OBI2015

Caderno de Tarefas

Modalidade **Universitária** • Fase **1**

29 de maio de 2015

A PROVA TEM DURAÇÃO DE **5** 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.
- 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 Python devem ser arquivos com sufixo *.py*; 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 Python: *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.

Cobra coral

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

O professor Rui está desenvolvendo um sistema automático para identificar se uma cobra é uma coral verdadeira ou uma falsa coral. A cobra coral verdadeira é venenosa e os anéis coloridos no seu corpo seguem o padrão `...BVBPBVBPBVBP...`, onde `B`,`V` e `P` representam as cores branco, vermelho e preto, respectivamente. Já a falsa coral não é venenosa e os anéis seguem o padrão `...BVPBVPBVPBVP....`

O problema é que os sensores do sistema do professor Rui produzem apenas uma sequência de quatro números representando um pedaço do padrão de cores. Só que ele não sabe qual número representa qual cor. Mas, por exemplo, se a sequência for `5 3 9 3`, podemos dizer com certeza que é uma coral verdadeira, mesmo sem saber qual número representa qual cor! Você deve ajudar o professor Rui e escrever um programa que diga se a coral é verdadeira ou falsa.

Entrada

A entrada consiste de apenas uma linha, contendo quatro números inteiros.

Saída

Seu programa deve imprimir na saída uma linha com a letra "`V`" se a coral for verdadeira ou com a letra "`F`", caso seja falsa.

Restrições

- Os quatro números têm valores entre 1 e 9, inclusive, e a sequência sempre representa uma coral verdadeira, ou uma coral falsa.

Exemplos

Entrada	**Saída**
`5 3 9 3`	`V`

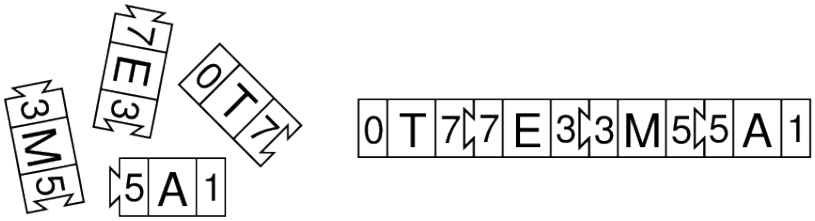
Entrada	**Saída**
`7 1 4 7`	`F`

Entrada	**Saída**
`6 2 6 8`	`V`

Quebra-cabeça

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

Jade precisa da sua ajuda para montar o quebra-cabeças que ela ganhou de presente da sua tia Zoraide! As peças são encaixadas lado a lado e contêm, cada uma, uma letra maiúscula. Quando o quebra-cabeças estiver montado, a sequência de letras revelará uma frase secreta. Cada peça possui, além da letra, dois números: um na parte esquerda e outro na parte direita. Uma peça se encaixa depois de outra, na sequência, quando seu número esquerdo for igual ao número direito da outra peça. O número esquerdo da primeira peça é sempre o 0 (zero) e o número direito da última peça é sempre o 1 (um). Cada número aparece no máximo uma vez na parte esquerda de alguma peça, e no máximo uma vez na parte direita. Sempre é possível encaixar todas as peças e em apenas uma única sequência! Veja um exemplo na figura, com quatro peças formando a palavra "TEMA".



Entrada

A primeira linha da entrada contém um número natural N, indicando o número de peças do quebra-cabeças. As N linhas seguintes contêm, cada uma, a descrição de uma peça na forma E C D, onde: E é o número esquerdo; C é a letra maiúscula; e D é o número direito.

Saída

Seu programa deve escrever uma única linha na saída, contendo a sequência de letras formada quando o quebra-cabeças está montado.

Restrições

- $3 \leq N \leq 100000$; $0 \leq E \leq 200000$; e $0 \leq D \leq 200000$
- Há exatamente uma maneira de montar o quebra-cabeças utilizando todas as peças dadas.

Exemplos

Entrada	**Saída**
4 5 A 1 0 T 7 3 M 5 7 E 3	TEMA

Entrada	**Saída**
3 197452 I 1 0 O 39999 39999 B 197452	OBI

Lápis de cor

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

Roberto tem um conjunto de lápis com 10 tons diferentes de uma mesma cor, numerados de 0 a 9. Numa folha de caderno quadriculado alguns quadrados foram coloridos inicialmente com o tom 0. Roberto precisa determinar, para cada quadrado Q não colorido, qual é a distância dele para o quadrado mais próximo de tom 0. A distância entre dois quadrados é definida com o número mínimo de movimentos ortogonais (para: *esquerda, direita, cima, baixo*) para ir de um quadrado para o outro. O quadrado Q, então, deve ser colorido com o tom cuja numeração corresponde à distância determinada. Se a distância for maior ou igual a 9, o quadrado deve ser colorido com o tom 9. Seu programa deve colorir e imprimir a folha quadriculada dada na entrada.

Entrada

A primeira linha da entrada contém apenas um inteiro N, determinando as dimensões da folha quadriculada, $N \times N$. As N linhas seguintes definem a folha inicialmente. Cada linha contém uma sequência de N caracteres: '`*`' se o quadrado não está colorido, e '`0`' se está colorido com o tom 0.

Saída

Seu programa deve imprimir o tabuleiro totalmente colorido, de acordo com a regra definida acima.

Restrições

- $3 \leq N \leq 1000$.

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 40 pontos, $N \leq 100$

Exemplos

Entrada	**Saída**
8 **000*** ******** *****0** ******** *****000 *******0 0******0 ********	21000123 32111123 43221012 34332111 23321000 12332110 01233210 12344321

Entrada	**Saída**
3 *** *** **0	432 321 210

Divisores

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

Um pesquisador precisa saber o número de divisores de um número dado. Por exemplo, 660 tem 24 divisores: 1, 2, 3, 4, 5, 6, 10, 11, 12, 15, 20, 22, 30, 33, 44, 55, 60, 66, 110, 132, 165, 220, 330, 660.

A fatoração de 660 em fatores primos é: $2^2 \times 3 \times 5 \times 11$. Então o número de divisores é calculado pelo produtos dos expoentes acrescentados de um: $(2+1) \times (1+1) \times (1+1) \times (1+1) = 24$.

Um outro exemplo é o número 50, que tem 6 divisores. De fato, $50 = 2 \times 5^2$, portanto o número de divisores é $(1+1) \times (2+1) = 6$.

Entrada

A entrada consiste de uma linha contendo um inteiro N.

Saída

Seu programa deve escrever uma única linha na saída, contendo um único número inteiro, a quantidade de divisores de N.

Restrições

- $1 \leq N \leq 10^4$

Exemplos

Entrada	**Saída**
660	24

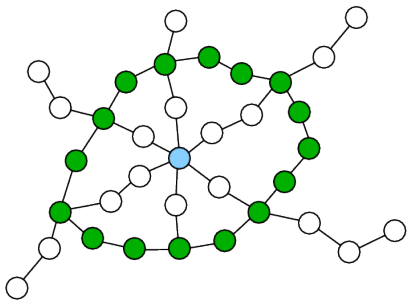
Entrada	**Saída**
50	6

Entrada	**Saída**
9216	33

Metrô

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

Os sistemas de Metrô das cidades da Cosmolândia seguem sempre o mesmo padrão. Há uma estação central e uma linha circular, de modo que a estação central está dentro da área delimitada pela linha circular. A partir da estação central saem pelo menos 5, e no máximo 100, linhas radiais que não se interceptam entre si e que vão pelo menos até alguma estação da linha circular, podendo ir muito além, até os subúrbios mais remotos da cidade. Veja um exemplo na figura abaixo, que possui 6 linhas radiais e onde a linha circular passa por 16 estações.



Cada sistema possui N estações, numeradas de 1 a N. O número de uma estação não tem relação alguma com a sua posição no sistema. O problema é o seguinte. Um engenheiro da Agência Federal de Trânsito da Cosmolândia precisa descobrir rapidamente por quantas estações a linha circular passa. Ele tem apenas a informação sobre as ligações entre pares de estações, sem qualquer identificação das linhas às quais pertencem as duas estações do par. Você pode ajudá-lo?

Entrada

A primeira linha da entrada contém dois inteiros N e M, respectivamente, o número total de estações e de ligações entre pares de estações. As M linhas seguintes contêm, cada uma, dois inteiros P e Q, indicando que a estação de número P está ligada à estação de número Q.

Saída

Seu programa deve imprimir um inteiro, representando o número de estações pelas quais passa a linha circular do sistema.

Restrições

- $6 \leq N \leq 20000$, $M \leq 20100$

Exemplos

Entrada	**Saída**
6 10 1 6 1 3 4 1 3 4 5 4 3 5 2 5 3 2 6 3 2 6	5

Entrada	**Saída**
12 17 7 1 9 3 5 1 7 12 7 3 4 10 3 2 6 8 10 5 7 6 2 11 7 10 12 4 8 9 3 12 1 8 7 4	8