivos com sufixo *_py3.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.

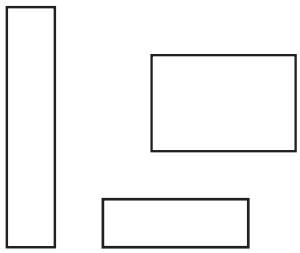
Robo marciano

Nome do arquivo: `robo.c`, `robo.cpp`, `robo.pas`, `robo.java`, `robo.js`, `robo_py2.py` *ou* `robo_py3.py`

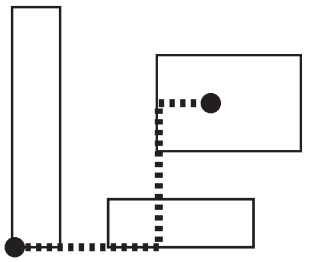
Uma empresa de turismo aeroespacial está se preparando para a exploração comercial de Marte. Ela implantou uma base de operações no planeta, onde conduz experimentos que visam garantir a segurança de futuros turistas.

A base em Marte é composta por um conjunto de áreas retangulares cobertas por um teto protetor contra a radiação solar. As áreas retangulares têm lados paralelos aos eixos Norte-Sul e Leste-Oeste. Vários robôs, controlados por comandos enviados desde o Centro de Operações da empresa, na Terra, deslocam-se constantemente pela base para acessar materiais e equipamentos.

Os robôs podem deslocar-se apenas nas quatro direções cardeais (norte, sul, leste e oeste), mas podem transitar tanto em áreas cobertas como não cobertas. Em particular, um robô pode entrar e sair de uma área coberta por qualquer ponto da borda dessa área. Para preservar a vida útil dos robôs, é importante que eles se mantenham o máximo possível protegidos da intensa radiação solar, ou seja, que eles transitem preferencialmente nas áreas cobertas da base.



Áreas protegidas



Exemplo de trajeto do robô

Dadas as descrições das áreas cobertas, a posição atual de um robô e a posição para a qual este robô deve se deslocar, sua tarefa é determinar a menor distância que o robô deve percorrer fora das áreas cobertas para chegar à posição de destino.

Entrada

A primeira linha da entrada contém quatro inteiros X_i, Y_i, X_f e Y_f indicando, respectivamente, a posição inicial do robô, (X_i, Y_i) e a posição final do robô, (X_f, Y_f).

A segunda linha contém um único inteiro N, indicando o número de áreas cobertas. Cada uma das N linhas seguintes contém quatro inteiros X_1, Y_1, X_2 e Y_2 indicando uma região retangular coberta, tal que (X_1, Y_1) e (X_2, Y_2) são vértices opostos do retângulo de lados paralelos aos eixos. Duas áreas cobertas podem ter regiões comuns.

Saída

Seu programa deve produzir uma única linha, com um único número inteiro, a menor distância que o robô deve percorrer em áreas não cobertas para ir da posição inicial à posição final do robô.

Restrições

- $0 \leq N \leq 1000$
- $0 \leq X_i, Y_i, X_f, Y_f \leq 10^6$
- $0 \leq X_1 \leq X_2 \leq 10^6$ e $0 \leq Y_1 \leq Y_2 \leq 10^6$

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 30 pontos, $X_1, Y_1, X_2, Y_2, X_i, X_f, Y_i, Y_f \leq 10$ e $N \leq 5$.

- Para um conjunto de casos de testes valendo outros 50 pontos, $X_1, Y_1, X_2, Y_2, X_i, X_f, Y_i, Y_f \leq 1000$ e $N \leq 100$.

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

Exemplo de entrada 2	**Exemplo de saída 2**
2 5 5 0 1 0 0 1 5	5

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

Matriz super-legal

Nome do arquivo: `matriz.c`, `matriz.cpp`, `matriz.pas`, `matriz.java`, `matriz.js`, `matriz_py2.py` *ou* `matriz_py3.py`

Denotando por $A_{i,j}$ o elemento na i-ésima linha e j-ésima coluna da matriz A, dizemos que uma matriz é "legal" se a condição

$$A_{1,1} + A_{lin,col} \leq A_{1,col} + A_{lin,1}$$

é verdadeira para todo $lin > 1$ e $col > 1$.

Adicionalmente, dizemos que a matriz é "super-legal" se cada uma de suas submatrizes com pelo menos duas linhas e duas colunas é legal. Lembre que uma submatriz S de uma matriz $M_{L \times C}$ é uma matriz que inclui todos os elementos $M_{i,j}$ tais que $l_1 \leq i \leq l_2$ e $c_1 \leq j \leq c_2$, para $1 \leq l_1 \leq l_2 \leq L$ e $1 \leq c_1 \leq c_2 \leq C$.

A sua tarefa é, dada uma matriz A, determinar a maior quantidade de elementos de uma submatriz super-legal da matriz A.

Entrada

A primeira linha contém dois inteiros L e C indicando respectivamente o número de linhas e o número de colunas da matriz. Cada uma das L linhas seguintes contém C inteiros X_i representando os elementos da matriz.

Saída

Seu programa deve produzir uma única linha, com apenas um número inteiro, a maior quantidade de elementos de uma submatriz super-legal da matriz da entrada, ou zero no caso de não existir uma submatriz super-legal.

Restrições

- $2 \leq L, C \leq 1000$
- $-10^6 \leq X_i \leq 10^6$

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 10 pontos, $L, C \leq 3$.
- Para um conjunto de casos de testes valendo outros 50 pontos, $L, C \leq 300$.

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

`3 3`
`1 4 10`
`5 2 6`
`11 1 3`
	`9`

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

`3 3`
`1 3 1`
`2 1 2`
`1 1 1`
	`4`

Exemplo de entrada 3	**Exemplo de saída 3**
5 6 1 1 4 0 3 3 4 4 9 7 11 13 -3 -1 4 2 8 11 1 5 9 5 9 10 4 8 10 5 8 8	15

Supermercado

Nome do arquivo: `super.c`, `super.cpp`, `super.pas`, `super.java`, `super.js`, `super_py2.py` *ou* `super_py3.py`

Maria está participando de um programa de intercâmbio no reino da Nlogônia. Ela está gostando muito da experiência, e decidiu fazer um churrasco para suas novas amigas da escola. Como não tem muito dinheiro, Maria vai fazer uma pesquisa para comprar carne no supermercado mais barato que encontrar.

No entanto ela está um pouco confusa para saber qual supermercado tem o menor preço. O dinheiro na Nlogônia é o Bit, abreviado por B$, mas não é esse o problema. O problema é que o costume na Nlogônia é informar o preço de uma maneira diferente do que Maria está acostumada. Os preços são anunciados como "X Bits por Y gramas do produto".

Por exemplo o preço de um dado produto é anunciado como sendo B$ 24,00 por 250 gramas em um supermercado, B$ 16,00 por 100 gramas em outro supermercado, B$ 19,00 por 120 gramas em outro supermercado, e assim por diante.

Você pode ajudar Maria?

Dados os preços anunciados pelos supermercados no bairro em que Maria mora, determine o menor valor que Maria deve gastar para comprar 1 kilograma (1000 gramas) de carne.

Entrada

A primeira linha contém um número inteiro N, o número de supermercados próximos à casa de Maria. Cada uma das N linhas seguintes indica o preço da carne em um supermercado e contém um número real P e um número inteiro G, indicando que G gramas de carne custam P Bits.

Saída

Seu programa deve produzir uma única linha, com apenas um número real, o menor preço para comprar 1 kilograma de carne. O resultado deve ser escrito com exatamente dois dígitos após o ponto decimal.

Restrições

- $1 \leq N \leq 100$
- $0 < P \leq 1000.00$, representado com dois dígitos após o ponto decimal.
- $1 \leq G \leq 1000$

Exemplo de entrada 1	**Exemplo de saída 1**
3 3.0 100 2.0 100 5.0 100	20.00

Exemplo de entrada 2	**Exemplo de saída 2**
4 100.00 500 190.00 1000 200.00 900 110.00 550	190.00

Exemplo de entrada 3	**Exemplo de saída 3**
5 46.50 794 25.72 130 66.00 800 22.45 110 38.99 453	58.56