OBI2017

Caderno de Tarefas

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

9 de junho de 2017

A PROVA TEM DURAÇÃO DE **2** HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

v1.0

Instruções

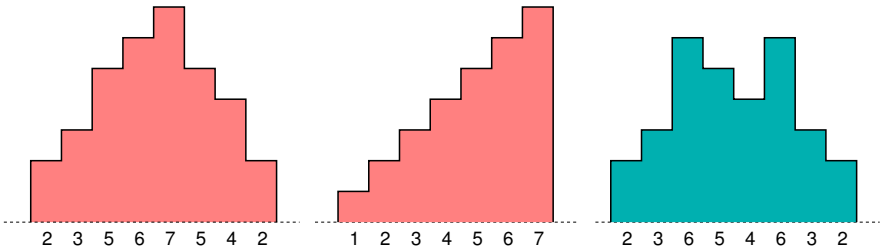
LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 6 páginas (não contando a folha de rosto), numeradas de 1 a 6. 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.

Montanha

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

Um sistema de informações geográficas computadorizado está representando o perfil de uma montanha através de uma sequência de números inteiros, na qual não há dois números consecutivos iguais, como ilustrado na figura abaixo para três montanhas. Os números representam a altura da montanha ao longo de uma certa direção.



O gerente do sistema de informações geográficas pesquisou e encontrou uma maneira de identificar se uma sequência de números inteiros representa uma montanha com mais de um pico, ou com apenas um pico. Ele observou que, como não há números consecutivos iguais, se houver três números consecutivos na sequência, tal que o número do meio é menor do que os outros dois números, então a montanha tem mais de um pico. Caso contrário, a montanha tem apenas um pico. De forma mais rigorosa, se a sequência é $A = [A_1, A_2, A_3, \ldots, A_{N-2}, A_{N-1}, A_N]$, ele quer saber se há uma posição i, para $2 \leq i \leq N-1$, tal que $A_{i-1} > A_i$ e $A_i < A_{i+1}$.

Para ajudar o gerente, seu programa deve determinar, dada a sequência de números inteiros representando a montanha, se ela tem mais de um pico, ou se tem um pico apenas.

Entrada

A primeira linha da entrada contém um inteiro N, representando o tamanho da sequência. A segunda linha contém N inteiros A_i, $1 \leq i \leq N$, representando a sequência de alturas da montanha.

Saída

Seu programa deve imprimir uma linha contendo o caractere "`S`" se há mais de um pico, ou o caractere "`N`" se há apenas um pico.

Restrições

- $3 \leq N \leq 1000$
- $1 \leq A_i \leq 1000$, para $1 \leq i \leq N$

Exemplos

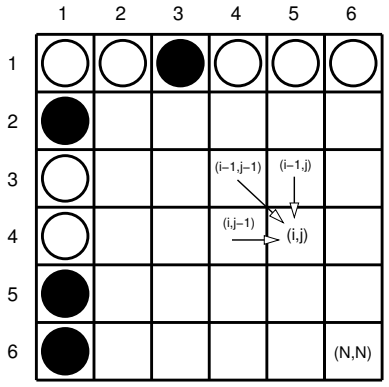
Entrada	**Saída**
8 2 3 5 6 7 5 4 2	N

Entrada	**Saída**
8 2 3 6 5 4 6 3 2	S

Jogo de Tabuleiro

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

Flavinho não se cansa de bolar joguinhos para passar o tempo. Ele diz que é uma boa forma de treinar a memória e a capacidade de resolver problemas. Dessa vez ele inventou uma forma de preencher um tabuleiro de N linhas e N colunas com pedras brancas e pretas. Inicialmente ele coloca, aleatoriamente, pedras brancas e pretas em todas as células da primeira coluna e da primeira linha. A figura ao lado dá um exemplo de tabuleiro com $N = 6$. Ele chama essas pedras iniciais de sementes. Uma vez colocadas as sementes, as demais células do tabuleiro serão preenchidas com uma pedra branca ou preta de acordo com a seguinte regra.



Considere a célula na posição (i, j), para $i > 1$ e $j > 1$. Para saber a cor da pedra nessa célula, Flavinho precisa saber a cor das pedras nas três células $\{(i, j-1), (i-1, j-1), (i-1, j)\}$. A figura também ilustra quais células são usadas para determinar a cor da pedra na célula (i, j). Se houver mais pedras brancas do que pretas nessas três células, a cor da pedra na célula (i, j) será preta. Se houver mais pedras pretas do que brancas, a cor será branca.

Note que, por essa definição, a primeira célula a ser preenchida será a $(2, 2)$, pois será a única vazia para a qual já saberemos a cor das três pedras necessárias. No exemplo da figura, a pedra na célula $(2, 2)$ será da cor preta, pois há duas brancas e uma preta entre as células $\{(2, 1), (1, 1), (1, 2)\}$.

Neste problema, dado N e a cor das sementes, seu programa deve computar a cor da pedra que será colocada na célula (N, N).

Entrada

A primeira linha da entrada contém um inteiro N, representando o número de linhas e colunas do tabuleiro. As N linhas seguintes contêm, cada uma, N inteiros definindo o tabuleiro inicial. Os inteiros na primeira linha e na primeira coluna do tabuleiro serão sempre 0 ou 1, representando uma pedra branca ou preta, respectivamente. Os demais inteiros serão sempre 9, indicando que a célula correspondente está vazia inicialmente.

Saída

Seu programa deve imprimir uma linha contendo um inteiro representando a cor da pedra que será colocada na célula (N, N): 0 se for branca, 1 se for preta.

Restrições

- $2 \leq N \leq 100$

Exemplos

Entrada	**Saída**
2 0 1 1 9	0

Entrada	**Saída**
6 0 0 1 0 0 0 1 9 9 9 9 9 0 9 9 9 9 9 0 9 9 9 9 9 1 9 9 9 9 9 1 9 9 9 9 9	1

Castelos da Nlogônia

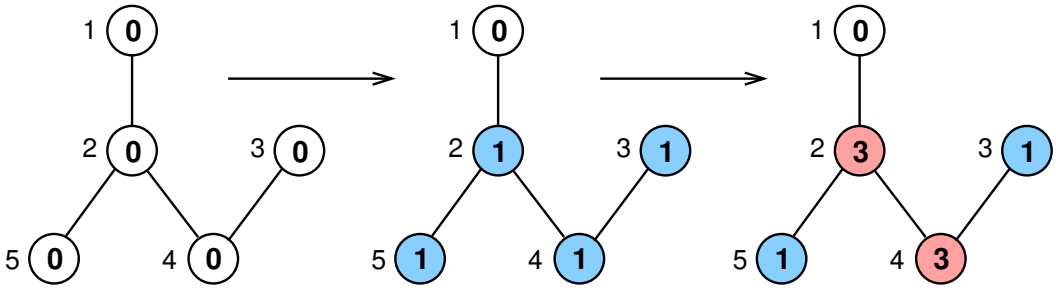
Nome do arquivo: `cores.c`, `cores.cpp`, `cores.pas`, `cores.java`, `cores.js`, `cores.py2` *ou* `cores.py3`

O rei da Nlogônia não consegue decidir de que cor ele vai mandar pintar os castelos do reino. Nos últimos tempos ele tem dado ordens bastante extravagantes do tipo: "pintem todos os castelos no caminho entre o castelo A e o castelo B, inclusive eles, da cor C". Ele pode falar "no" caminho, porque os castelos da Nlogônia estão ligados por estradas entre eles de modo que existe exatamente um caminho entre quaisquer dois castelos, possivelmente passando por outros castelos, sem repetir castelos. De outra forma, sempre é possível ir de qualquer castelo para qualquer outro castelo e por apenas um caminho, sem repetir castelos.

A Nlogônia tem N castelos, identificados por números de 1 a N. A figura ilustra uma sequência de duas operações de colorir sobre cinco castelos, numerados de 1 a 5, com cores identificadas por inteiros de 0 a 3:

- colorir os castelos de 5 até 3 com a cor 1;
- colorir os castelos de 2 até 4 com a cor 3.

Ao final, os castelos de 1 a 5 terão as cores 0, 3, 1, 3 e 1, respectivamente.



Neste problema, considerando que os N castelos na Nlogônia inicialmente estão pintados da cor zero, dados os pares de castelos que estão ligados por uma estrada e uma sequência de M ordens de pintura, seu programa deve imprimir a cor que cada castelo vai ter ao final, depois que todas as ordens de pintura forem executadas em sequência.

Entrada

A primeira linha da entrada contém dois inteiros N e M, respectivamente o número de castelos e o número de ordens de pintura. Os castelos são indexados de 1 a N. As $N-1$ linhas seguintes contêm, cada uma, dois inteiros U e V distintos, indicando que existe uma estrada entre os castelos U e V diretamente. Nas M linhas seguintes, cada linha contém três inteiros P, Q e C, representando uma ordem de pintura entre os castelos P e Q, não necessariamente distintos, com a cor C.

Saída

Seu programa deve imprimir uma única linha, contendo a sequência de cores dos castelos de 1 a N, após todas as M ordens de pinturas terem sido executadas em sequência.

Restrições

- $2 \leq N \leq 100$ e $1 \leq M \leq 100$
- $1 \leq U, V, P, Q \leq N$
- $0 \leq C \leq 100$

Exemplos

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

Entrada	**Saída**
9 4 7 1 1 2 6 2 9 6 2 5 8 6 4 5 3 4 1 4 2 7 8 4 3 4 1 9 2 8	4 8 1 1 2 8 4 4 8