Maratona de Programação
1998
Caderno de Problemas
Categoria - Sênior

Problema 1: Piso

`Arquivo: piso.c ou piso.pas`
`Entrada: piso.in`
`Saída: piso.out`

Uma empresa que trabalha com instalação de pisos, freqüentemente precisa aplainar o cimento sobre o qual as peças são colocadas. Esta tarefa pode ser feita com auxílio de um tipo de prensa pesada e de difícil deslocamento. Várias pessoas trabalham na instalação de um piso e, durante esse serviço, algumas delas necessitam de uma destas prensas para fazer um algum ajuste na instalação de um dos azulejos. Quando uma pessoa necessita de uma prensa, uma delas é levada à posição onde a pessoa está. Obviamente, enquanto a prensa não chega, o trabalhador está parado, esperando.
Na tentativa de minimizar o tempo de espera dos trabalhadores, a empresa pensou em uma regra de controle do movimento das prensa. A regra consiste no seguinte. Para cada requisição do uso de uma prensa, a que está mais próxima ao ponto de requisição é escolhida para ser levada. Cada prensa é identificada por um número de 1 a k e, em caso de haver duas prensas à mesma distância do trabalhador que fez a requisição, a prensa de número menor é escolhida para ser levada. Após usar a prensa esta é deixada nesta posição, esperando uma possível nova requisição para então ser transportada.
Você deve escrever um programa que recebe o número k de prensas, a posição inicial de cada uma delas e uma sequência de requisições. Seu programa deve calcular a distância total percorrida pelas prensas para atender a estas requisições de acordo com a regra descrita acima.
Uma posição inicial consiste de um par (x, y) de números reais, indicando um ponto no plano. Uma requisição também consiste de um par (x, y) de números reais, para onde uma prensa deve ser levada.

Entrada

A primeira linha do arquivo de entrada contém o número k de prensas. A seguir, nas k próximas linhas são dadas as posições das k prensas. A partir daí, cada linha contém a posição de uma requisição, até o fim do arquivo (`end-of-file`).

Saída

A saída consiste de um único número real: a distância total percorrida pelas prensas para atender a todas as requisições de acordo com a regra descrita acima. Você deve imprimir este número com duas casas decimais.

Exemplo

Entrada

```
3
1 1
2 5
-1 0
3 4
0 0.5
1 1
-1 0
```

Saída

```
3.65
```

Problema 2: Boliche

`Arquivo: boliche.c ou boliche.pas`
`Entrada: boliche.in`
`Saída: boliche.out`

Boliche é um dos jogos mais divertidos que existem, já que não exige grande experiência, força física, atenção ou dedicação. Os resultados são mais ou menos aleatórios, independentemente do jogador (já deu para perceber que não me dou muito bem no esporte, certo?). Um dos maiores desafios do jogo (maior do que derrubar os malditos pinos) é descobrir como marcar corretamente os pontos.
Para fazer isso, em geral as casas de boliche dispõem de computadores que fazem as anotações e marcações automaticamente, e o resultado se torna um completo mistério para todos os jogadores, que acreditam no programa. Entretanto, as regras para marcar pontos são bastante simples. Um jogo de boliche é composto de dez partidas. Em cada partida o jogador dispõe de duas bolas para derrubar os dez pinos. Se ele derrubar todos os pinos com a primeira bola fez um *strike* e o número de pontos que ele ganha nesta partida é dado por dez mais o número de pinos que ele derrubar em suas duas próximas bolas. Já se o jogador conseguir derrubar os dez pinos nas suas duas bolas, ele faz um *spare*, e o número de pontos na partida será dado por dez mais o número de pinos derrubados na próxima bola que ele arremessa. Caso o jogador derrube menos de dez pinos com as duas bolas, o número de pontos na partida será dado pelo número de pinos derrubados. Note que a décima partida é especial. Se o jogador derrubar todos os pinos na primeira bola, terá direito a jogar duas bolas extras. Senão, se ele fizer um *spare* (ou seja, derrubar os dez pinos na segunda bola) tem direito a uma bola extra.
Sua tarefa é fazer um programa que lê os resultados obtidos durante vários jogos de boliche, e dizer, para cada jogo, o número de pontos que cada um dos jogadores deste jogo fez.

Entrada

Cada um dos jogos é composto por diversas linhas. A primeira linha conterá o número de jogadores neste jogo ($0 < n \leq 6$). A seguir, para cada jogador, é dada uma linha, com o seu nome (composto de no máximo 10 letras minúsculas) e os resultados deste jogador. A última linha do arquivo conterá um zero.

Saída

Para cada jogo você deverá imprimir a mensagem `JOGO #1`, `JOGO #2`, etc, e a seguir o número de pontos que cada jogador fez (um jogador por linha). Deverá haver uma linha separando os resultados de cada jogo.

Exemplo

Entrada

```
3
dilma 1 1 0 0 2 1 0 3 10 10 10 10 0 0 0 0
carlinhos 10 10 10 10 10 10 10 10 10 10 10 10
coelho 10 2 8 3 4 5 3 4 4 2 8 4 6 8 1 9 1 10 3 4
1
jose 7 3 10 10 10 10 0 0 0 0 0 4 8 1 9 0
0
```

Saída

```
JOGO #1
dilma fez 98 pontos
carlinhos fez 300 pontos
coelho fez 134 pontos

JOGO #2
jose fez 132 pontos
```

Problema 3: Agência de Viagens

```
Arquivo:  agencia.c ou agencia.pas
Entrada:  agencia.in
Saída:  agencia.out
```

A estratégia da agência de viagens *USP Passeios* para ficar mais conhecida no mercado é oferecer de graça informação para os turistas que estão estudando a possibilidade de viajar de carro. Estes clientes em potencial podem telefonar para a agência e requisitar informação sobre o trajeto desejado; se a agência dispõe de informação sobre tal trajeto ela determina os mapas rodoviários envolvidos na viagem e os envia ao cliente pelo correio.
O escritório da *USP Passeios* não é muito grande, portanto frequentemente a agência revisa o conjunto de trajetos para os quais se propõe a guardar informação. Os trajetos menos populares são eliminados, e os mapas consequentemente não mais necessários descartados. Sua tarefa é escrever um programa que ajuda a *USP Passeios* nesta sua faxina periódica, ou seja, a partir das descrições dos trajetos disponíveis e da lista de trajetos a serem eliminados, indica quais mapas podem ser descartados.

Entrada

A entrada é composta por uma seqüência de descrições de trajetos (o número de trajetos é menor que 100). Em seguida vem uma seqüência de trajetos a serem eliminados:

```
<descricao do trajeto-1>
<descricao do trajeto-2>
...
<descricao do trajeto-n>
ELIMINA t1 t2 ... tk
```

onde

- a descrição do trajeto-i é formada pelo nome do trajeto (uma palavra, ou seja, uma seqüência de caracteres em que não aparece um espaço em branco), o número k ($0 < k \leq 20$) de mapas que compõem este trajeto e uma lista de inteiros que identifica os mapas, onde cada inteiro é um número entre 0 e 35000;
- t1, t2, ..., tk são nomes de trajetos.

Saída

A saída deve ser uma seqüência de inteiros representando a lista de mapas a serem descartados. Se nenhum mapa pode ser descartado, a saída deve ser a palavra NENHUM.

Exemplo 1

Entrada

```
cotia 1 1
mairipora 3 1 2 4
caxambu  3  3 2 5
ELIMINA cotia mairipora
```

Saída

```
1 4
```

Exemplo 2

Entrada

```
rio-maravilha 3 10001 212 31400
beto-carrero 3 4 5 212
porto-seguro 6 1 2 3 6 700 10001
buzios 5 1 209 3 8 31400
ELIMINA rio-maravilha
```

Saída

```
NENHUM
```

Problema 4: Truco

Arquivo: `truco.pas` e `truco.c`
Entrada: `truco.in`
Saída: `truco.out`

Organizar campeonatos não é uma tarefa completamente trivial. Especialmente um campeonato de truco, jogo conhecido por suas regras estranhas e jogadores mais estranhos ainda.
Depois de um longo campeonato de truco, com vários jogos, a comissão organizadora ficou na dúvida sobre se havia ou não um campeão, ou mesmo se as equipes podiam ser classificadas em alguma ordem (ainda que permitindo empates). Neste ponto eles resolveram fazer um programa para solucionar o problema.
Dados os resultados de diversas partidas entre as equipes concorrentes, o programa deverá decidir se há ou não uma classificação clara entre as equipes (eventualmente algumas empatadas), ou se esta classificação é impossível. Ao invés de atribuir pontos para vitórias (já que o número de partidas de cada equipe pode ser diferente), a comissão organizadora resolveu que se uma equipe A venceu a equipe B em uma partida, a classificação de A deve ser necessariamente melhor do que a classificação de B.
Seu programa deve ler os resultados de vários campeonatos de truco, determinando, para cada um deles se há ou não uma classificação (seguindo o critério da comissão organizadora) e, caso haja, qual é essa classificação.

Entrada

São dados vários campeonatos de truco. Cada campeonato é dado por várias linhas. A primeira linha contém o número de equipes participantes ($0 < n \leq 10$) e o número de jogos realizados por estas equipes ($0 < m \leq 50$). A seguir, vêm os m resultados, cada um em uma linha, com um par i, j com o significado de que a equipe i ganhou o jogo de j. Após o último campeonato aparece um zero.

Saída

Para cada campeonato você deverá imprimir uma mensagem com o número do campeonato (`CAMPEONATO #1`, etc). A seguir vem a classificação deste campeonato. Caso não haja uma classificação possível, seu programa deverá imprimir a mensagem `Nao ha classificacao possivel`. Caso a classificação seja possível seu programa deverá imprimir em cada linha a classificação (`1. lugar`, `2. lugar`, etc) e os números das equipes com tal classificação. Deverá haver uma linha em branco separando os campeonatos.

Exemplo

Entrada

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

Saída

```
CAMPEONATO #1
1. lugar 1 2
3. lugar 3 4
5. lugar 5

CAMPEONATO #2
Nao ha classificacao possivel
```

Problema 5: Torres

`Arquivo: torres.c ou torres.pas`
`Entrada: torres.in`
`Saída: torres.out`

Considere o problema de determinar o número de maneiras de dispormos n torres de um jogo de xadrez em um tabuleiro de dimensão $n \times n$ de tal forma que quaisquer duas torres não se ataquem. (Ou seja cada linha e cada coluna do tabuleiro deve conter exatamente uma torre.) No tabuleiro teremos ainda algumas posições *proibidas* que são posições onde não pode ser colocada nenhuma torre. Para os tabuleiros da Figura 1 as posições hachuradas indicam posições proibidas. Assim, por exemplo, no tabuleiro da Figura 1(a) existem 4 maneiras

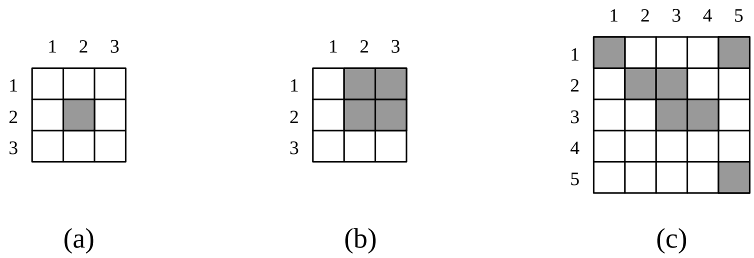

Figura 1: Exemplos de tabuleiros com posições proibidas.

para dispormos 3 torres, as maneiras são:
maneira 1: torres colocadas nas coordenadas (1,1), (2,3) e (3,2);
maneira 2: torres colocadas nas coordenadas (1,2), (2,1) e (3,3);
maneira 3: torres colocadas nas coordenadas (1,2), (2,3) e (3,1);
maneira 4: torres colocadas nas coordenadas (1,3), (2,1) e (3,2).

No tabuleiro da Figura 1(b) existem 0 (zero) maneiras de dispormos 3 torres. Finalmente, no tabuleiro da Figura 1(c) existem 25 maneiras de dispormos 5 torres de tal forma que quaisquer duas não se ataquem.

Entrada

A primeira linha do arquivo de entrada contém a dimensão do tabuleiro (você pode supor que a dimensão é menor que 10). O resto do arquivo de entrada contém uma lista de pares de números inteiros indicando as coordenadas das posições proibidas, uma coordenada por linha do arquivo. O arquivo de entrada é terminado por um `end-of-file`.

Saída

A saída do seu programa deve ser a mensagem `O numero de maneiras de dispormos` n `torres no tabuleiro e` k, onde n indica a dimensão do tabuleiro e k indica o número de maneiras de dispormos n torres no tabuleiro de tal forma que: (i) quaisquer duas não se ataquem; (ii) nenhuma torre seja colocada em uma posição proibida.

Exemplo

Entrada

```
3
2 2
```

Saída

```
O numero de maneiras de dispormos 3 torres no tabuleiro e 4.
```

Problema 6: Haras

```
Arquivo:  haras.c ou haras.pas
Entrada:  haras.in
Saída:  haras.out
```

Cavalos e cavaleiros são cheios de manias. Nem sempre um cavaleiro se adapta às características de um determinado cavalo, tornando o par (ou conjunto, como se diz no esporte) um desastre em qualquer torneio. O dono do haras deve tentar utilizar ao máximo seus cavalos e cavaleiros, de forma a não ter nenhum par incompatível. Muitas vezes não é possível encontrar conjuntos adequados, se os gostos dos cavaleiros forem muito restritivos.

Sua tarefa será, a partir dos dados de vários haras, determinar para cada um deles conjuntos que devem participar dos torneios. Para cada haras são dados n cavalos e n cavaleiros, e os cavalos que cada cavaleiro poderá montar. Você deverá então imprimir (se existir) n conjuntos (ou seja, pares cavaleiros, cavalos) em que os cavalos designados para cada cavaleiro estão entre os que ele poderá montar. É claro que cada cavalo (e cavaleiro) deverá aparecer em exatamente um conjunto, e o cavalo designado deve estar entre as opções do cavaleiro.

A entrada

São dados vários haras. Para cada haras é dado o número n ($0 < n \leq 20$) de cavalos e cavaleiros deste haras. A seguir, para cada cavaleiro é dada uma linha com o seu nome (composto apenas por letras minúsculas), o número de cavalos que ele pode montar e os nomes destes cavalos (também compostos apenas de letras minúsculas). Após os dados do último haras virá um zero.

A saída

Para cada haras você deverá imprimir uma linha identificando o mesmo (`HARAS #1`, etc). A seguir, deverá vir cada um dos n conjuntos deste haras, um por linha, com o nome do cavaleiro seguido pela palavra `monta` e o nome do cavalo que ele deverá montar. No caso de não existirem n conjuntos adequados, seu programa deverá imprimir a mensagem `Nao existem conjuntos adequados para todos os cavalos e cavaleiros`. Deverá haver uma linha em branco entre os resultados de cada haras.

Exemplo

Entrada

```
5
pessoa  3 sunshine bluesky ventania
jose  2 ventania aloha
carlinhos  1 sunshine
dilma  3 ventania aloha sunshine
```

```
coelho  2 aloha darkblue
4
pessoa  2 aloha bluesky
jose 2 bluesky aloha
carlinhos  1 bluesky
dilma 2 ventania sunshine
0
```

Saída

```
HARAS #1
pessoa monta bluesky
jose monta ventania
carlinhos monta sunshine
dilma monta aloha
coelho monta darkblue

HARAS #2
Nao existem conjuntos adequados para todos os cavalos e cavaleiros
```