6ª MARATONA DE PROGRAMAÇÃO DA FIPP

DIA 20/05 - 14H

CADERNO DE PROBLEMAS

20/05/2010 – 14:00 às 18:00

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 6 problemas, 2 deles estão descritos em inglês.

1. É permitido o uso de material impresso, livros, apostilas e dicionários bem como lápis, caneta ou lapiseira, borracha, régua e papel para rascunho. O acesso à internet é bloqueado durante a realização da prova. A ajuda do ambiente de desenvolvimento como C, C++ ou Java pode ser utilizada.
2. A correção das resoluções dos problemas será automatizada por meio do sistema de submissão eletrônica BOCA, baseada nos resultados obtidos com uma série de execuções dos algoritmos de cada equipe.
3. Siga atentamente as exigências da tarefa quanto ao formato da entrada e saída do seu algoritmo. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
4. Os problemas não estão ordenados, neste caderno, por ordem de dificuldade. Procure resolver primeiro os problemas mais fáceis.
5. Utilize para nomear os seus arquivos-fonte os nomes indicados nos problemas de acordo com a linguagem de programação utilizada.
6. Os problemas devem ser resolvidos utilizando o raciocínio entrada-processamento-saída, ou seja, não é necessário armazenar toda a saída para exibí-la.

Observações:

Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (teclado) e escritos na saída padrão (tela). Utilize as funções padrão para entrada e saída de dados:

- em C: `scanf`, `getchar`, `printf`, `putchar`;
- em C++: as mesmas de C ou os objetos `cout` e `cin`;
- em Java:

```
Scanner sc = new Scanner(); int i = sc.nextInt();
String s = sc.next(); float f = sc.nextFloat();
System.out.println(); System.out.printf("%d",i);
```

Faculdade de Informática de Presidente Prudente

http://www.unoeste.br/fipp/maratona

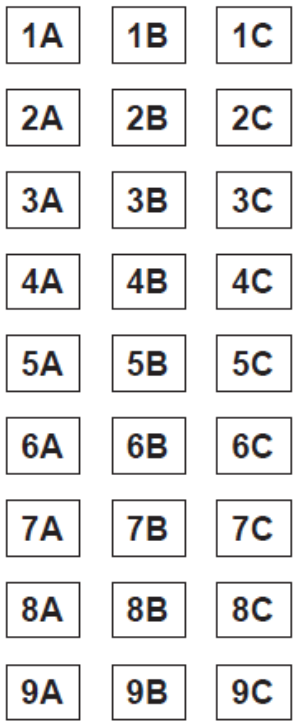
Avião (1 – vermelha)

Arquivo fonte: *aviao.c*, *aviao.cpp* ou *aviao.java*

Su Zuki é um empresário japonês acostumado a fazer viagens de avião, sempre na classe econômica, e quer saber qual seu assento com base no novo sistema da companhia aérea.

Todos os aviões contêm uma classe executiva e uma econômica, de forma que as primeiras fileiras do avião pertencem à classe executiva e as restantes à classe econômica.

Cada assento do avião é indicado por um número correspondente a sua fileira e por uma letra que indica a sua posição na fileira, sendo *A* a posição mais à esquerda da fileira, *B* a posição à direita do assento *A*, *C* o assento à direita do assento *B*, e assim por diante, seguindo o alfabeto de 26 letras. Por exemplo, a assento 9*B* está localizado na nona fileira, logo à direita do assento 9*A*. A figura abaixo mostra a numeração utilizada em um avião com nove fileiras de três assentos cada.



A companhia aérea adotou, para a classe econômica, um sistema no qual o bilhete indica a posição do passageiro na fila de embarque e não seu assento no vôo. A fila de embarque contém apenas passageiros da classe econômica.

Su Zuki descobriu que o primeiro passageiro da fila de embarque deve sempre sentar-se no assento localizado na primeira fileira da classe econômica, posição *A*. O segundo passageiro deve sentar-se nesta mesma fileira, posição *B*, e assim por diante, até que todos os assentos dessa fileira estejam ocupados. Esse processo é repetido a cada fileira da classe econômica, até que acabem os assentos desta classe ou todos os passageiros da fila já tenham embarcado.

Caso a classe econômica já esteja lotada e ainda haja passageiros na fila, esses passageiros embarcarão somente no próximo vôo.

Como viajante frequente, Su Zuki conhece bem os diversos modelos de aviões e é capaz de dizer o número total de fileiras no avião, o número de posições por fileira, e a partir de que fileira começa a classe econômica. Com base nessas informações, ele pediu a sua ajuda para descobrir, a partir de sua posição na fila, se ele tem assento garantido neste vôo e, caso tenha, qual seu assento.

Entrada

A entrada é composta de vários casos de teste. O teste contém uma linha com quatro inteiros F, C, E, B ($2 \leq F \leq 1.000$, $1 \leq C \leq 26$, $1 \leq E \leq F$, $2 \leq B \leq 50.000$) indicando, respectivamente, o número total de fileiras no avião, o número de posições por fileira, o número da primeira fileira da classe econômica e a posição na fila de embarque do Sr. Zuki.

O final da entrada é indicado por $F = 0$, $C = 0$, $E = 0$ e $B = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Seu programa deve imprimir, na saída padrão, uma única linha, contendo um inteiro e uma letra maiúscula, indicando a fileira e a posição em que o Su Zuki irá sentar-se, ou a frase "PROXIMO VOO" (em maiúsculas e sem acentos) caso não haja assentos suficientes para o Sr. Zuki no vôo.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`5 5 2 12`
`50 12 13 185`
`12 10 6 100`
`0 0 0 0`
	`4 B`
	`28 E`
	`PROXIMO VOO`

Insensibilidade (2 – branca)

Source file: *insens.c*, *insens.cpp* ou *insens.java*

O planeta Bizz fica a 133 upals de distância do planeta Terra (onde "upals" é uma unidade de medida dada por "um monte de anos-luz"), e parece ser o único planeta com vida além do nosso. Este planeta é muito interessante, pois, em cada país, seus habitantes têm uma característica diferente.

Um desses países é a Cegônia, que tem como característica o fato de que todos os seus habitantes são cegos. Em compensação, todos possuem um "sexto sentido" acentuado, podendo perceber o que está à sua volta mesmo sem enxergar.

Este ano, o governo da Cegônia fará um censo, e dentre os dados de seus habitantes, quer saber o quanto de insensibilidade cada pessoa possui. A insensibilidade indica quão ruim é a capacidade das pessoas de perceber os objetos à sua volta sem precisar enxergar.

Tal teste é feito da seguinte maneira: a pessoa é colocada em uma sala onde encontram-se vários objetos em posições pré-determinadas. A pessoa deve, então, dizer quais são as coordenadas de cada objeto dentro da sala.

Para cada objeto, calcula-se o quadrado da distância entre a posição adivinhada pela pessoa e a posição real do objeto; esse valor é chamado de *D*. O nível de insensibilidade da pessoa é dado pela soma de todos os *D*.

Por exemplo, suponha que na sala existam 4 objetos, nas coordenadas (1, 1), (3, 4), (5, 7) e (10, 10). Se a pessoa então disser que os objetos estão, respectivamente, nas posições (1, 2), (5, 4), (5, 7) e (19, 10), o valor de *D* para cada objeto será 1, 4, 0 e 81 e, portanto o nível de insensibilidade da pessoa é 1 + 4 + 0 + 81 = 86.

Você precisa fazer um programa que, dadas as coordenadas verdadeiras dos objetos e as coordenadas indicadas por uma pessoa, diga qual é o nível de insensibilidade dessa pessoa.

Entrada

A entrada é composta de vários casos de teste. A primeira linha da entrada contém um único inteiro N ($1 \leq N \leq 1.000$), indicando quantos objetos estão no quarto. As N linhas seguintes contêm cada uma quatro inteiros X_1, Y_1, X_2, Y_2 ($0 \leq X_i \leq 1000$). Cada linha representa um objeto: a posição real do objeto é (X_1, Y_1), e a posição onde a pessoa disse estar tal objeto é (X_2, Y_2).

O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Seu programa deve imprimir, na saída padrão, uma única linha, contendo um único inteiro, indicando o nível de insensibilidade da pessoa estudada.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
4 1 1 1 2 3 4 5 4 5 7 5 7 10 10 19 10 5 0 0 0 0 1 3 1 3 4 10 11 10 2 2 3 3 0 1 0 1 0	86 51

Jogo do Bicho (3 – verde)

Arquivo fonte: *bicho.c*, *bicho.cpp* ou *bicho.java*

Em um país muito distante, as pessoas são viciadas em um jogo de apostas bastante simples. O jogo é baseado em números e é chamado jogo do bicho. O nome do jogo deriva do fato que os números são divididos em 25 grupos, dependendo do valor dos dois últimos dígitos (dezenas e unidades), e cada grupo recebe o nome de um animal. Cada grupo é associado a um animal da seguinte forma: o primeiro grupo (burro) consiste nos números 01, 02, 03 e 04; o segundo grupo (águia) é composto dos números 05, 06, 07 e 08; e assim em diante, até o último grupo contendo os números 97, 98, 99 e 00.

As regras do jogo são simples. No momento da aposta, o jogador decide o valor da aposta V e um número N ($0 \leq N \leq 1000000$). Todos os dias, na praça principal da cidade, um número M é sorteado ($0 \leq M \leq 1000000$). O prêmio de cada apostador é calculado da seguinte forma:

- se M e N têm os mesmos quatro últimos dígitos (milhar, centena, dezena e unidade), o apostador recebe $V \times 3000$ (por exemplo, $N = 99301$ e $M = 19301$);
- se M e N têm os mesmos três últimos dígitos (centena, dezena e unidade), o apostador recebe $V \times 500$ (por exemplo, $N = 38944$ e $M = 83944$);
- se M e N têm os mesmos dois últimos dígitos (dezena e unidades), o apostador recebe $V \times 50$ (por exemplo, $N = 111$ e $M = 552211$);
- se M e N têm os dois últimos dígitos no mesmo grupo, correspondendo ao mesmo animal, o apostador recebe $V \times 16$ (por exemplo, $N = 82197$ e $M = 337600$);
- se nenhum dos casos acima ocorrer, o apostador não recebe nada.

Obviamente, o prêmio dado a cada apostador é o máximo possível de acordo com as regras acima. No entanto, não é possível acumular prêmios, de forma que apenas um dos critérios acima deve ser aplicado no cálculo do prêmio. Se um número N ou M com menos de quatro dígitos for apostado ou sorteado, assuma que dígitos 0 devem ser adicionados na frente do número para que se torne de quatro dígitos; por exemplo, 17 corresponde a 0017.

Dado o valor apostado, o número escolhido pelo apostador, e o número sorteado, seu programa deve calcular qual o prêmio que o apostador deve receber.

Entrada

A entrada contém vários casos de teste. Cada caso consiste em apenas uma linha, contendo um número real V e dois inteiros N e M, representando respectivamente o valor da aposta com duas casas decimais ($0.01 \leq V \leq 1000.00$), o número escolhido para a aposta ($0 \leq N \leq 1000000$) e o número sorteado ($0 \leq M \leq 1000000$). O final da entrada é indicado por uma linha contendo $V = M = N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada um dos casos de teste seu programa deve imprimir uma linha contendo um número real, com duas casas decimais, representando o valor do prêmio correspondente à aposta dada.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
32.20 32 213929 10.50 32 213032 2000.00 340000 0 520.00 874675 928567 10.00 1111 578311 0 0 0	515.20 5250.00 6000000.00 0.00 500.00

Rouba-Monte (4 – azul)

Arquivo fonte: *rouba.c*, *rouba.cpp* ou *rouba.java*

Um dos jogos de cartas mais divertidos para crianças pequenas, pela simplicidade, é Rouba-Monte. O jogo utiliza um ou mais baralhos normais e tem regras muito simples. Cartas são distinguidas apenas pelo valor (ás, dois, três, . . .), ou seja, os naipes das cartas não são considerados (por exemplo, ás de paus e ás de ouro têm o mesmo valor).

Inicialmente as cartas são embaralhadas e colocadas em uma pilha na mesa de jogo, chamada de pilha de compra, com face voltada para baixo. Durante o jogo, cada jogador mantém um monte de cartas, com face voltada para cima; em um dado momento o monte de um jogador pode conter zero ou mais cartas. No início do jogo, todos os montes dos jogadores têm zero cartas. Ao lado da pilha de compras encontra-se uma área denominada de área de descarte, inicialmente vazia, e todas as cartas colocadas na área de descarte são colocadas lado a lado com a face para cima (ou seja, não são empilhadas).

Os jogadores, dispostos em um círculo ao redor da mesa de jogo, jogam em sequência, em sentido horário. As jogadas prosseguem da seguinte forma:

- O jogador que tem a vez de jogar retira a carta de cima da pilha de compras e a mostra aos outros jogadores; vamos chamar essa carta de carta da vez.
- Se a carta da vez for igual a alguma carta presente na área de descarte, o jogador retira essa carta da área de descarte colocando-a, juntamente com a carta da vez, no topo de seu monte, com as faces voltadas para cima, e continua a jogada (ou seja, retira outra carta da pilha de compras e repete o processo).
- Se a carta da vez for igual à carta de cima de um monte de um outro jogador, o jogador "rouba" esse monte, empilhando-o em seu próprio monte, coloca a carta da vez no topo do seu monte, face para cima, e continua a jogada.
- Se a carta da vez for igual à carta no topo de seu próprio monte, o jogador coloca a carta da vez no topo de seu próprio monte, com a face para cima, e continua a jogada.
- Se a carta da vez for diferente das cartas da área de descarte e das cartas nos topos dos montes, o jogador a coloca na área de descarte, face para cima, e a jogada se encerra (ou seja, o próximo jogador efetua a sua jogada). Note que esse é o único caso em que o jogador não continua a jogada.

O jogo termina quando não há mais cartas na pilha de compras. O jogador que tiver o maior monte (o monte contendo o maior número de cartas) ganha o jogo. Se houver empate, todos os jogadores com o monte contendo o maior número de cartas ganham o jogo.

Entrada

A entrada é composta de vários casos de teste. A primeira linha de um caso de teste contém dois inteiros N e J, representando respectivamente o número de cartas no baralho ($2 \leq N \leq 10.000$) e o número de jogadores ($2 \leq J \leq 20$ e $J \leq N$). As cartas do baralho são representadas por números inteiros de 1 a 13 e os jogadores são identificados por inteiros de 1 a J. O primeiro jogador a jogar é o de número 1, seguido do jogador de número 2, . . ., seguido pelo jogador de número J, seguido pelo jogador de número 1, seguido do jogador de número 2, e assim por diante enquanto houver cartas na pilha de compras. A segunda linha de um caso de teste contém N inteiros entre 1 e 13, separados por um espaço em branco, representando as cartas na pilha de compras. As cartas são retiradas da pilha de compras na ordem em que aparecem na entrada. O final da entrada é indicado por uma linha com $N = J = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste seu programa deve imprimir uma linha, contendo o número de cartas do monte do jogador ou jogadores que ganharam a partida, seguido de um espaço em branco, seguido do(s) identificador(es) dos jogadores que ganharam a partida. Se há mais de um jogador vencedor imprima os identificadores dos jogadores em ordem crescente, separados por um espaço em branco.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
4 2 10 7 2 5 6 3 1 2 1 2 1 2 8 2 3 3 1 1 2 3 4 5 0 0	0 1 2 5 1 3 2

Triangle (5 – amarela)

Source file name: *triangle.c*, *triangle.cpp* ou *triangle.java*

A long time ago, the Egyptians figured out that a triangle with sides of length 3, 4, and 5 had a right angle as its largest angle. You must determine if other triangles have a similar property.

Input

Input represents several test cases, followed by a line containing 0 0 0. Each test case has three positive integers, less than 10.000, denoting the lengths of the sides of a triangle.

The input must be read from standard input.

Output

For each test case, a line containing "right" if the triangle is a right triangle, and a line containing "wrong" if the triangle is not a right triangle.

The output must be written to standard output.

Sample input	**Output for the sample input**

`6 8 10`
`25 52 60`
`5 12 13`
`0 0 0`
	`right`
	`wrong`
	`right`

Tornado! (6 – laranja)

Source file name: *tornado.c*, *tornado.cpp* ou *tornado.java*

Is this crazy weather the result of mankind's continuous interference in the environment? Or is it simply the normal cycle of climate changes through the ages? No one seems to know for sure, but the fact is that natural phenomena such as tornadoes and hurricanes have been hitting our country with more force and frequence than in past decades.

One tornado has just hit Silverado Farm, a cattle and milk producer, and made havoc. The barn roof was torn, several trees were uprooted, the farm truck was overturned... But the worst thing is that the tornado destroyed several sections of the fence that surrounded the property. The fence was very well built, with concrete posts every two meters, and barbed wire enclosing the whole farm perimeter (the perimeter, in meters, is an even number, making the fence perfectly regular).

Now several posts are broken or missing, and there are gaps in the fence. To prevent the cattle from getting out of the property, the fence must be restored as quickly as possible. Reconstructing the fence to its original form, with concrete posts, will take a long time. In the meantime, the farm owners decided to close the gaps with a temporary fence, made with wooden posts. Wooden posts will be placed in exactly the same spots where missing/broken concrete posts were/are. However, in order to make the temporary reconstruction faster and less expensive, the owners decided to use fewer posts: a wooden post will be used to replace a missing/broken concrete post only if the length of the barbed wired needed to close the distance to the next post (wooden or concrete) exceeds four meters.

Given the description of which posts are missing/broken, you must write a program to determine the smallest number of wooden posts needed to close all the gaps in the fence, according to the owners' decision.

Input

The input contains several test cases. The first line of a test case contains one integer N indicating the number of original concrete posts in the fence ($5 \leq N \leq 5000$). The second line of a test case contains N numbers X_i indicating the state of each concrete post after the tornado ($0 \leq X_i \leq 1$ for $1 \leq i \leq N$). If $X_i = 1$ post i is in good condition, if $X_i = 0$ post i is broken or missing. Note that post X_N is next to post X_1. The end of input is indicated by $N = 0$.

The input must be read from standard input.

Output

For each test case in the input your program must produce one line of output, containing an integer indicating the smallest number of wooden posts that are needed to restore the fence, according to the owners' decision.

The output must be written to standard output.

Sample input	**Output for the sample input**
10 1 0 0 1 0 0 1 0 1 1 11 1 0 0 1 0 0 0 1 1 0 1 12 0 0 0 0 0 1 1 0 0 0 1 1 0	2 2 3