Rush

2^a Maratona de Programação dos Alunos da UFPR – 2011/1

Sevidor BOCA:

`http://maratona.c3sl.ufpr.br/boca/`

Organizadores:

Bruno César Ribas, André Luiz Guedes, Eduardo Augusto Ribas

Lembretes:

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova.
- A correção é automatizada, portanto, siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa. Deve-se considerar entradas e saídas padrão.
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. As soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Teste seu programa antes de submetê-lo. A cada problema detectado (erro de compilação, erro em tempo de execução, solução incorreta, formatação imprecisa, tempo excedido ...), há penalização de 20 minutos. O tempo é critério de desempate entre duas ou mais equipes com a mesma quantidade de problemas resolvidos.
- Utilize o *clarification* para dúvidas da prova. Os juízes podem opcionalmente atendê-lo com respostas acessíveis a todos.

Prova Rush

2ª Maratona de Programação dos Alunos da UFPR

28 de junho de 2011

Sumário

1 Problema A: Esquerda Volver!

`Arquivo: esquerda.[c|cpp|java|pas]`

Este ano o sargento está tendo mais trabalho do que de costume para treinar os recrutas. Um deles é muito atrapalhado, e de vez em quando faz tudo errado - por exemplo, ao invés de virar à direita quando comandado, vira à esquerda, causando grande confusão no batalhão.

O sargento tem fama de durão e não vai deixar o recruta em paz enquanto não aprender a executar corretamente os comandos. No sábado à tarde, enquanto todos os outros recrutas estão de folga, ele obrigou o recruta a fazer um treinamento extra. Com o recruta marchando parado no mesmo lugar, o sargento emitiu uma série de comandos "esquerda volver!" e "direita volver!". A cada comando, o recruta deve girar sobre o mesmo ponto e dar um quarto de volta na direção correspondente ao comando. Por exemplo, se o recruta está inicialmente com o rosto voltado para a direção norte, após um comando "esquerda volver!" ele deve ficar com o rosto voltado para a direção oeste. Se o recruta está inicialmente com o rosto voltado para o leste, após um comando "direita volver!" ele deve ter o rosto voltado para o sul.

No entando, durante o treinamento, em que o recruta tinha inicialmente o rosto voltado para o norte, o sargento emitiu uma série tão extensa de comandos, e tão rapidamente, que até ele ficou confuso, e não sabe mais para qual direção o recruta deve ter seu rosto voltado após executar todos os comandos. Você pode ajudar o sargento?

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro N que indica o número de comandos emitidos pelo sargento ($1 \leq N \leq 1000$). A segunda linha contém N caracteres, descrevendo a série de comandos emitidos pelo sargento. Cada comando é representado por uma letra: 'E' (para "esquerda volver!" e 'D' (para "direita volver!"). O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma única linha de saída, indicando a direção para a qual o recruta deve ter sua face voltada após executar a série de comandos, considerando que no início o recruta tem a face voltada para o norte. A linha deve conter uma letra entre 'N', 'L', 'S' e 'O', representando respectivamente as direções norte, leste, sul e oeste

A saída deve ser escrita na saída padrão.

Exemplo de entrada

```
3
DDE
2
EE
0
```

Exemplo de saída

```
L
S
```

2 Problema B: Bora Bora

`Arquivo: bora.[c|cpp|java|pas]`

Bora Bora é um jogo de cartas simples para crianças, inventado na Ilha do Pacífico Sul de mesmo nome. Duas ou mais pessoas podem jogar, usando um baralho de cartas normais. As cartas têm os valores normais: Ás, 2, 3, 4, 5, 6, 7, 8, 9, 10, Valete, Dama e Rei. Cada carta tem também um dos quatro naipes: Paus, Ouros, Copas e Espadas.

Os jogadores sentam-se em um círculo ao redor da mesa e jogam por turnos. O próximo jogador a jogar pode ser aquele à esquerda (sentido horário) ou o jogador à direita (sentido anti-horário) do jogador atual, dependendo das cartas jogadas, como você verá. No começo, o sentido do jogo é horário.

O baralho é embaralhado e é dada a cada jogador uma mão de cartas. O restante do baralho é colocado, virado para baixo, na mesa; a essa pilha é dado o nome de pilha de saque. Então a primeira (a mais acima) carta é removida da pilha e colocada na mesa, virada para cima, começando outra pilha, chamada de pilha de descarte.

O objetivo do jogo é que um jogador descarte todas suas cartas. Em cada turno, um jogador descarta no máximo uma carta. Uma carta pode ser descartada apenas se ela tem o mesmo valor ou o mesmo naipe da carta que se encontra no topo da pilha de descarte. Um jogador descarta uma carta colocando-a, virada para cima, na pilha de descarte (essa carta se torna a carta do topo). Se um jogador não tiver uma carta passível de ser descartada em seu turno, ele deve sacar uma carta da pilha de saque e adicionar à sua mão; se ele puder descartar essa carta, ele o faz, caso contrário ele não faz mais nada e seu turno acaba. Um jogador sempre descarta a carta mais alta que ele consegue. O valor de uma carta é determinado primeiro pelo valor da carta e então pelo naipe. A ordem dos valores é o valor em si (Ás é o mais fraco e Rei o mais forte), e a ordem dos naipes é, do menor para o maior, Paus, Ouros, Copas e Espadas. Portanto, a carta de maior valor é o Rei de Espadas e a de menor valor é o Ás de Paus. Como exemplo, a Dama de Ouros tem um valor maior que um Valete (qualquer naipe) mas tem um valor menor que a Dama de Copas.

Algumas das cartas descartadas afetam o jogo, como se segue:

- quando uma Dama é descartada, a direção de jogo é invertida: se a direção é horária, ela se torna anti-horária, e vice-versa;
- quando um Sete é descartado, o próximo jogador deve sacar duas cartas da pilha de saque (o número de cartas em sua mão aumenta por dois), e perde a vez (não descarta nenhuma carta);
- quando um Ás é descartado, o próximo jogador deve sacar uma carta da pilha de saque (o número de cartas em sua mão aumenta por um), e perde sua rodada (não descarta nenhuma carta);
- quando um Valete é descartado, o próximo jogador perde a vez (não descarta nenhuma carta).

Perceba que a penalidade da primeira carta da pilha de descarte (a carta sacada da pilha de saque no começo) é aplicada ao primeiro jogador a jogar. Por exemplo, se o primeiro jogador é p e a primeira cartana pilha de descarte é um Ás, o jogador p saca uma carta da pilha de saque e não descarta nenhuma carta em seu primeiro turno. Note também que se a

primeira carta é uma Dama, o sentido do jogo é invertido para o anti-horário, mas o primeiro jogador a jogar permanece o mesmo. O vencedor é o jogador que descarta todas suas cartas primeiro (o jogo acaba depois de o vencedor descartar sua última carta).

Dada a descrição do baralho embaralhado e o número de jogadores, escreva um programa que determine quem vencerá o jogo.

Entrada

A entrada contém diversos casos de teste. A primeira linha de um caso de teste contém três inteiros P, M e N, separados por espaço, indicando respectivamente o número de jogadores ($2 \leq P \leq 10$), o número de cartas distribuídas para cada jogador no começo da partida ($1 \leq M \leq 11$) e o número de cartas no baralho embaralhado ($3 \leq N \leq 300$). Cada uma das próximas N linhas contém a descrição de uma carta. Uma carta é descrita por um inteiro X e um caractere S, separados por um espaço, representando respectivamente o valor da carta e seu naipe. O valor das cartas é mapeado com inteiros de 1 a 13 (Ás é 1, Valete é 11, Dama é 12 e Rei é 13). Os naipes das cartas são designados pela primeira letra do naipe: 'C' (Paus - Clubs), 'D' (Ouros - Diamonds), 'H' (Copas - Hearts) ou 'S' (Espadas - Spades).

Os jogadores são identificados com valores de 1 a P e sentam-se em um círculo, no sentido horário, $1, 2 \cdots P, 1$. As primeiras $P \times M$ cartas do baralho são distríbuidas aos jogadores: as primeiras M cartas ao primeiro jogador (jogador 1), as próximas M cartas ao segundo jogador (jogador 2), e assim por diante. Depois de distribuir as cartas aos jogadores, a próxima carta do baralho - a $(P \times M + 1)$-ésima carta - é usada para começar a pilha de descarte, e as cartas restantes formam a pilha de saque. A $(P \times M + 2)$-ésima carta a aparecer na entrada é a carta do topo da pilha de saque, e a última carta a aparecer na entrada (a N-ésima carta) é a carta do fundo da pilha de saque (a última carta que pode ser sacada). O jogador 1 é sempre o primeiro a jogar (mesmo se a carta usada para começar a pilha de descarte é uma Dama). Todos os casos de teste têm um vencedor, e em todos casos de teste o número de cartas no baralho é suficiente para jogar até o fim da partida.

O final da entrada é indicado por uma linha contendo apenas três zeros, separados por espaços.

Saída

Para cada caso de teste na entrada, seu programa deve imprimir uma única linha, contendo o número do jogador que ganha a partida.

Exemplo de entrada

```
2 2 10
1 D
7 D
1 S
3 C
13 D
1 S
5 H
12 D
7 S
2 C
3 2 11
1 S
7 D
11 D
3 D
7 D
3 S
11 C
8 C
9 H
6 H
9 S
3 3 16
1 H
10 C
13 D
7 C
10 H
2 S
2 C
10 S
8 S
12 H
11 C
1 C
1 C
4 S
5 D
6 S
0 0 0
```

Exemplo de saída

```
1
3
2
```

3 Problema C: Sub-sequências

`Arquivo: sub.[c|cpp|java|pas]`

Uma *subsequência* de uma *sequência de caracteres* S é definida como uma sequência de caracteres de S, não necessariamente consecutivos, na mesma ordem emque eles ocorrem na sequência original.

Dadas duas sequências de caracteres, S_1 e S_2, dizemos que S_1 possui grau N de independência em relação a S_2 se, dada qualquer subsequência de tamanho N de S_1, não é possível formar tal subsequência a partir de S_2.

Por exemplo, o grau de independência da sequência S_1 =‘ababaa’ em relação à sequência S_2 =‘abbaa’ é igual a 3, pois todas as subsequências de S_1 de tamanho 1 (‘a’,‘b’) e todas as subsequências de tamanho 2 (‘aa’, ‘ab’, ‘ba’, ‘bb’) podem ser formadas a partir de S_2, mas a subsequência ‘bab’, de tamanho 3, não pode ser formada a partir de S_2.

Escreva um programa que, dadas duas sequências S_1 e S_2, determine o grau N de independência de S_1 em relação a S_2.

Entrada

A entrada contém vários conjuntos de testes, que devem ser lido do *dispositivo de entrada padrão*. A primeira linha da entrada possui um inteiro N que determina quantos conjuntos de teste serão lidos. Cada conjunto de teste possui 3 linhas. A primeira linha, de um conjunto de teste, contém dois inteiros M e P que indicam respectivamente o comprimento da sequência S_1 ($1 \leq N \leq 2000$) e o comprimento da sequência S_2 ($1 \leq N \leq 2000$). A segunda linha contém a sequência S_1 e a terceira linha contém a sequência S_2. As sequências são formadas somente pelas letras minúsculas sem acento (‘a’ - ‘z’). As sequências possuem no máximo 2000 caracteres. Sempre existe uma solução para os casos de teste fornecidos.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha para cada caso de teste, contendo o grau N de independência de S_1 em relação S_2

Exemplo de entrada

```
3
6 5
ababaa
abbaa
5 5
babab
babba
6 11
banana
anbnaanbaan
```

Exemplo de saída

```
3
3
5
```

4 Problema D: Mini-Poker

`Arquivo: poker.[c|cpp|java|pas]`

Mini-Poker é o nome de um jogo de cartas que é uma simplificação do Poker, um dos mais famosos jogos de cartas do mundo. Mini-Poker é jogado com um baralho normal de 52 cartas, com quatro naipes (copas, paus, espadas e ouro), cada naipe compreendendo treze cartas(Ás, 2, 3, 4, 5, 6, 7, 8, 9, 10, Valete, Dama, Rei).

No início do jogo, cada jogador recebe cinco cartas. O conjunto de cinco cartas vale um certo número de pontos, de acordo com as regras descritas abaixo. Diferentemente do jogo de Poker normal, em Mini-Poker o naipe das cartas é desconsiderado. Assim, para simplificar a descrição do jogo, vamos utilizar os números de 1 a 13 para identificar as cartas do baralho, na ordem dada acima. Uma outra diferença é que pode ocorrer empate entre mais de um vencedor; nesse caso os vencedores dividem o prêmio.

As regras para a pontuação em Mini-Poker são as seguinte:

1. Se as cinco cartas estão em sequência a partir da carta x (ou seja, os valores das cartas são $x, x+1, x+2, x+3$ e $x+4$), a pontuação é $x+200$ pontos. Por exemplo, se as cartas recebidas são $10, 9, 8, 11$ e 12, a pontuação é 208 pontos.

2. Se há quatro cartas iguais x (uma quadra, ou seja, os valores das cartas são x, x, x, x e y), a pontuação é $x+180$ pontos. Por exemplo, se as cartas recebidas são 1,1,1,10 e 1, a pontuação é 181 pontos.

3. Se há três cartas iguais x e duas outras cartas iguais y (uma trinca e um par, ou seja, os valores das cartas são x, x, x, y e y), a pontuação é $x+160$. Por exemplo se as cartas recebidas são 10,4,4,10 e 4, a pontuação é 164 pontos.

4. Se há três cartas iguais x e duas outras cartas diferentes y e z (uma trinca, ou seja, os valores das cartas são x, x, x, y e z), a pontuação é $x+140$ pontos. Por exemplo, se as cartas recebidas são 2,3,2,2 e 13, a pontuação é 142 pontos.

5. Se há duas cartas iguais x, duas outras cartas iguais y ($x \neq y$ e uma outra carta distinta z (dois pares, ou seja, os valores das cartas são x, x, y, y e z), a pontuação é $3 \times x + 2 \times y + 20$ pontos, em que $x > y$. Por exemplo, se as cartas recebidas são 12,7,12,8 e 7, a pontuação é 70 pontos.

6. Se há apenas duas cartas iguais x e as outras são todas distintas (um par, ou seja, os valores das cartas são x, x, y, z e t), a pontuação é x pontos. Por exemplo, se as cartas recebidas são 12,13,5,8 e 13, a pontuação é 13 pontos.

7. Se todas as cartas são distintas, não há pontuação.

Escreva um programa que, fornecidas as cartas dadas a um jogador, calcule pontuação do jogador naquela jogada.

Entrada

A entrada é composta por vários casos de teste, cada um correspondendo a uma jogada. A primeira linha da entrada contém um inteiro N que indica o número de casos de teste ($1 \leq N \leq 10$). Cada uma das N linhas seguintes contém cinco números inteiros C_1, C_2, C_3, C_4, C_5, representando as cinco cartas recebidas por um jogador ($1 \leq C_1, C_2, C_3, C_4, C_5 \leq 13$).

A entrada deve ser lida da entrada padrão.

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 sequencialmente a partir de 1. A segunda linha deve conter a pontuação do jogador considerando as cinco cartas recebidas. A terceira linha deve ser deixada em branco. A grafica mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

A saída deve ser escrita no dispositovo de saída padrão.

Restrições

- $1 \leq N \leq 100$
- $1 \leq C_1, C_2, C_3, C_4, C_5 \leq 13$

Exemplo de entrada

```
2
12 3 10 3 12
1 2 3 5 4
```

Exemplo de saída

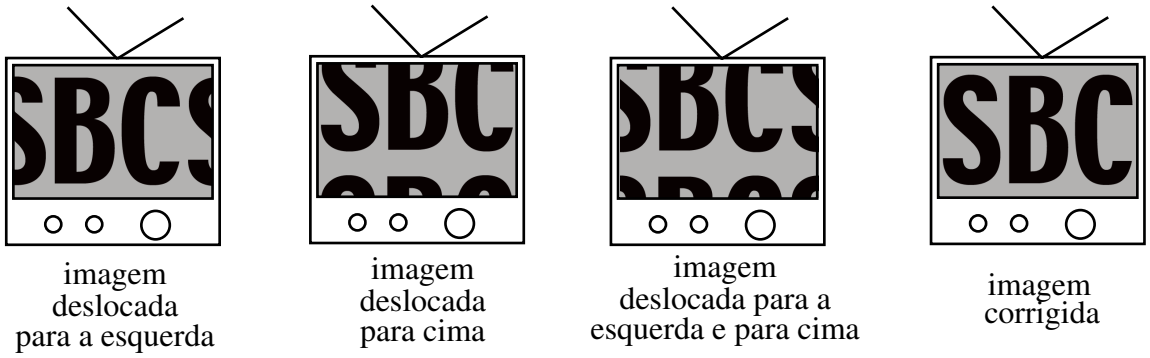
```
Teste 1
62

Teste 2
201
```

5 Problema E: TV da Vovó

`Arquivo: vovo.[c|cpp|java|pas]`

A vovó tem um televisor muito antigo, que ultimamente está exibindo um defeito incômodo: a imagem aparece "deslocada" (para cima ou para baixo, para o lado direito ou para o lado esquerdo). Quando a imagem está deslocada para cima, a parte da imagem que deixa de ser vista na parte superior reaparece na parte de baixo da tela. Da mesma forma, quando a imagem está deslocada a direita, a parte da imagem que deixa de ser vista à direita reaparece na tela do lado esquerdo.



A imagem do televisor pode ser vista como uma matriz de pontos organizados em linhas e colunas. Para consertar o televisor da vovó, você pode ajustar a imagem introduzindo uma série de "comandos de correção" em um painel de ajuste. Cada comando de correção desloca a imagem de um certo número de linhas (para cima ou para baixo) e um certo número de colunas (para a direita ou para a esquerda).

Tarefa

Dada uma matriz que representa uma imagem defeituosa e uma série de comandos de correção, seu programa deve calcular a matriz que representa a imagem resultante após todos os comandos terem sido aplicados sequencialmente.

Entrada

A entrada possui vários conjuntos de teste. Cada conjunto de teste inicia com a descrição da matriz que representa a imagem do televisor. A primeira linha contém dois inteiros M e N representando o número de linhas e o número de colunas da matriz ($1\ M \leq 1000$ e $1 \leq N \leq 1000$). As M linhas seguintes da entrada contém cada uma N inteiros, descrevendo o valor de cada ponto da imagem. Após a descrição da imagem, segue-se a descrição dos comandos de correção. Cada comando de correção é descrito em uma linha contendo dois inteiros X e Y. O valor de X representa o deslocamento na direção horizontal (valor positivo representa deslocamento para a direita, valor negativo para a esquerda), e o valor de Y representa o deslocamento da direção vertical (valor positivo para cima, valor negativo para baixo). O final da lista de comandos é indicado por $X = Y = 0$, e o final da entrada é indicado por $M = N = 0$.

Saída

Para cada conjunto de teste, o seu programa deve produzir uma imagem na saída. A primeira linha da saída deve conter um identificador do conjunto de teste, no formato "*Teste* n", onde n é numerado seqüencialmente a partir de 1. A seguir deve aparecer a matriz que representa a imagem resultante, no mesmo formato da imagem de entrada. Ou seja, as N linhas seguintes devem conter cada uma M inteiros que representam os pixels da imagem. Após a imagem deixe uma linha em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Restrições

- $0 \leq N \leq 1000$ ($N = 0$ apenas para indicar o final da entrada)
- $0 \leq M \leq 1000$ ($M = 0$ apenas para indicar o final da entrada)
- $0 \leq X \leq 1000$
- $0 \leq Y \leq 1000$
- $0 \leq$ número de comandos de correção em cada conjunto de teste ≤ 1000

Exemplo de Entrada

```
3 3
1 2 3
4 5 6
7 8 9
1 0
1 -1
0 0
3 4
6 7 8 5
10 11 12 9
2 3 4 1
-3 2
0 0
0 0
```

Exemplo de Saída

```
Teste 1
8 9 7
2 3 1
5 6 4

Teste 2
1 2 3 4
5 6 7 8
9 10 11 12
```

6 Problema F: *Passeios* de Lua de mel

`Arquivo: honey.[c|cpp|java]`

Emma está fazendo caminhadas pelas montanhas com Xande, seu maridão recém-casado, em sua lua de mel. Nestas caminhadas, eles vão de uma cabana até uma próxima pousada ou cabana, a cada dia. Infelizmente, Xande não está tão bem em forma como Emma e está começando a ficar cansado. Como Emma não quer começar o seu recém-casamento com nenhum conflito sério (e precisa de alguém para mantê-la aquecida nas noites), ela decidiu planejar as viagens dos dias seguinte para que elas não sejam tão extenuante para se novo maridão, nosso herói: Xande.

Nos últimos dias, Emma descobriu um fato surpreendente sobre seu marido. Ele não está tão cansado pela distância/comprimento de sua viagem diária ou pela quantidade total de metros que tiveram que escalar em suas caminhadas. Em vez disso, Xande está mais cansado, pela maior diferença entre o ponto mais alto e o mais baixo da rota do dia. Emma assume que isto é devido a fatores psicológicos. Assim, parece ser muito mais difícil escalar de 500 metros até 1.500 metros de altura de uma só vez, do que subir de 200 a 400 metros dez vezes, embora você tenha subido o dobro neste último caso.

O Problema

Dado um mapa de altitude do terreno, você deve ajudar Emma a encontrar um caminho que minimiza a diferença de altitude entre seus pontos mais alto e o mais baixo, de modo que Xande não se sinta tão cansado. A cabana de partida se localiza no canto superior–esquerdo e seu destino é o canto inferior–direito do mapa. Eles podem se mover em qualquer uma das quatro direções, mas não na diagonal.

Entrada

A primeira linha contém o número de cenários. Cada cenário começa com um número n ($2 \leq n \leq 100$), que representa o tamanho da área. As elevações do terreno são dadas como uma matriz $n \times n$ de inteiros ($h_{i,j}$) ($0 \leq h_{i,j} \leq 200$) nas n linhas, onde cada linha contém n elevações separadas por espaços.

Saída

A saída para todo cenário inicia com a linha contendo "`Scenario #i:`", onde i é o número do cenário, começando em 1. Em seguida, imprima uma única linha contendo a diferença entre a maior e a menor elevação do caminho ótimo. Termine a saída para cada cenário com uma linha em branco.

Exemplo de Entrada

```
1
5
1 1 3 6 8
1 2 2 5 5
4 4 0 3 3
8 0 2 2 4
4 3 0 3 1
```

Exemplo de Saída

```
Scenario #1:
3
```

7 Problema G: Facebook

`Arquivo: facebook.[c|cpp|java|pas]`

Rebecca acaba de entrar para o Facebook, uma rede social na internet. Como ela acabou de se registrar, ela ainda não possui muitos amigos na sua lista de contatos. Após fazer uma pesquisa, ela descobriu que os seus antigos amigos de escola (trolls que adoravam mexer com computadores) também fazem parte do Facebook. Rebecca então decidiu chamá-los para serem seus amigos. Porém, eles resolveram trollar com a Rebecca, e cada um deles só vai aceitar o pedido de Rebecca quando ela já for amiga de alguns dos outros amigos do grupo. Assim, para conseguir ter todos os seus antigos amigos de escola na sua lista de amigos do Facebook, ela deve cumprir as exigências de cada um deles.

Tarefa

Rebecca acha que pode encontrar uma seqüência de nomes dos amigos, de modo que se ela pedir a cada um deles para ser sua amiga no Facebook, obedecendo a seqüência, todas as exigências serão cumpridas e todos eles irão aceitar o seu pedido. Rebecca precisa da sua ajuda para resolver esse problema de forma rápida. A sua tarefa é escrever um programa para encontrar uma seqüência de nomes que resolva o problema (desbancando seus amigos trolls), ou dizer que não é possível encontrar tal seqüência (ser uma forever alone).

Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro N que indica o número de antigos amigos da Rebecca ($1 \leq N \leq 30$). A linha seguinte irá conter N nomes de amigos, separados por espaço em branco. Cada nome não terá mais de 15 letras, e serão todos distintos. Nas próximas N linhas serão indicadas as exigências que a Rebecca deve cumprir. Cada linha descreve a exigência de um amigo e começará com o nome desse amigo, seguido de um número M ($0 \leq M \leq N$–1), que indica o número de pessoas que aquele amigo quer que a Rebecca seja seguida antes, e M nomes de amigos (cada item na linha separado por espaço em branco). O final da entrada é indicado por $N = 0$.

Saída

Para cada conjunto de teste seu programa deve produzir três linhas na saída. A primeira linha deverá conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado seqüencialmente a partir de 1. A segunda linha deve conter a seqüência de nomes de amigos (cada nome seguido de um espaço em branco) que resolve o problema da Rebecca, ou a palavra "impossivel", quando não houver uma seqüência possível (note a ausência de acentuação). Se existir mais de uma seqüência de amigos que resolve o problema, imprima qualquer uma delas (mas apenas uma). A terceira linha deverá ser deixada em branco. A grafia mostrada no Exemplo de Saída abaixo deverá ser seguida rigorosamente.

Exemplo de Entrada

```
5
Joao Maria Tadeu Jose Ricardo
Joao 2 Maria Ricardo
Maria 1 Tadeu
Jose 1 Joao
Tadeu 0
Ricardo 0
3
Joao Maria Renata
Maria 1 Joao
Joao 1 Renata
Renata 1 Maria
0
```

Exemplo de Saida

```
Teste 1
Ricardo Tadeu Maria Joao Jose

Teste 2
impossivel
```