Maratona de Programação

Universidade de São Paulo – 2003

Caderno de Problemas

Departamento de Ciência da Computação, IME–USP

Problema A: Prague vikings?

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

Vestígios de uma antiga civilização viking foram descobertos nos arredores de Praga, e uma grande quantidade de material impresso foi achada junto ao sítio arqueológico. Como esperado, a leitura deste material mostrou-se uma tarefa árdua e desafiadora, já que essa civilização utilizava um esquema de codificação de texto para evitar que seu conhecimento fosse assimilado por seus rivais.

Recentemente, pesquisadores tchecos anunciaram com grande euforia à imprensa a compreensão do mecanismo de codificação utilizado por esses vikings. De acordo com os pesquisadores, o alfabeto viking era composto pelas letras de A até Z (incluindo as letras 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 era movida para o início da lista. Por exemplo, a codificação viking para a mensagem:

$$A\ B\ B\ B\ A\ A\ B\ B\ B\ B\ A\ C\ C\ A\ B\ B\ A\ A\ A\ B\ C$$

era dada pela seguinte seqüência de inteiros:

1 2 1 1 2 1 2 1 1 1 2 3 1 2 3 1 2 1 1 2 3

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

Entrada

Seu programa deve estar preparado para trabalhar com diversas instâncias. Cada instância tem a estrutura que segue. Na primeira linha é 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.

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, você deve imprimir o texto decodificado. Uma linha em branco deve separar a saída de cada instância.

Exemplo

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
```

Saída

```
Instancia 1
ABBBAABBBBACCABBAAABC

Instancia 2
VEGAN

Instancia 3
XXX

Instancia 4
VIDALONGAAOSSTRAIGHTEDGERS
```

Problema B: Anachronic?

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

Mr. Řež Slovankou é proprietário de uma empresa em Praga especializada no transporte de produtos perecíveis. Por ser uma das mais antigas e tradicionais empresas do ramo, ela é responsável por mais de 90% dos transportes deste tipo de mercadoria realizadas no país. Diariamente, vários caminhões carregados deixam postos da empresa (local onde os produtos ficam armazenados) com destino aos centros consumidores.

Visando um futuro ingresso no Espaço Econômico Europeu, o governo da República Tcheca, através do órgão de inspeção e vigilância sanitária, baixou uma série de medidas para assegurar ainda mais a qualidade dos alimentos consumidos pela população. Para não perder mercado, a empresa de Mr. Řež precisa adequar-se, o mais rápido possível, às novas regras.

Segundo o governo, a partir de agora, produtos destinados a diferentes centros consumidores não mais poderão compartilhar um mesmo caminhão. Além disso, todo transporte deverá ser concluído em um período abaixo de um tempo limite, especificado para cada mercadoria.

A adequação a essas novas regras trará consigo um aumento generoso nos custos operacionais da empresa. Preocupado com isso, Mr. Řež decidiu que seus caminhões deverão satisfazer às regras, mas que também deverão percorrer sempre o menor caminho possível até os centros consumidores (para economizar combustível, gastar menos os pneus, etc.).

Para resolver este problema logístico, Mr. Řež pediu auxílio a você, que disse ser capaz de construir um programa que recebe um mapa da malha viária (em que cada estrada tem associado um comprimento em kilômetros e um tempo em minutos que leva para ser percorrida), a origem e o destino do transporte, o tempo limite, e diz se é ou não possível realizar a entrega de acordo com as restrições impostas. Em caso afirmativo, você ainda disse que responderá o tempo gasto e a distância percorrida.

Mr. Řež ficou muito impressionado e satisfeito com sua prestatividade, e prometeu-lhe uma boa gratificação.

Entrada

Seu programa será capaz de resolver várias instâncias do problema logístico. Cada instância possui a estrutura descrita abaixo.

Na primeira linha são fornecidos o número de cidades ($0 \leq n \leq 100$) e o número de estradas ($0 \leq m \leq 10000$) da malha viária. Nas próximas m linhas são dados, em cada linha, quatro números inteiros separados por espaços em branco, x y c t, em que x e y são as cidades de onde sai e chega uma estrada, respectivamente, c é o comprimento desta estrada e t o tempo gasto para percorrê-la (obtido junto ao departamento nacional de estradas e rodagem, que afere o trânsito de tempos em tempos). Observe que $1 \leq x, y \leq n$ e que $c, t \geq 0$. Uma característica interessante na República Tcheca é que as estradas são de mão única para evitar as ultrapassagens indevidas, que causam acidentes.

Na linha seguinte é fornecido um inteiro $k \geq 0$ que representa o número de entregas que seu programa deve planejar nesta malha viária. As próximas k linhas possuem, em

ordem, a cidade de origem, a cidade de destino e o tempo limite $\hat{t}$ para o transporte da mercadoria, em minutos. De acordo com o órgão de inspeção e vigilância sanitária, nenhum transporte com duração superior a 100 horas será admitido. Logo, $0 \leq \hat{t} \leq 6000$. Um valor $n = 0$ indica o término das instâncias e não deve ser processado.

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. As próximas k linhas referem-se à possibilidades das entregas. Para $1 \leq j \leq k$, você deve imprimir na j-ésima linha a mensagem `Impossivel` se não for possível realizar a entrega dentro do tempo limite, ou `Possivel - <d> km, <t> min`, onde `<d>` é a distância que deve ser percorrida e `<t>` o tempo gasto para percorrer a distância. Se houver mais de um caminho com a menor distância possível, escolha o de menor tempo.

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

Exemplo

Entrada

```
5 8
1 2 1 3
1 4 1 1
2 3 1 8
2 4 3 1
3 5 1 3
4 5 4 2
5 1 8 8
5 3 1 1
5
1 5 7
5 1 5
1 3 4
1 3 12
1 3 3
0 0
```

Saída

```
Instancia 1
Possivel - 5 km, 3 min
Impossivel
Possivel - 6 km, 4 min
Possivel - 2 km, 11 min
Impossivel
```

Problema C: Tic-Tac-Toe?

`Arquivo: t3.[c|cpp|pas|java]`

Mickayil Romanoff ganhou de presente de aniversário um jogo muito interessante: um jogo da velha tridimensional. O jogo é feito de $n \times n$ pinos, dispostos em forma de uma matriz quadrada. Cada pino tem espaço para receber n bolinhas das cores branco e azul. Como no jogo da velha tradicional o objetivo é conseguir uma seqüência completa (em qualquer direção) de n bolinhas da mesma cor. Note que ao colocar uma bolinha num dos pinos ela necessariamente cai até chegar ao primeiro nível vazio por causa da gravidade.

Depois de vários jogos, Mickayil percebeu que não conseguia saber se alguém tinha ganho. Sua tarefa neste problema é ajudar ao Mickayil, escrevendo um programa que recebe uma partida e determina quem ganhou.

Entrada

São dadas várias instâncias. A primeira linha de cada instância contém a dimensão $0 \leq n \leq 30$ da matriz. A seguir, em cada uma das próximas n^3 linhas são dadas alternadamente as posições em que os jogadores estão jogando as bolinhas começando pelo jogador branco. Cada posição é dada pelo pino em que a bolinha da cor correspondente foi colocada, ou seja, um par (i, j) onde $i, j \in \{1, \ldots, n\}$. A entrada termina com um zero.

Saída

Você deverá imprimir um cabeçalho indicando o número da instância que está tratando (`Instancia h`) e na linha seguinte a mensagem de que jogador foi o vencedor da partida (`Branco ganhou` ou `Azul ganhou`), ou se o jogo empatou (`Empate`). Lembre-se de que vence a partida o jogador que primeiro conseguiu uma seqüência completa. As instâncias deverão estar separadas por uma linha em branco.

Exemplo

Entrada	Saída

`2`
`1 1`
`1 2`
`1 2`
`2 1`
`2 2`
`2 1`
`2 2`
`1 1`
`0`
	`Instancia 1`
	`Branco ganhou`

Problema D: A Cluster to avoid floods?

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

A Academia de Ciências da República Tcheca, preocupada com as inundações ocorridas durante os últimos verões em Praga, está fomentando o desenvolvimento de um novo cluster computacional para, entre outras tarefas, promover uma previsão do tempo mais acurada. Este novo cluster é composto por m máquinas iguais operando em paralelo. Por razões orçamentárias, cada máquina pode processar uma única tarefa por vez, e cada tarefa não pode ser processada em mais de uma máquina simultaneamente. O cluster permite, no entanto, preempção. Ou seja, é possível interromper a execução de uma tarefa e retorná-la posteriormente, em outra máquina inclusive.

Por estar em Praga para a realização de um evento relativo à Ciência da Computação, você foi convidado a desenvolver uma versão preliminar do escalonador de tarefas do cluster. Nesta versão, é fornecido um conjunto de tarefas T, em que cada tarefa $t \in T$ possui:

- Um requisito de processamento p_t que denota o número de unidades de tempo necessárias para realizar tal tarefa;
- Um instante de liberação r_t, que representa a unidade de tempo a partir da qual a tarefa está disponível para processamento (ela poderia estar aguardando dados, por exemplo);
- E um valor $d_t \geq p_t + r_t$ que indica o primeiro instante, em unidades de tempo, em que a tarefa deve, impreterivelmente, ter sido completada. Isto é, a tarefa t deve ser realizada no intervalo $[r_t, d_t)$.

Seu escalonador deve receber estes dados, de acordo com o formato descrito abaixo e dizer se existe ou não um escalonamento viável, isto é, um escalonamento que complete todas as tarefas nos intervalos de tempo permitidos.

Entrada

Seu escalonador deve estar preparado para trabalhar com diversas instâncias de entrada. Cada instância segue o formato que segue. Na primeira linha, são fornecidos os números de máquinas, $0 \leq m \leq 100$, e de tarefas, $0 \leq n \leq 1000$, respectivamente. Nas próximas n linhas são fornecidos os valores $p_t \geq 0, r_t \geq 0$ e $d_t \geq 0$ (uma tripla por linha) para as tarefas $t \in T$. Os instantes r_t e d_t são inteiros, e p_t é decimal. Valores $m = 0$ e $n = 0$ indicam o término do processamento das instâncias e não devem ser processados. Todos os valores da entrada que constem em uma mesma linha são separados por um número qualquer de espaços em branco.

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 deve

ser impresso `Viavel` ou `Nao viavel`, dependendo do escalonamento para a instância ser ou não viável, respectivamente.

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

Exemplo

Entrada

```
3 4
1.5  3 5
1.25 1 3
2.1  3 7
3.6  5 9
3 1
3 1 2
0 0
```

Saída

```
Instancia 1
Viavel

Instancia 2
Nao viavel
```

Problema E: Magic numbers?

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

"Os números sempre desempenharam um papel de acentuado relevo não só nos altos campos da Fé e da Verdade, como no humílimos terreiros da Superstição e do Erro." (Prof. Marão)

Malba Tahan, em seu clássico "O Homem Que Calculava", conta uma fábula de superstição envolvendo os números quadripartidos. Mal sabia ele que séculos antes, na antiga civilização Tcheca, a superstição envolvendo os números quadripartidos já se fazia presente. Na antiguidade, uma importante comunidade que vivia nos arredores de Neratovice, utilizava as propriedades dos Números quadripartidos para prever o futuro, batizar as crianças e até mesmo para escolher os seus líderes.

Um número inteiro n é quadripartido se existe alguma divisão desse número em quatro parcelas inteiras ($p_1+p_2+p_3+p_4 = n$) e um operador mágico (m) de modo que a primeira parcela somada ao operador mágico, a segunda diminuída dele, a terceira multiplicada por ele e a quarta dividida por ele dêem o mesmo resultado ($p_1+m = p_2-m = p_3*m = p_4/m$).

Assim, 128 é quadripartido, porque podemos dividir 128 em 4 parcelas ($31, 33, 32$ e 32) de modo que existe um operador mágico (no caso, 1) que faz com que p_1+m, p_2-m, p_3*m e p_4/m sejam iguais. De fato: $31 + 1 = 33 - 1 = 32 * 1 = 32/1 = 32$.

Um grupo de pesquisadores de Praga está reconstruindo o passado de Neratovice, e pediu a sua ajuda. Eles querem que você faça um programa que identifique quando um número é ou não quadripartido e qual é o seu operador mágico associado.

Entrada

Cada linha da entrada contém um inteiro n ($0 \leq n \leq 500000$) que seu programa deverá analisar e classificar em quadripartido ou não. O valor $n = 0$ corresponde ao final do arquivo de entrada e não deve ser processado.

Saída

Para cada valor da entrada, seu programa deve imprimir um identificador `Instancia h`, em que `h` é um número inteiro, seqüêncial e crescente a partir de 1. Na linha seguinte, separados por um espaço em branco, os cinco números que comprovam a condição de quadripartido, quando n for quadripartido. Siga a ordem: m p_1 p_2 p_3 p_4.

Se n não for quadripartido, seu programa deve imprimir a mensagem n `nao e quadripartido`. No primeiro caso, é possível que exista mais de uma seqüência que atenda às condições estabelecidas. Se isto ocorrer, seu programa deverá escolher a que apresentar o maior valor possível para m.

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

Exemplo

Entrada

```
128
1
8
0
```

Saída

```
Instancia 1
7 7 21 2 98

Instancia 2
1 nao e quadripartido

Instancia 3
1 1 3 2 2
```

Problema F: Drunk people?

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

Nos invernos de Praga o frio é insuportável. Para manter o corpo aquecido, diversos moradores utilizam-se de bebidas quentes (como café ou chá) ou alcoólicas. De tempos em tempos, Mr. Třeboň freqüenta um pequeno bar de seu bairro. Porém, depois de muitas horas, alguns acabam exagerando nas doses. Nesse momento, Mr. Třeboň precisa retornar a sua casa. O problema é que ele não está se sentindo bem (ele está bêbado mesmo :-) e não lembra onde mora. Então ele começa a tocar as campainhas das casas e a perguntar se mora ali. Assim faz até encontrar sua casa. Devido ao seu estado, o Mr. Třeboň não segue nenhuma ordem lógica para tocar as campainhas das casas. Após tocar uma campainha e verificar que não é a sua casa, ele irá continuar procurando. Além disso, ele não consegue memorizar quais campainhas já tocou. A forma como ele escolhe as casas para tocar a campainha segue uma distribuição de probabilidade condicionada apenas à última casa tocada. Considere que alguém sempre atende à porta e responde ao Mr. Třeboň se ele mora ali ou não. Queremos saber qual a chance dele não conseguir chegar em casa para dormir, sabendo que após tocar um certo número de campainhas ele não aguentará mais e ficará por ali mesmo.

Entrada

O arquivo de entrada tem a seguinte composição para cada instância: a primeira linha contém os inteiros $0 \leq n \leq 100$, $0 < t \leq n$, $0 < k \leq n$, $0 < m \leq 100$, representando o número de casas, a casa inicial, a casa do Mr. Třeboň e a quantidade de casas que ele poderá tocar a campainha para tentar chegar em casa, respectivamente. São dadas então n linhas. A i-ésima linha representa a casa i e contém os números $a_{i1}, \ldots, a_{ij}, \ldots, a_{in}$, separados por brancos, representando a probabilidade do Mr. Třeboň ir da casa i para a casa j. A entrada termina com $n = 0$.

Saída

O arquivo de saída deve conter, para cada instância de entrada, um identificador `Instancia h`, em que `h` é um número inteiro, seqüencial e crescente a partir de 1. Na linha seguinte um número (arredondado em 6 casas decimais) indicando a probabilidade do Mr. Třeboň não ter encontrado a sua casa após m campainhas tocadas.

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

Exemplo

Entrada

```
2 1 2 1
0.5 0.5
0.5 0.5
3 1 2 2
0.25 0.25 0.5
0.25 0.5 0.25
0.5 0.25 0.25
0
```

Saída

```
Instancia 1
0.500000

Instancia 2
0.562500
```

Problema G: Venn diagram?

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

Mickayil estava na escola estudando teoria de conjuntos. As escolas em Praga são bastante rigorosas. A professora ensinou para os alunos como desenhar diagramas de Venn para três conjuntos, e pediu aos alunos que trabalhassem com os diagramas. Para impressionar a professora com diagramas bem desenhados, Mickayil decidiu que iria fazer diagramas coloridos. Para atingir esse objetivo, ele recortou três figuras convexas usando finos papéis coloridos. Mickayil percebeu que, sobrepondo as figuras, as intersecções podiam gerar novas cores.

Assim, temos três figuras convexas sobre uma superfície plana, cada figura com uma cor. Quando figuras estão sobrepostas, a cor que vemos é uma mistura das cores das figuras. Podemos representar as cores por inteiros entre 0 e 15. A mistura de cores é feita através da soma dos valores das correspondentes cores módulo 16. Queremos saber o que enxergamos ao olhar para o diagrama de Venn do Mickayil, isto é, quais cores e em que quantidade elas aparecem. Vamos medir isso através da área visível de cada cor.

Entrada

São dadas várias instâncias. Cada instância é composta por três figuras. Essas figuras são dadas uma por linha, com o formato

$$n_i \; c_i \; x_{i1} \; y_{i1} \; x_{i2} \; y_{i2} \; \ldots \; x_{ij} \; x_{ij} \; \ldots \; x_{in} \; y_{in}$$

para $i = 1, 2, 3$, onde $0 \leq n_i \leq 200$ é o número de pontos na borda da figura i, c_i é a cor da figura i e os pares (x_{ij}, y_{ij}) representam os pontos da borda de i no sentido anti–horário, dados por inteiros. A entrada termina quando $n_1 = 0$ (note que essa instância não deve ser processada).

Saída

Você deverá imprimir um cabeçalho indicando o número da instância que está tratando (`Instancia h`) e nas linhas seguintes o par cor e área total visível desta cor. Estas linhas devem ser ordenadas decrescentemente pelas áreas. Resolva os empates de forma crescente na cor. Apenas para simplificar a apresentação, mostre os valores das áreas arredondados em duas casas decimais. As instâncias deverão ser separadas por uma linha em branco.

Exemplo

Entrada

```
4 5 -1 -1 1 -1 1 1 -1 1
4 13 0 0 2 0 2 2 0 2
3 7 3 0 4 0 4 1
4 1 0 1 3 1 3 4 0 4
4 2 2 2 5 2 5 5 2 5
4 8 1 0 4 0 4 3 1 3
0
```

Saída

```
Instancia 1
5 3.00
13 3.00
2 1.00
7 0.50

Instancia 2
2 6.00
1 4.00
8 4.00
9 3.00
3 1.00
10 1.00
11 1.00
```

Problema H: DNA storage?

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

A Universidade Charles, situada em Praga, a exemplo de diversas outras universidades de renome ao redor do mundo, instituiu recentemente um programa interdepartamental de pós-graduação na área de biologia computacional. Integrante do corpo docente, Ms. Dolejškova está atualmente interessada no problema das árvores filogenéticas, e trabalhando, portanto, com n cadeias de DNA. Para simplificar o trabalho, Ms. Dolejškova resolveu trabalhar apenas com cadeias gênicas de comprimento m (isto é, todas as cadeias possuem exatamente m bases nitrogenadas).

Um subproblema interessante envolve o armazenamento das n cadeias em disco. Até o momento, Ms. Dolejškova está utilizando um esquema ingênuo que requer $n \times m$ caracteres, além dos delimitadores. Isto é, todas as seqüências são gravadas dentro de um arquivo texto, seqüencialmente. Mr. Chuchle, um colega de departamento e especialista em técnicas de armazenamento, sugeriu uma alternativa que pode ser mais econômica.

Segundo Mr. Chuchle, é possível armazenar uma cadeia juntamente com informações que permitam transformá-la em outras. Mais especificamente, considere duas cadeias de DNA $D_1 = ACTA$ e $D_2 = AGTC$, onde $A,\ C,\ G,\ T$ representam as bases nitrogenadas adenina, citosina, guanina e timina, nesta ordem. Observe que é possível transformar D_1 em D_2 trocando-se as bases nitrogenadas C e A das posições 2 e 4 de D_1 para G e C, respectivamente. Considere agora uma terceira cadeia $D_3 = CGTC$. É necessária apenas uma modificação para transformar D_2 em D_3 e são necessárias três modificações para transformar D_1 em D_3. Logo, é vantajoso permitir a transitividade das modificações entre as cadeias.

Ms. Dolejškova observou rapidamente que, se as cadeias envolvidas forem muito diferentes entre si, este esquema de armazenamento alternativo não oferece ganhos. Assim, em vez de adotá-lo prontamente, ela solicitou a você que construa um programa que recebe as n cadeias, e determina o número mínimo de transformações que devem ser gravadas (além de uma cadeia) para que seja possível, no futuro, obter-se novamente as n cadeias originais. Baseado no resultado fornecido por seu programa, Ms. Dolejškova vai decidir qual dos esquemas deve utilizar em cada instância de dados que tiver.

Entrada

Seu programa deve estar preparado para trabalhar com diversas instâncias. Cada instância tem a estrutura que segue. Na primeira linha são fornecidos dois inteiros n e m ($0 \leq n \leq 100$ e $1 \leq m \leq 1000$) que representam, nesta ordem, o número de cadeias de DNA e o comprimento delas. Nas próximas n linhas são fornecidas as n cadeias, uma por linha, sem espaços adicionais. Cada cadeia é uma seqüência de caracteres tomada sobre o alfabeto $\Sigma = \{A,\ C,\ G,\ T\}$. Um valor $n = 0$ indica o final das instâncias e não deve ser processado.

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, você deve imprimir o número mínimo de transformações que devem ser gravadas para esta instância. Uma linha em branco deve separar a saída de cada instância.

Exemplo

Entrada

```
3 4
ACTA
AGTC
CGTC
4 5
AAAAA
ATATA
ATCTA
AACAA
0 0
```

Saída

```
Instancia 1
3

Instancia 2
4
```

Problema I: Money for something?

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

Após ter ouvido a notícia de que a vigésima oitava edição do ACM International Collegiate Programming Contest será realizada em Praga, em março de 2004, Mr. Henry ficou com vontade de conhecer a cidade. Durante seus planejamentos para a viagem, ele percebeu que seria muito interessante ter recursos financeiros suficientes. Em conversa com o gerente de sua conta bancária, Mr. Henry decidiu aplicar mais agressivamente seus ganhos. Tentando minimizar os riscos, ele selecionou aplicações com ganhos pré-fixados ao longo de um período fixado. Todas essas aplicações apresentam um retorno de crescimento monotônico, ou seja, quanto maior a quantia investida, maior o retorno. A grande questão tornou-se escolher o quanto deve ser aplicado, de um montante disponível, em cada aplicação para maximizar o ganho. O banco de Mr. Henry só permite aplicações de valores inteiros de dinheiro (não utiliza centavos).

Para esta nobre tarefa, Mr. Henry solicitou o seu auxílio, e você, prontamente, disse que escreveria um programa para ajudá-lo. :-)

Entrada

Seu programa deve estar preparado para solucionar diversas instâncias fornecidas em seqüência. Cada uma delas possui o formato que segue. Na primeira linha é dada a quantidade de dinheiro, denotada por m, que Mr. Henry deseja aplicar. Tem-se que m é inteiro e $0 \leq m \leq 10000$. Na segunda linha é dado o número n de aplicações em que o montante m pode ser investido. Tem-se que n é um valor inteiro e que $0 < n \leq 50$.

As próximas n linhas possuem o mesmo formato. Na i-ésima linha ($1 \leq i \leq n$) são fornecidos m valores, em que o j-ésimo valor ($1 \leq j \leq m$) é o montante que a i-ésima aplicação devolve quando investido uma quantidade j de dinheiro.

Todos os valores da entrada que constem em uma mesma linha são separados por um número qualquer de espaços em branco.

Um valor $m = 0$ indica que não há mais instâncias a serem solucionadas.

Saída

Para cada instância da entrada, você deverá imprimir um identificador `Instancia h`, em que `h` é um número inteiro, seqüencial e crescente a partir de 1. Na linha seguinte, imprima o lucro obtido com as aplicações. Nas próximas n linhas, você deve imprimir o número da aplicação (em ordem crescente), dois pontos, espaço em branco e quanto de dinheiro deve ser nela investido. Se existir mais de uma opção de investimento que forneça o maior lucro, escolha uma delas.

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

Exemplo

Entrada

```
5
4
11 12 13 14 15
 0  5 10 15 20
 2 10 30 32 33
20 21 22 23 24
0
```

Saída

```
Instancia 1
56
1: 1
2: 0
3: 3
4: 1
```