OBI2017

Caderno de Tarefas

Modalidade **Programação** • **Nível 1** • Fase **Nacional**

19 de agosto de 2017

A PROVA TEM DURAÇÃO DE **4** 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 8 páginas (não contando a folha de rosto), numeradas de 1 a 8. 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.

Zip

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

Um jogo de cartas que faz muito sucesso no reino da Nlogônia é chamado zip. Nesse jogo, apenas os valores das cartas são utilizados (ás a rei), os naipes das cartas são ignorados. Para simplificar, neste problema vamos considerar os valores das cartas como inteiros de 1 a 13.

Em cada partida do jogo cada jogador recebe duas cartas. Cada jogador então mostra uma de suas cartas, e os jogadores fazem suas apostas (na Nlogônia só é permitido apostar grãos de feijão). Após as apostas, os jogadores mostram a sua segunda carta.

As regras para determinar quem ganha a partida são simples, baseadas nos valores das cartas de cada jogador:

- se as duas cartas têm o mesmo valor, o jogador recebe como pontuação na partida duas vezes a soma dos valores das cartas.
- se os valores das duas cartas são números consecutivos (por exemplo, 2 e 3, ou 13 e 12), o jogador recebe como pontuação na partida três vezes a soma dos valores das cartas.
- caso contrário, o jogador recebe como pontuação na partida a soma dos valores das cartas.

Ganha a partida o jogador que tiver recebido a maior pontuação. Se houver empate, a aposta acumula para a próxima partida.

Lia e Carolina estão jogando zip, e querem que você escreva um programa para conferir quem ganhou cada partida.

Entrada

A entrada é composta por quatro linhas, cada uma contendo um inteiro. As duas primeiras linhas indicam as cartas de Lia, as duas últimas linhas indicam as cartas de Carolina.

Saída

Seu programa deve produzir uma única linha, contendo o nome da jogadora que venceu a partida. Se houve empate, a linha deve conter a palavra `empate` (em minúsculas).

Restrições

- As cartas que cada jogadora recebe têm o valor entre 1 e 13.

Exemplos

Entrada	**Saída**
3 3 7 4	Lia

Entrada	**Saída**
2 3 11 4	empate

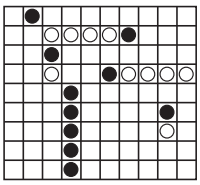
Entrada	**Saída**
5 5 4 3	Carolina

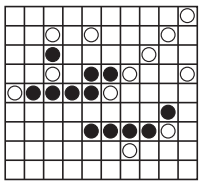
Gomoku

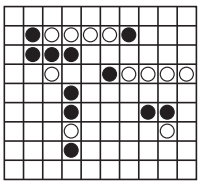
Nome do arquivo: `gomoku.c`, `gomoku.cpp`, `gomoku.pas`, `gomoku.java`, `gomoku.js`, `gomoku.py2` *ou* `gomoku.py3`

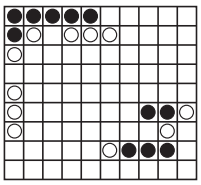
Gomoku é um jogo japonês milenar, jogado em um tabuleiro de 15x15 células e pedras pretas e brancas. Um jogador joga com as pedras brancas, o outro joga com as pedras pretas. O objetivo do jogo, como o nome sugere – go-moku em japonês quer dizer cinco pedras – é colocar cinco pedras da mesma cor consecutivamente ou numa mesma linha, ou numa mesma coluna, ou numa diagonal do tabuleiro.

As figuras abaixo ilustram algumas configurações do jogo (apenas uma parte do tabuleiro é mostrada):


pretas ganham


brancas ganham


ninguém ganha


pretas ganham

Entrada

A entrada é composta por quinze linhas, cada uma contendo quinze inteiros. Cada inteiro representa o conteúdo de uma célula do tabuleiro. O número 1 indica uma pedra preta, o número 2 indica uma pedra branca e o número 0 indica que não há pedra na célula.

Saída

Seu programa deve produzir uma única linha, contendo o inteiro 1 se o jogador com as pedras pretas venceu, o inteiro 2 se o jogador com as pedras brancas venceu, ou o número 0 se ainda não há vencedor na configuração da entrada.

Restrições

- Cada célula tem o valor 0, 1 ou 2.
- Em todos os testes, há apenas um vencedor, ou não há vencedor.

Exemplo

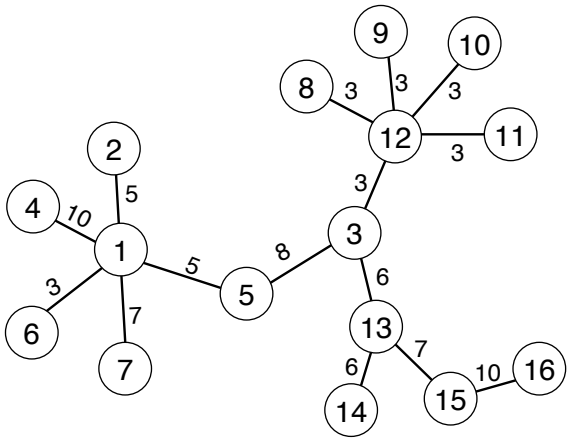
Entrada	**Saída**
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 2 2 2 1 0 0 0 0 0 0 0 0 1 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0	1

Entrada	**Saída**
0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 2 1 1 1 2 0 2 2 1 0 0 0 0 0 0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 2 0 2 2 1 1 0 0 0 0 0 0 0 0 2 1 1 1 1 2 2 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 2 0 0 0 0 0 0 0 0 2 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2

Visita entre cidades

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

A Linearlândia é composta de N cidades, numeradas de 1 até N. Para alguns pares de cidades existe exatamente uma estrada bidirecional entre as duas cidades do par. Os pares de cidades ligados diretamente por uma estrada são escolhidos de forma que sempre é possível ir de qualquer cidade para qualquer outra cidade por um, e somente um, caminho (um caminho é uma sequência de estradas, sem repetição).
Dada a lista de pares de cidades ligados diretamente por estradas, as distâncias entre os pares de cidades, uma cidade origem e uma cidade destino, seu programa deve computar qual a distância entre a cidade de origem e a cidade destino, usando as estradas. Por exemplo, na figura, a distância para ir da cidade 12 para a cidade 7 é 23; a distância da cidade 15 para a cidade 12 é 16; e a distância da cidade 7 para a cidade 15 é 33.



Entrada

A primeira linha da entrada contém três inteiros N, A e B, representando o número de cidades na Linearlândia, a cidade origem e a cidade destino, respectivamente. As cidades são identificadas por inteiros de 1 a N. As $N-1$ linhas seguintes contém, cada uma, três inteiros P, Q e D, indicando que existe uma estrada ligando diretamente as cidades P e Q, com distância D.

Saída

Seu programa deve imprimir uma linha contendo um inteiro representando a distância para ir de A até B pelas estradas de Linearlândia.

Restrições

- $2 \leq N \leq 10000$
- $1 \leq A \leq N$, $1 \leq B \leq N$, $A \neq B$
- $1 \leq P \leq N$, $1 \leq Q \leq N$
- $1 \leq D \leq 100$

Exemplos

Entrada	**Saída**
4 2 4 1 2 10 2 3 11 3 4 12	23

Entrada	**Saída**
16 15 7 3 5 8 12 3 3 5 1 5 2 1 5 4 1 10 6 1 3 7 1 7 12 8 3 12 9 3 12 10 3 12 11 3 3 13 6 13 14 6 15 13 7 15 16 10	33

Bits

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

Duas sequências de N bits são distintas se, para alguma posição i, $1 \leq i \leq N$, o bit na posição i de uma sequência é distinto do bit na posição i da outra sequência. As duas sequências de $N = 10$ bits abaixo são distintas pois, por exemplo, os bits na posição 7, da esquerda para a direita, são distintos:

```
0 1 0 0 0 1 1 0 1 0
1 0 0 1 0 1 0 0 1 0
```

Mas veja que as duas sequências acima, apesar de distintas, têm uma característica em comum: não há três bits 1 consecutivos nelas. Neste problema, dado o número de bits N e um K, seu programa deve computar quantas sequências distintas de N bits existem, nas quais não há K bits 1 consecutivos.

Entrada

A entrada consiste de uma linha contendo os dois inteiros N e K.

Saída

Imprima uma linha contendo um inteiro, representando o número de sequências distintas de N bits, nas quais não há K bits 1 consecutivos. Porque esse número pode ser muito grande, você deve imprimir o resto da divisão dele por $10^9 + 7$.

Restrições

- $1 \leq N \leq 1000$
- $1 \leq K \leq N + 1$

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 20 pontos, $N \leq 20$

Exemplos

Entrada	**Saída**
`4 2`	`8`

Entrada	**Saída**
`10 3`	`504`