OBI2016

Caderno de Tarefas

Modalidade **Programação** • **Nível 2** • Fase **1**

3 de junho de 2016

A PROVA TEM DURAÇÃO DE **5** HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

v1.0

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.
- 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; soluções na linguagem Python2 devem ser arquivos com sufixo *.py2*; soluções na linguagem Python3 devem ser arquivos com sufixo *.py3*; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*. 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 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 Python2 ou Python3: *read,readline,readlines,input,print,write*
  - em Javascript: *scanf, printf*
- 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.

Lâmpadas do hotel

Nome do arquivo: `hotel.c`, `hotel.cpp`, `hotel.pas`, `hotel.java`, `hotel.js`, `hotel.py2` *ou* `hotel.py3`

Você está de volta em seu hotel na Tailândia depois de um dia de mergulhos. O seu quarto tem duas lâmpadas. Vamos chamá-las de A e B. No hotel há dois interruptores, que chamaremos de C_1 e C_2. Ao apertar C_1, a lâmpada A acende se estiver apagada, e apaga se estiver acesa. Se apertar C_2, cada uma das lâmpadas A e a B troca de estado: se estiver apagada, fica acesa e se estiver acesa apaga.

Você chegou no hotel e encontrou as lâmpadas em um determinado estado, como foram deixadas por seu amigo. Vamos chamar o estado inicial da lâmpada A de I_A e o estado inicial da lâmpada B de I_B. Você gostaria de deixar as lâmpadas em uma certa configuração final, que chamaremos de F_A e F_B, respectivamente, apertando os interruptores a menor quantidade de vezes possível. Por exemplo, se as duas lâmpadas começam apagadas, e você quer que apenas a lâmpada A termine acesa, basta apertar o interruptor C_1.

Dados os estados iniciais e desejados das duas lâmpadas (acesa/apagada), determine o número mínimo de vezes que interruptores devem ser apertados.

Entrada

A entrada contém quatro inteiros: I_A, I_B, F_A e F_B, os estados iniciais das lâmpadas A e B e os estados finais desejados das lâmpadas A e B, respectivamente e nessa ordem. Os valores de I_A, I_B, F_A e F_B possíveis são 0, se a lâmpada estiver apagada e 1 caso contrário.

Saída

Seu programa deverá imprimir um único número, o número mínimo de interruptores que devem ser apertados.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 25 pontos, as duas lâmpadas começam sempre apagadas ($I_A = I_B = 0$).

Exemplos

Entrada	**Saída**
`0 0 1 1`	`1`

Entrada	**Saída**
`0 0 0 1`	`2`

Chaves

Nome do arquivo: `chaves.c`, `chaves.cpp`, `chaves.pas`, `chaves.java`, `chaves.js`, `chaves.py2` *ou* `chaves.py3`

Seu amigo Juca está enfrentando problemas com programação. Na linguagem C, algumas partes do código devem ser colocadas entre chaves `"{ }"` e ele frequentemente esquece de colocá-las ou as coloca de forma errada. Porém, como Juca tem dificuldade para entender os erros de compilação, ele nunca sabe exatamente o que procurar. Por isso ele te pediu para fazer um programa que determine se um código está com as chaves balanceadas, ou seja, se é válido. Um código está com as chaves balanceadas se:

- Não há chaves (como por exemplo "`Bom`" ou "`Correto`");
- O código é composto por uma sequência de códigos válidos (como por exemplo "`Bom Correto`" ou "`{}{}`" ou "`{}Correto`"); ou
- O código é formado por um código válido entre chaves (como por exemplo "`{{}}`" ou "`{Bom}`").

O código de Juca é composto por N linhas de até 100 caracteres cada. Pode haver linhas vazias e espaços consecutivos.

Entrada

A primeira linha contém um inteiro N, representando o número de linhas no código. As N linhas seguintes contém até 100 caracteres.

Saída

Seu programa deve produzir uma única linha, contendo uma única letra, `"S"` se o código está com as chaves balanceadas e `"N"`, caso contrário.

Restrições

- $1 \leq N \leq 10^3$.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 50 pontos, $N = 1$ e todos os carateres são `"{"` ou `"}"` (como no terceiro exemplo).

Exemplos

Entrada	**Saída**

6
#include <stdio.h>
 	
int main(void)
{
  printf("Hello World\n");
}
	S

Entrada	**Saída**

5
{I{N{
}F{])
 	
}O}R{
}M}A{T}I{C@!!{onze}!!}
	S

Entrada	**Saída**
1 {{}}{{}}}	N

Entrada	**Saída**
1 {{{3}}}{{{2}}a{{1}}{0}	N

Chuva

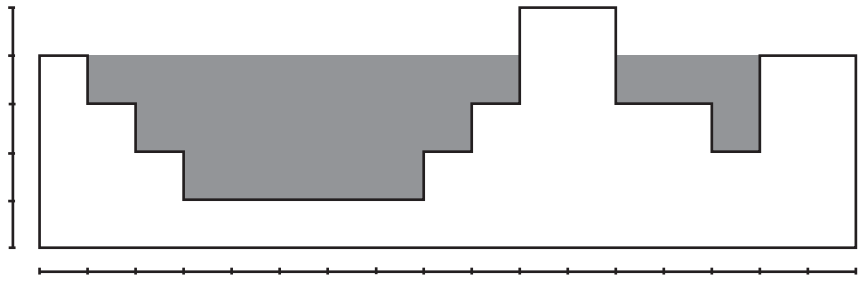
Nome do arquivo: `chuva.c`, `chuva.cpp`, `chuva.pas`, `chuva.java`, `chuva.js`, `chuva.py2` *ou* `chuva.py3`

É período de chuva no Reino Quadrado. Nos últimos anos, o Rei Maior Quadrado (RMQ) ordenou a construção de uma enorme piscina para refrescar seus súditos. A piscina é composta por diversas seções de mesma largura e comprimento, mas podem ter alturas diferentes. A altura de cada seção é um número inteiro em metros.

Durante o período de chuvas fortes, o Rei nem precisa gastar água para encher a piscina - basta deixar que a chuva faça esse trabalho. A chuva cai uniformemente em todas as seções da piscina, enchendo - até que não haja mais capacidade para acumular água.

O Rei o contratou para calcular quantas seções estarão cobertas com água, durante a estação de chuva. Uma seção da piscina pode ser considerada coberta com água se ela possuír água com pelo menos 1m de profundidade.

O caso do exemplo 3 pode ser visto na figura abaixo, que apresenta um corte lateral da piscina. As seções 2 a 10 e 13 a 15 ficarão cobertas de água.



Entrada

A primeira linha contém um inteiro, N, o número de seções da piscina. Seguem N linhas, cada uma com um inteiro H_i, a altura da i-ésima seção, em metros.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o número de seções da piscina cobertos por água.

Restrições

- $1 \leq N \leq 10^5$, $1 \leq H_i \leq 10^9$ $(1 \leq i \leq N)$

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 20 pontos, $N \leq 10^3$.

Exemplos

Entrada	**Saída**
4 2 1 1 2	2

Entrada	**Saída**
6 5 2 6 1 3	2

Entrada	**Saída**
17 4 3 2 1 1 1 1 1 2 3 5 5 3 3 2 4 4	12

Toca do Saci

Nome do arquivo: `toca.c`, `toca.cpp`, `toca.pas`, `toca.java`, `toca.js`, `toca.py2` *ou* `toca.py3`

Depois de muito procurar, Emília finalmente conseguiu encontrar a toca do Saci. A toca tem formato retangular, e é formada por um quadriculado de salas quadradas de mesmo tamanho, com N salas em uma dimensão e M salas na outra dimensão. A figura abaixo mostra um exemplo de mapa da toca, com cinco salas na dimensão horizontal e quatro salas na dimensão vertical. Há uma única entrada, pela sala marcada com o número 3 no mapa. As salas da toca são muito parecidas, para confundir quem tenta encontrar o Saci, e têm portas que comunicam-se apenas com salas vizinhas nas direções horizontal e vertical do mapa.

0	1	1	1	0
0	2	0	1	1
0	0	0	0	1
3	1	1	1	1

Emília entrou na toca seguindo o Saci com o objetivo de pegar o seu chapéu, e só vai devolvê-lo se o Saci prometer não fazer mais diabrites no Sítio. Muito esperta, ela foi deixando estrelinhas coloridas pelas salas que passou (marcadas com o número 1 no mapa), para saber o caminho de volta. Ela pegou o chapéu do Saci enquanto ele dormia, e começou o caminho de volta. Está muito escuro e ela precisa acender um fósforo em cada sala, para ver as estrelinhas que marcam o caminho. No meio do caminho, ela percebeu que seus fósforos estavam acabando e agora está com medo de não ter fósforos suficientes. Ela está na sala marcada com o número 2 no mapa. Você pode ajudá-la?

Dado o mapa da toca, como no exemplo acima, escreva um programa para saber por quantas salas Emília deve passar até encontrar a saída.

Entrada

A primeira linha da entrada contém dois inteiros N e M que indicam respectivamente os números de salas nas duas dimensões da toca. Cada uma das N linhas seguintes contém M números inteiros entre 0 e 3. O valor 0 indica uma sala sem estrelinhas; o valor 1 indica uma sala com estrelinhas deixadas por Emília; o valor 2 indica uma sala com estrelinhas que é a sala onde Emília está; finalmente, o valor 3 indica uma sala com estrelinhas que é a saída. Considere que, durante o trajeto da entrada até a sala marcada com o valor 2, Emília não passou mais do que uma vez por uma mesma sala, e não existe ambiguidade no caminho de volta (em outras palavras, a cada ponto do trajeto de volta, existe apenas uma sala marcada para Emília voltar).

Saída

Seu programa deve imprimir uma única linha, contendo o número de salas que Emília deve passar, seguindo as estrelinhas, até chegar à saída da toca.

Restrições

A entrada obedece às seguintes restrições:

- $1 \leq N \leq 1000$

- $1 \leq M \leq 1000$
- cada sala tem o valor 0, 1, 2 ou 3.
- apenas uma sala tem o valor 2.
- apenas uma sala tem o valor 3.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 20 pontos, Emília está em uma sala que só possui uma sala vizinha com estrelinhas (como no exemplo 1).

Exemplos

Entrada	**Saída**
4 5 0 1 1 1 0 0 2 0 1 1 0 0 0 0 1 3 1 1 1 1	12

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

Entrada	**Saída**
4 5 0 1 2 1 0 0 1 0 1 1 0 0 0 0 1 3 1 1 1 1	10

Sanduíche

Nome do arquivo: `sanduiche.c`, `sanduiche.cpp`, `sanduiche.pas`, `sanduiche.java`, `sanduiche.js`, `sanduiche.py2` *ou* `sanduiche.py3`

Você está na Seletiva para a IOI e depois de um dia cansativo de provas, chegou a hora do jantar. Hoje, trouxeram um sanduíche muito longo cortado em N pedaços de diversos tamanhos diferentes. Você gostaria de comer uma quantidade total de sanduíche de comprimento D, porém há uma regra: para evitar bagunça, você só pode ou pegar uma sequência contínua de pedaços, ou pegar pedaços das extremidades. Você sabe a sequência $C_1, C_2, ... C_N$ dos comprimentos dos pedaços na ordem em que estão posicionados no sanduíche. Agora, para otimizar o seu jantar, quer fazer um programa que com esses dados responda de quantas formas você pode escolher os pedaços do sanduíche que vai comer. Em outras palavras, deve contar quantos pares (i, j), $1 \leq i \leq j \leq N$, existem tais que o somatório $C_i + C_{i+1} + ... + C_j$ seja igual a D e quantos pares (i, j), $1 \leq i < j \leq N$, existem tais que o somatório $C_1 + C_2 + ... + C_i + C_j + C_{j+1} + ... + C_N$ seja igual a D.

Entrada

A primeira linha contém dois inteiros N e D, representando respectivamente o número de pedaços e a quantidade de sanduíche que você quer comer. A segunda linha contém N inteiros $C_1, C_2, ... C_N$, onde C_i é o tamanho do i-ésimo pedaço.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o número de maneiras de comer pedaços de sanduíche com soma D.

Restrições

- $2 \leq N \leq 10^6$.
- $1 \leq D \leq 10^9$.
- $1 \leq C_i \leq 10^3$.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 20 pontos, $N \leq 200$.
- Em um conjunto de casos de teste equivalente a 40 pontos, $N \leq 1000$.

Exemplos

Entrada	**Saída**
`5 10` `1 2 3 4 3`	`3`

Entrada	**Saída**
`5 5` `1 1 1 1 1`	`5`

Entrada	**Saída**
9 618 665 658 248 282 428 562 741 290 457 5	0