Universidade do Estado Santa Catarina – UDESC

Universidade Federal do Paraná – UFPr

UNICENTRO – Guarapuava – Pr.
Maratona Doméstica de Programação – 2008-2

(Lurdes)

Organizadores: Claudio (DCC), André Guedes (DInf)
e Eleandro (UNICENTRO)

3 de setembro de 2008

Lembrete:

- É permitido consultar livros, anotações ou qualquer outro material impresso cópia durante a prova.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa.
- Todos os problemas têm o mesmo valor na correção. Logo, leia e comece com os mais fáceis. Para esta prova, sugiro que comece com o **H** (agah).
- Procure resolver o problema de maneira eficiente. Na correção, a eficiência também é levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Nas implementações, as entradas e as saídas conhecidas serão padrão (ler e escrever na saída padrão).
- Utilize o *Clarification* para dúvidas da prova. Opcionalmente, os juízes respondem, contudo, os esclarecimentos são acessíveis a todos.
- A novidade é o nome dos programas que seguem um padrão no sufixo, definido por uma letra, relativo ao problema submetido. Exemplo: `montecrista_e.c`, neste caso, refere-se ao problema '**e**'.

Agradecimentos

- Ao pessoal do suporte da UDESC, Grupo Colméia, C^3SL da UFPr.
- Ao DCC e demais anônimos que tornaram este evento uma realidade.
- Pelas traduções: Fernando Sasse, Rosso, Marcelo e o Wanderley

Problema A: Subconjuntos de Alfabetos

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

Palavras são anagramas umas das outras se ela têm o mesmo comprimento e contêm as mesmas letras. Por exemplo, ‘pots’, ‘stop’, ‘spot’ e ‘opts’ são anagramas. Palavras compartilham o mesmo alfabeto base se elas têm todas as suas letras em comum, mesmo que algumas delas sejam repetidas. Portanto, ‘curse’ e ‘rescue’ compartilham o mesmo alfabeto base, mas ’cure’ não compartilha um alfabeto base com tais palavras, pois não tem um ’s’.

Escreva um programa que leia repetidamente pares de palavras e determina se elas compartilham um alfabeto base ou não.

A entrada deve consistir de uma série de linhas, cada uma contendo um única palavra com menos de 20 caracteres minúsculos. Sucessivas linhas devem formar os pares de palavras. O fim da entrada deve ser sinalizado por uma linha contendo um único ’#’.

A saída deve consistir de uma única linha para cada par de palavras. A primeira palavra do par deve começar na linha 1, a segunda palavra deve começar na coluna 21 e partindo na coluna 41 a palavra ’yes’ ou ’no’, dependendo das palavras formarem um alfabeto base ou não.

Exemplo de Entrada

```
curse
rescue
curse
cure
#
```

Exemplo de Saída

```
curse                rescue               yes
curse                cure                 no
```

Problema B: Rotações e Reflexões

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

Muitos jogos, brincadeiras e quebra-cabeças dependem em determinar se dois padrões sobre uma grade retangular são os "mesmos" ou não. Por exemplo, no clássico problema da 8 rainhas, temos 96 maneiras diferentes de arranjar estas 8 rainhas seguramente (sem que nenhuma se ataque mutuamente) sobre um tabuleiro de xadrez. Contudo, estas 96 soluções podem ser mostradas que consistem de rotações e/ou reflexões a partir de apenas 12 padrões básicos.

Escreva um programa que leia pares de padrões e determine se existe uma simples transformação que converterá um no outro. Devido os padrões simétricos conhecidos há muitos relacionamentos um com o outro, e as transformações que devem ser verificadas seguem numa específica ordem/seqüência. As possíveis transformações (em ordem) são dadas por:

Preservar	Os padrões são idênticos
Rotação de 90 graus	O padrão foi rotacionado de 90 graus no sentido horário
Rotação de 180 graus	O padrão foi rotacionado de 180 graus no sentido horário
Rotação de 270 graus	O padrão foi rotacionado de 180 graus no sentido horário
Reflexão	O padrão foi refletido sobre o eixo horizontal (efetivamente como um espelho preso na parte superior do padrão)
Combinação	Uma reflexão (acima descrito), seguido por uma das rotações acima
Impróprio	Os padrões não casam sob nenhuma das transformações acima

A entrada consistirá de uma série de pares de padrões. Cada conjunto consistirá de uma linha contendo um simples inteiro `N` ($2 \leq N \leq 10$) o qual fornece o tamanho do padrão, seguido pelas `N` linhas do padrão. Cada linha consistirá de `N` pontos (`'.'`) ou `'x'`s (especificando uma linha do padrão original), em seguida um espaço, e um outro conjunto de `N` pontos (`'.'`) e `'x'`s (especificando uma linha do padrão transformado). Isto é, o início do segundo padrão a ser comparado. O arquivo terminará por uma linha consistindo de um simples zero (`0`).

A saída consistirá de uma série de linhas, uma para cada par de padrão da entrada. Cada linha consistirá em uma das seguintes mensagens como saída: '`Preserved`', '`Rotated through m degrees`' (onde m é um valor de 90, 180 ou 270), '`Reflected`', '`Reflected and rotated through m degrees`', '`Improper`'. Os termos permanecem em inglês para não alterarmos os arquivos originais.

Exemplo de Entrada

```
5
x...x ....x
.x... ...x.
...x. .x...
..x.x ..x..
....x xx..x
2
```

```
x. xx
x. xx
4
..x. ...x
xx.. ....
.... xx..
...x ..x.
4
.x.. ..x.
.x.x x...
.... ..xx
..x. ....
0
```

Exemplo de Saída

```
Rotated through 90 degrees
Improper
Reflected
Reflected and rotated through 270 degrees
```

Problem C: Fatores e Fatoriais

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

O fatorial de um número N (escrito $N!$) é definido como o produto de todos os inteiros de 1 até N. Isto freqüentemente é definido recursivamente. Com a seguir:

$$1! = 1$$

$$N! = N * (N - 1)!$$

Fatoriais crescem muito rápido —5! = 120, 10! = 3,628,800. Uma maneira de especificar números tão grandes é especificando a quantidade de vezes que um número primo ocorre neles, assim, 825 pode ser especificado como (0 1 2 0 1) significando: nenhum dois (indicado pelo primeiro 0), 1 três , 2 cincos, nenhum sete e 1 onze. Em resumo, é o teorema fundamental da aritmética: $825 = 3^1 \times 5^2 \times 11^1$.

Escreva um programa que irá ler um número N ($2 \leq N \leq 100$) e escreva o seu fatorial em termos dos números primos que ele contém. Por exemplo: $5! = 2^3 \times 3^1 \times 5^1$.

A entrada consiste de uma série de linhas, cada linha contendo um único inteiro N. O arquivo terminará com uma linha contendo apenas um único 0.

A saída consistirá de uma série de blocos de linhas, com um bloco para cada linha de entrada. Cada bloco iniciará pelo número N, justificado (alinhado) a direita em um campo de tamanho 3, e os caracteres '`!`', espaço (isto, espaço em branco mesmo), e '`=`'. Isto será seguido por uma lista do número de vêzes que cada primo ocorre em N!.

Estes números devem ser justificados à direita em campos de tamanho 3 e cada linha (exceto a última que pode ser mais curta), deve conter quinze números. Qualquer linha após a primeira deve ser indentada. Siga o *layout* do exemplo abaixo, exatamente.

Exemplo de Entrada

```
5
53
0
```

Exemplo de Saída

```
  5! =  3  1  1
 53! = 49 23 12  8  4  4  3  2  2  1  1  1  1  1  1
        1
```

Problem D: Seqüência Maximal

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

Dada uma seqüência de N inteiros de ($1 \leq N \leq 100$), a *seqüência maximal* (para o propósito deste problema) é a sub-seqüência com a maior soma. Portanto, para a seqüência:

1, 2, 8, -7, 3, 5, -20, 2, 1

Uma sub-seqüência maximal vai (ou segue) do primeiro ao sexto valor com a soma igual a 12, uma vez que nenhuma outra seqüência destes números gera uma soma maior que esta.

Escreva um programa que leia uma seqüência valores e encontre a sub-seqüência maximal. Seu programa deve ser capaz de produzir uma resposta em tempo aceitável[1], mesmo para uma seqüência de 100 números.

A entrada será dada pelo número N numa linha e, na linha subseqüente, os respectivos N números. Haverá um conjunto destas entradas cujo fim do conjunto será indicado quando ocorrer um N = 0. Todas as seqüências de entrada produzirão valores de seqüência maximal positivos.

A saída consistirá nos índices de início e fim da seqüência e da respectiva soma para cada seqüência dada, todos justificados à direita em campos de 10 caracteres, um resultado por linha.

Se houver mais de uma seqüência maximal então, a mais curta (aquela contendo o menor número de elementos) deve ser mostrada.

Se houver mais de uma seqüência maximal de tamanhos mínimos então, aquela com o menor índice de início deve ser mostrada.

Exemplo de Entrada

```
11
1  2  8  -7  3  5  -20  2  1 -2 -3
4
-10 105 6 -2
0
```

Exemplo de Saída

```
         1         6         12
         2         3        111
```

[1] Em caso tempo excedido a 5 segundos de CPU, o André avaliará a resposta e talvez dê mais tempo de CPU ao problema.

Problem E: Monte Crista

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

Nas proximidades de Joinville há o famoso caminho dos Peabiru, ou Tape Aviru - que ligava o Atlântico e o Pacífico. Resquícios desse caminho pré-hispânico são encontrados em pontos descontínuos nos estados de Santa Catarina, do Paraná, de São Paulo, no Paraguai, na Bolívia e no Peru. E pelo que dizem, a conhecida Trilha Inca faz parte desse caminho.

A trilha de acesso começa no morro Monte Crista. Os poucos remanescentes em pedra contam uma história notável: pessoas, cargas, sonhos partiam da região de São Francisco do Sul (ou de Iguape/Cananéia, no litoral paulista) e subiam a serra na direção do interior da América, articulando civilizações da costa atlântica às do altiplano andino e da costa pacífica.

Dizem, que próximo ao acesso final deste altiplano há uma escadaria feita de pedras. Nestas pedras encontraram-se algumas inscrições, as quais codificavam mensagens entre os jesuítas que ali utilizavam a trilha.

Um grupo de ecologistas, em visita ao morro Monte Crista encontrou uma destas inscrições codificadas, e como havia um pesquisador/estudioso em códigos neste grupo, este decifrou o código. A euforia foi geral, a imprensa foi chamada, o *circo foi armado*, para saberem como os ecologistas tinham decodificado as inscrições.

De acordo com o pesquisador de código, o alfabeto jesuíta era composto pelas letras de A até Z (incluindo as letras e K, W e Y). A codificação era realizada da forma que segue. Inicialmente, era construída uma lista em que a letra A aparecia na primeira posição, a letra B aparecia na segunda, e assim sucessivamente, com as letras seguindo a mesma ordem que em nosso alfabeto. Em seguida, o texto a ser codificado era varrido da esquerda para a direita e, para cada letra l encontrada, o número de sua posição na lista era impresso e l (a letra) era movida para o início da lista. Por exemplo, a codificação jesuíta para a mensagem:

ABBBAABBBBACCABBAAABC

era dada pela seguinte seqüência de inteiros:

12112121112312312 1123

Os ecologistas pediram a sua ajuda para construir um programa que receba uma seqüência de inteiros que representa uma mensagem codificada e decodifica-a.

Aviso: voce só deve ir lá no inverno, preservando a ecologia do local e preparado, pois um número significativo de cobras jararacas procuram ajudar a preservar este lugar.

Entrada

Seu programa deve estar preparado para trabalhar com diversas instâncias. Cada a instância tem a estrutura que segue. Na primeira linha fornecido um inteiro m ($0 \leq m \leq 10000$) que representa o número de inteiros que compõem o texto codificado. Na próxima linha são dados, separados por espaços em branco, os m valores inteiros (cada valor é maior ou igual a 1 e menor ou igual a 26). Um valor $m = 0$ indica o final das instâncias e não deve ser processado. Mas a linha em branco já foi impressa!

Saída

Para cada instância solucionada, você dever imprimir um identificador `Instancia h` em que `h` é um número inteiro, seqüencial e crescente a partir de 1. Na linha seguinte, você deve imprimir o texto decodificado. Uma linha em branco deve separar a saída de cada instância.

Exemplo de Entrada

```
21
1 2 1 1 2 1 2 1 1 1 2 3 1 2 3 1 2 1 1 2 3
5
22 6 8 4 15
3
24 1 1
26
22 10 6 4 13 16 16 12 5 1 4 20 1 21 21 5 10 7 16 6 15 12 5 3 8 9
0
```

Exemplo de Saída

```
Instancia 1
ABBBAABBBBACCABBAAABC

Instancia 2
VEGAN

Instancia 3
XXX

Instancia 4
VIDALONGAAOSSTRAIGHTEDGERS
```

Problem F: O Penúltimo Imperador

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

Muito se conhece do último imperador da China, imortalizado no clássico filme vencedor do Oscar. Porém, seu antecessor, o Imperador Thang Po Lop teve uma vida muito mais interessante, uma vez que morreu ainda na cidade proibida, cercado de concubinas e criados eunucos. O Imperador Po Lop era um grande colecionador de pauzinhos (daqueles que os orientais utilizam para comer). Desde seus 9 anos ele os guardava e construía com eles enormes labirintos utilizando uma estratégia bastante interessante. Inicialmente Po Lop escolhia e um dos pátios retangulares da cidade proibida para construir o labirinto, e esse labirinto a sempre ocupava todo o espaço do pátio escolhido. Os pauzinhos eram então colocados nesse pátio aparentemente em lugares aleatórios, sempre paralelos a um dos cantos do pátio. O imperador nunca colocava pauzinhos sobrepostos (nem mesmo parte deles), **apesar de ser possível existirem cruzamentos ou até mesmo pauzinhos se encostando**. Consta na biografia do imperador Po Lop que ele construiu labirintos gigantescos, sempre tomandos esses cuidados.

Infelizmente havia um problema. Apesar de exímio construtor de labirintos, o imperador era incapaz de saber se afinal o labirinto continha ou não um caminho ligando sua a entrada a sua saída (sempre em lados opostos do pátio). Para saber isso, ele se utilizava a ida a de seus eunucos. Ele instruía o eunuco a procurar o caminho naquele labirinto. Muitas ia vezes, o eunuco dizia não ser possível. O imperador Po Lop se zangava e degolava o infeliz, pois duvidava da resposta do criado. Felizmente, além de muito paciente (não com e a eunucos) o imperador era bastante cuidadoso, e anotava criteriosamente as informações sobre os labirintos que construía. Estas anotações foram encontradas na biblioteca da cidade proibida quando da revolução e salvas da destruiçãoo. Sua tarefa neste problema é resolver finalmente o enigma, verificando se os labirintos construídos pelo Imperador Po Lop têm ou não saída.

Entrada

Seu programa deve estar preparado para trabalhar com diversos labirintos, doravante denominados instâncias. Cada instância iniciada com uma linha contendo 5 números, ditos $n\ x_i\ y_i\ x_f\ y_f$. O valor n indica o número de pauzinhos que foram usados para construir o labirinto. O par (x_i, y_i) o canto inferior esquerdo do pátio e também o ponto de partida. O par (x_f, y_f) o canto superior direito e também ponto de chegada do e e labirinto. Nas próximas n linhas são dadas as coordenadas $x_1\ y_1$ $x_2\ y_2$ representando os extremos (x_1, y_1) e (x_2, y_2) de um dos pauzinhos usados na construção do labirinto. O arquivo de entrada termina com $n < 0$. Pode-se assumir que todos os números dados são a inteiros e que $n \leq 1000$.

Saída

Para cada instância solucionada, você deverá imprimir um identificador `Instancia h` em que `h` é um número inteiro, seqüencial e crescente a partir de 1. Na linha seguinte, voce deve imprimir `sim` se existir uma maneira de ir do ponto de partida do labirinto até seu ponto de chegada (sem atravessar nenhum pauzinho ...) e imprimir `nao` em caso contrário.

Um linha em branco deve separar a saída de cada instância.

Exemplo de Entrada

```
1 0 0 2 2
0 1 1 1
```

```
2 0 0 2 2
1 2 1 0
0 1 2 1
-1
```

Exemplo de Saída

```
Instancia 1
sim

Instancia 2
nao
```

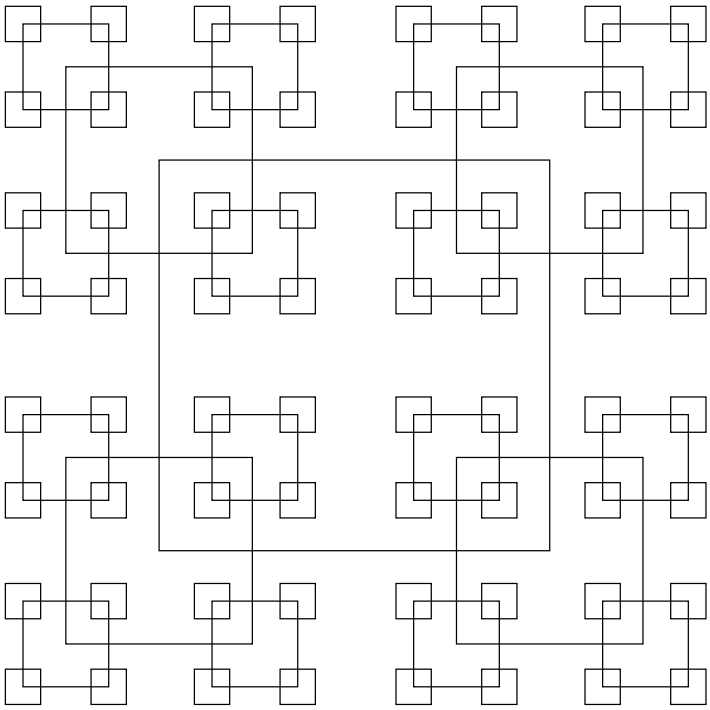
Problem G: Quadrados

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

Geometricamente, qualquer quadrado tem um único, bem definido ponto central. Em uma rede isso é verdade somente se os lados do quadrado são um número ímpar de pontos em comprimento. Como cada número ímpar pode ser escrito na forma $2k+1$, podemos caracterizar qualquer quadrado destes especificando k. Ou seja, podemos dizer que um quadrado cujos lados são de comprimento têm tamanho k. Definimos um padrão de quadrados como segue:

1. O maior quadrado é de tamanho k (lados têm tamanho $2k + 1$) e é centrado em uma rede de tamanho 1024 (os lados da rede têm tamanho 2049).
2. O menor quadrado permissível é de tamanho 1 e o maior é de tamanho 512, de modo que $1 \leq k \leq 512$.
3. Todos quadrados de tamanho $k > 1$ têm um quadrado de tamanho `k div 2` centrado em cada um de seus 4 vértices. (div significa divisão inteira, por exemplo, `9 div 2 = 4`).
4. O vértice superior esquerdo da tele tem coordenadas `(0,0)` e o inferior esquerdo tem coordenadas `(2048,2048)`.

Portanto, dado um valor de k, podemos desenhar um único padrão de quadrados, de acordo com as regras acima. Além disso, qualquer ponto da tela será cercado por zero ou mais quadrados. (Se o ponto está na borda de um quadrado, ele é considerado com cercado por este quadrado). Portanto, se o lado do maior quadrado é 15, então o seguinte padrão será formado:



Escreva um programa que lerá um valor de k e as coordenadas de um ponto, determinando quantos quadrados contêm o ponto.

A entrada consistirá de uma série de linhas. Cada linha conterá um valor de k e a coordenadas de um ponto. O arquivo será terminado por uma linha consistindo de três zeros `(0,0,0)`.

A saída consistirá de uma série de linhas, uma para cada linha da entrad. Cada linha consistirá do número de quadrados contendo o ponto específico, justificado à direita em um campo de largura 3.

Exemplo de Entrada

```
500 113 941
0 0 0
```

Exemplo de Saída

```
  5
```

Problem H: Retângulos

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

Meu filho está com dificuldades no trabalho de casa de matemática. Por favor, vocês poderiam ajudá-lo? Ele está trabalhando com a área do retângulo vocês sabem que, $area = comprimento \times largura$. O professor dele, forneceu uma tabela de comprimentos, larguras e áreas. Em cada linha da tabela falta um dos 3 valores; meu filho tem que calcular o valor que falta em cada linha e escrevê-lo na linha de forma que os valores em cada linha representem o comprimento, largura e área de um retângulo.

A entrada é uma série de linhas, cada uma contendo 3 inteiros, l (comprimento), w (largura) e a (área) (sendo $0 \leq l$ e $w \leq 100$, $0 \leq a \leq 10.000$) representando o comprimento, a largura e a área do retângulo, nesta ordem. Os inteiros são separados por um espaço. Em cada linha apenas um valor é substituído por um zero. A última linha contém `0 0 0` e não deve ser processada.

A saída é a mesma série de linhas mas com o zero de cada uma substituído pelo valor correto para o comprimento, largura ou a área, o que for apropriado. O novo valor será sempre um inteiro.

Exemplo de Entrada

```
2 0 6
6 5 0
0 8 80
9 0 45
0 0 0
```

Exemplo de Saída

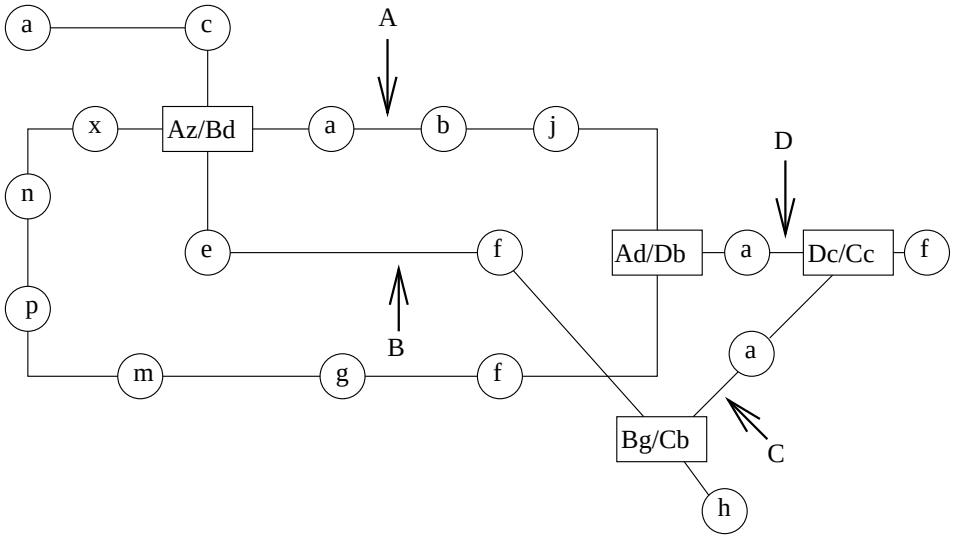
```
2 3 6
6 5 30
10 8 80
9 5 45
```

Problem Z: Route Finding

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

Many cities provide a comprehensive public transport system, often integrating bus routes, suburban commuter train services and underground railways. Routes on such systems can be categorised according to the stations or stops along them. We conventionally think of them as forming lines (where the vehicle shuttles from one end of the route to the other and returns), loops (where the two ends of the "branch" are the same and vehicles circle the system in both directions) and connections, where each end of the route connects with another route. Obviously all of these can be thought of as very similar, and can connect with each other at various points along their routes. Note that vehicles can travel in both directions along all routes, and that it is only possible to change between routes at connecting stations.

To simplify matters, each route is given a designation letter from the set 'A' to 'Z', and each station along a route will be designated by another letter from the set 'a' to 'z'. Connecting stations will have more than one designation. Thus an example could be:



A common problem in such systems is finding a route between two stations. Once this has been done we wish to find the "best" route, where "best" means "shortest time".

Write a program that will read in details of such a system and then will find the fastest routes between given pairs of stations. You can assume that the trip between stations always takes 1 unit of time and that changing between routes at a connecting station takes 3 units of time.

Input will consist of two parts. The first will consist of a description of a system, the second will consist of pairs of stations. The description will start with a number between 1 and 26 indicating how many routes there are in the system. This will be followed by that many lines, each describing a single route. Each line will start with the route identifier followed by a '`:`' followed by the stations along that route, in order. Connections will be indicated by an '`=`' sign followed by the complete alternative designation. All connections will be identified at least once, and if there are more than two lines meeting at a connection, some or of all the alternative designations may be identified together,. That is, there may be sequences such as '…hc=Bg=Cc=Abd…'. If the route forms a loop then the last station will be the same as the first. This is the only situation in which station letters will be repeated. The next portion of the input file will consist of a sequence of lines each containing two stations written contiguously. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each pair of stations in the input. Each line will consist of the time for the fastest route joining the two stations, right justified in a field of width 3, followed by a colon and a space and the sequence of stations representing the shortest journey. Follow the example shown below. Note that there will always be only one fastest route for any given pair of stations and that the route must start and finish at the named stations (not at any synonyms thereof), hence the time for the route must include the time for any inter-station transfers.

The example input below refers to the diagram given above.

Sample input

```
4
A:fgmpnxzabjd=Dbf
D:b=Adac=Ccf
B:acd=Azefg=Cbh
C:bac
AgAa
AbBh
BhDf
#
```

Sample output

```
  5: Agfdjba
  9: Abaz=Bdefgh
 10: Bhg=Cbac=Dcf
```