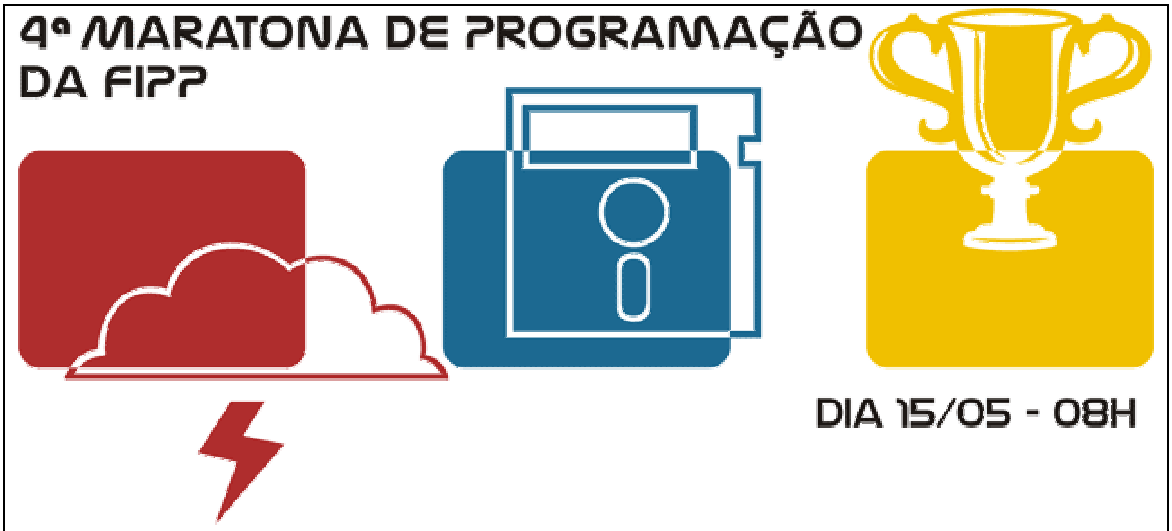
4ª MARATONA DE PROGRAMAÇÃO DA FIPP



DIA 15/05 - 08H

CADERNO DE PROBLEMAS

15/05/2008 – 08:00 às 12:00

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 5 problemas, 2 deles estão descritos em inglês.

1. É proibido utilizar livros, apostilas impressas, manuais ou qualquer outro material que não seja fornecido pela comissão organizadora durante a prova, com exceção de dicionários de inglês. É permitida apenas a consulta do *help* do ambiente de programação se este estiver disponível.
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 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:
```
Scanner sc = new Scanner(); int i = sc.nextInt();
String s = sc.next(); float f = sc.nextFloat();
System.out.println(); System.out.printf("%d",i);
```

Quando a linguagem escolhida para a resolução dos problemas for o Pascal, recomendamos que não seja utilizada a instrução: `uses newdelay, crt;`

Faculdade de Informática de Presidente Prudente

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

Penalidade mínima (1 – vermelho)

Arquivo fonte: *penalidade.c*, *penalidade.cpp*, *penalidade.pas* ou *penalidade.java*

A Sra. Bastos é uma elaboradora de passatempos matemáticos e pediu para que você criasse um programa que conseguisse jogar de forma eficiente a sua mais nova criação.

O jogo consiste em um tabuleiro formado por casas dispostas em N linhas por N colunas. Cada casa contém um inteiro não-negativo. No começo do jogo, uma peça é colocada na casa localizada no canto superior esquerdo, ou seja, na posição (1,1). O objetivo do jogo é mover a peça até a casa localizada no canto inferior direito (posição (N,N)) somente movendo um único quadrado para baixo ou para a direita em cada passo. Além disso, a peça não pode ser colocada em nenhum quadrado que contenha o número zero.

O custo do caminho utilizado para percorrer o tabuleiro corresponde ao produto de todos os números das casas percorridos no caminho. A penalidade é definida utilizando a representação decimal do custo, sendo representada pelo número de dígitos zeros, contados da direita para a esquerda, antes do primeiro dígito diferente de zero. Por exemplo, um custo igual a 501000 tem penalidade 3, e um custo igual a 501 tem penalidade zero.

O objetivo do jogo é conseguir chegar à casa (N,N) através de um caminho "otimizado". Dizemos que o caminho foi otimizado se a penalidade for mínima.

Tarefa

Escreva um programa que, dado um tabuleiro, determine a penalidade do custo otimizado.

Entrada

A entrada contém vários conjuntos de testes, que deve ser lido do dispositivo de entrada padrão (normalmente o teclado). A primeira linha da entrada contém um inteiro N que indica o número de linhas e colunas do tabuleiro ($1 \leq N \leq 1000$). As N linhas seguintes contêm N inteiros I cada ($1 \leq I \leq 1000000$), que representam o valor da casa do tabuleiro naquela posição. Existe pelo menos uma solução possível para todos os casos de teste.

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

Exemplo de Entrada

```
3
1 2 3
4 5 6
7 8 9
3
5 7 6
4 0 1
3 2 5
4
1 3 0 0
0 8 2 25
6 5 0 3
0 15 7 4
0
```

Saída

Seu programa deve imprimir, na saída padrão, uma única linha, contendo a penalidade do custo "otimizado".

Exemplo de Saída

```
0
1
2
```

He is offside! (2 – amarelo)

Source file: *he.c*, *he.cpp*, *he.pas* ou *he.java*

Hemisphere Network is the largest television network in Tumbolia, a small country located east of South America (or south of East America). The most popular sport in Tumbolia, unsurprisingly, is soccer; many games are broadcast every week in Tumbolia.

Hemisphere Network receives many requests to replay dubious plays; usually, these happen when a player is deemed to be offside by the referee. An attacking player is *offside* if he is nearer to his opponents' goal line than the second last opponent. A player is not offside if

- he is level with the second last opponent or
- he is level with the last two opponents.

Through the use of computer graphics technology, Hemisphere Network can take an image of the field and determine the distances of the players to the defending team's goal line, but they still need a program that, given these distances, decides whether a player is offside.

Input

The input contains several test cases. The first line of each test case contains two integers A and D separated by a single space indicating, respectively, the number of attacking and defending players involved in the play ($2 \le A, D \le 11$). The next line contains A integers B_i separated by single spaces, indicating the distances of the attacking players to the goal line ($1 \le B_i \le 10^4$). The next line contains D integers C_j separated by single spaces, indicating the distances of the defending players to the goal line ($1 \le C_j \le 10^4$). The end of input is indicated by $A = D = 0$.

The input must be read from standard input.

Sample Input

```
2 3
500 700
700 500 500
2 2
200 400
200 1000
3 4
530 510 490
480 470 50 310
0 0
```

Output

For each test case in the input print a line containing a single character: "Y" (uppercase) if there is an attacking player offside, and "N" (uppercase) otherwise.

The output must be written to standard output.

Sample Output

```
N
Y
N
```

Magic Trick (3 – laranja)

Source file: *magic.c*, *magic.cpp*, *magic.pas* ou *magic.java*

A magician invented a new card trick and presented it in the prestigious American Conference of Magicians (ACM). The trick was so nice it received the 'Best Magic Award' at the conference. The trick requires three participants: the magician himself, a spectator and an assistant. During the trick the spectator is asked to shuffle a deck of 52 cards and pick randomly 5 cards out of the deck. The five cards are given to the assistant (without the magician seeing the cards) who looks at them and shows four of the five cards one by one to the magician. After seeing the four cards the magician magically guesses the missing fifth card!

The trick works because once the assistant has the five cards he can always choose four of them and use those to 'code' information about the fifth one. The code is based on an ordering of the cards. Cards are ordered first by their suits and then by their face value. We will use the following order:

- H < C < D < S (Hearts, Clubs, Diamonds, Spades) for suits; and
- 1 < 2 < ... < 9 < T < J < Q < K for face values, where T, J, Q and K stand for Ten, Jack, Queen and King, respectively.

Assume the spectator chose the cards JD, 8S, 7H, 8C, QH (Jack of Diamonds, 8 of Spades, 7 of Hearts, 8 of Clubs and Queen of Hearts). The strategy for the assistant is the following:

- Find a suit *s* which appears at least twice in the set of chosen cards (Hearts in the example). If more than one suit appears two times, choose the one with lowest order.
- Hide the card *x* with suit *s* that is at most six positions ahead in the cyclic order 1 < 2 < ... < T < J < Q < K < 1 < 2 < ... of another card *y* of the same suit. That is always possible since there are only thirteen cards with the same suit (in the example the assistant hides QH). If two or more cards satisfy the criteria above, choose the one with the smallest face value.
- Show *y* to the magician. At this point the magician knows the suit of the hidden card, and also knows that the face value of the hidden card *x* is at most six positions in front of the face value of *y*.
- With the three cards the assistant has left, he must code a number between 1 and 6. That can be done as follows. Say the three cards $z1$, $z2$, $z3$ are in the order $z1 < z2 < z3$. Each of six possible orders in which these three cards can be shown may be interpreted to convey information about a number.

– $z1$, $z2$, $z3$ means 1,

– $z1$, $z3$, $z2$ means 2,

– $z2$, $z1$, $z3$ means 3,

– $z2$, $z3$, $z1$ means 4,

– $z3$, $z1$, $z2$ means 5,

– $z3$, $z2$, $z1$ means 6.

In this way, once the magician is shown the four cards one by one, he has enough information to "magically" guess the fifth one!

Your job is to develop a program that, given the four cards shown by the assistant, informs the magician which is the hidden card.

Input

The input contains several test cases. The first line in the input contains an integer N specifying the number of test cases ($1 \leq N \leq 10000$). Each test case is composed by one line, which contains the description of the four cards, separated by a space, in the order they were presented by the assistant.

The input must be read from standard input.

Sample Input

```
2
7H 8S 8C JD
TC 2D 1S 5H
```

Output

For each test case in the input your program must produce one line of output, containing the description of the hidden card.

The output must be written to standard output.

Sample Output

```
QH
1C
```

Lobo Mau (4 – branco)

Arquivo fonte: *lobo.c*, *lobo.cpp*, *lobo.pas* ou *lobo.java*

Na fazenda do Sr. Amarante existe um certo número de ovelhas. Enquanto elas estão dormindo profundamente, alguns lobos famintos tentam invadir a fazenda e atacar as ovelhas. Ovelhas normais ficariam indefesas diante de tal ameaça, mas felizmente as ovelhas do Sr. Amarante são praticantes de artes marciais e conseguem defender-se adequadamente.

A fazenda possui um formato retangular e consiste de campos arranjados em linhas e colunas. Cada campo pode conter uma ovelha (representada pela letra 'k'), um lobo (letra 'v'), uma cerca (símbolo '#') ou simplesmente estar vazio (símbolo '.'). Consideramos que dois campos pertencem a um mesmo pasto se podemos ir de um campo ao outro através de um caminho formado somente com movimentos horizontais ou verticais, sem passar por uma cerca. Na fazenda podem existir campos vazios que não pertencem a nenhum pasto. Um campo vazio não pertence a nenhum pasto se é possível "escapar" da fazenda a partir desse campo (ou seja, caso exista um caminho desse campo até a borda da fazenda).

Durante a noite, as ovelhas conseguem combater os lobos que estão no mesmo pasto, da seguinte forma: se em um determinado pasto houver mais ovelhas do que lobos, as ovelhas sobrevivem e matam todos os lobos naquele pasto. Caso contrário, as ovelhas daquele pasto são comidas pelos lobos, que sobrevivem. Note que caso um pasto possua o mesmo número de lobos e ovelhas, somente os lobos sobreviverão, já que lobos são predadores naturais, ao contrário de ovelhas.

Tarefa

Escreva um programa que, dado um mapa da fazenda do Sr. Amarante indicando a posição das cercas, ovelhas e lobos, determine quantas ovelhas e quantos lobos estarão vivos na manhã seguinte.

Entrada

A entrada contém vários conjuntos de testes, que devem ser lidos do dispositivo de entrada padrão (normalmente o teclado). A primeira linha da entrada contém dois inteiros R e C que indicam o número de linhas ($3 \leq R \leq 200$) e de colunas ($3 \leq C \leq 200$) de campos da fazenda. Cada uma das R linhas seguintes contém C caracteres, representando o conteúdo do campo localizado naquela linha e coluna (espaço vazio, cerca, ovelha ou lobo).

Exemplo de Entrada

```
6 6
...#..
.##v#.
#v.#.#
#.k#.#
.###.#
...###
8 8
.######.
#..k...#
#.####.#
#.#v.#.#
#.#.k#k#
#k.##..#
#.v..v.#
.######.
9 12
.###.#####..
#.kk#...#v#.
#..k#.#.#.#.
#..##k#...#.
#.#v#k###.#.
#..#v#....#.
#...v#v####.
.####.#vv.k#
.......#####
0 0
```

Saída

Seu programa deve imprimir, na saída padrão, uma única linha, contendo dois inteiros, sendo que o primeiro representa o número de ovelhas e o segundo representa o número de lobos que ainda estão vivos na manhã seguinte.

Exemplo de Saída

```
0 2
3 1
3 5
```

Bafo (5 – verde)

Arquivo fonte: *bafo.c*, *bafo.cpp*, *bafo.pas* ou *bafo.java*

Álbuns de figurinhas – sejam de times de futebol, princesas ou super-heróis – têm marcado gerações de crianças e adolescentes. Conseguir completar um álbum é uma tarefa muitas vezes árdua, envolvendo negociações com colegas para a troca de figurinhas. Mas a existência das figurinhas propicia uma outra brincadeira, que foi muito popular entre crianças no século passado: o jogo de bater figurinhas (o famoso "Bafo").

O jogo é muito simples, mas divertido (e muito competitivo). No início de uma partida, cada criança coloca em uma pilha um certo número de figurinhas. Uma partida é composta de rodadas; a cada rodada as crianças batem com a mão sobre a pilha de figurinhas, tentando virá-las com o vácuo formado pelo movimento da mão. As crianças jogam em turnos, até que a pilha de figurinhas esteja vazia. Ganha a partida a criança que conseguir virar mais figurinhas.

Aldo e Beto estão jogando bafo com todas as suas figurinhas e pediram sua ajuda para calcular quem é o vencedor.

Tarefa

Você deve escrever um programa que, dada a quantidade de figurinhas que Aldo e Beto viraram em cada rodada, determine qual dos dois é o vencedor.

Entrada

A entrada é composta de vários casos de teste, cada um correspondendo a uma partida entre Aldo e Beto. A primeira linha de um caso de teste contém um número inteiro R que indica quantas rodadas ocorreram na partida. Cada uma das R linhas seguintes contém dois inteiros, A e B, que correspondem, respectivamente, ao número de figurinhas que Aldo e Beto conseguiram virar naquela rodada. Em todos os casos de teste há um único vencedor (ou seja, não ocorre empate). O final da entrada é indicado por $R = 0$.

A entrada deve ser lida do dispositivo de entrada padrão (normalmente o teclado).

Restrições

$1 \leq R \leq 1000$ ($R = 0$ apenas para indicar o final da entrada)
$0 \leq A \leq 100$
$0 \leq B \leq 100$

Exemplo de Entrada

```
2
1 5
2 3
3
0 0
4 7
10 0
0
```

Saída

Para cada caso de teste da entrada, seu programa deve produzir três linhas na saída. *A* primeira linha deve conter um identificador do caso de teste, no formato "`Teste n`", onde *n* é numerado seqüencialmente a partir de 1. A segunda linha deve conter o nome do vencedor (Aldo ou Beto). A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

A saída deve ser escrita no dispositivo de saída padrão (normalmente a tela).

Exemplo de Saída

```
Teste 1
Beto

Teste 2
Aldo
```