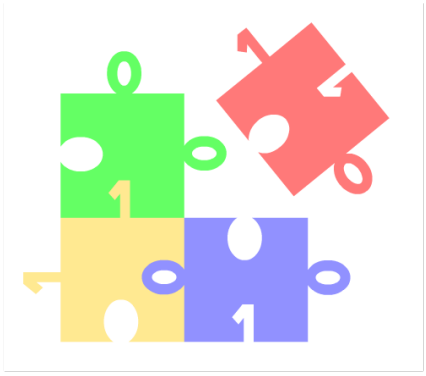


4ª Minimaratona de Programação UFSCar Sorocaba 2014

16 de maio de 2014

Sessão Principal

Este caderno de problemas contém 6 problemas em páginas numeradas de 1 a 13.

Apoio:

PROEX
Pró-Reitoria de Extensão

fit
Instituto de Tecnologia

Informações Gerais

Exceto quando explicitamente indicado, as seguintes condições valem para todos os problemas:

Entrada

- A entrada deve ser lida da entrada padrão.
- A entrada é composta de vários casos de teste. Cada caso de teste é descrito usando um número de linhas que depende do problema.
- Quando uma linha de dados contém vários valores, estes são separados por um único espaço. Não aparecem outros espaços na entrada. Não existem linhas vazias.
- Cada linha, incluindo a última, possui um indicador de fim de linha.
- O fim da entrada é indicado por uma linha contendo valores específicos que dependem do problema. Esta linha não deve ser processada como um caso de teste.

Saída

- A saída deve ser escrita na saída padrão.
- O resultado de cada teste deve aparecer na saída usando um número de linhas que depende do problema.
- Quando uma linha de saída contém vários valores, estes devem ser separados por um único espaço. Não podem aparecer outros espaços na saída. Não devem existir linhas vazias.
- Cada linha, incluindo a última, deve possuir um indicador de fim de linha.
- Nenhum indicador especial deve ser escrito para indicar o fim da saída.
- O formato da saída deve seguir rigorosamente o padrão descrito no problema.

Problema A

Arbitragem

Nome do arquivo: arbitragem.c, arbitragem.cpp, arbitragem.java

Tipicamente, sempre perde-se dinheiro na conversão de moedas. Todos sabem disso. No entanto, como a taxa de conversão de moedas é muito dinâmica, e as taxas de conversão são determinadas pelas imprevisíveis relações comerciais internacionais, em algumas raras situações, há a possibilidade de que uma sequência cíclica de conversões de moedas permita que se ganhe dinheiro. Por exemplo, considerando as seguintes taxas de conversão entre as moedas real, dólar e euro:

Moeda	**Real**	**Dólar**	**Euro**
Real	1	0,45	0,33
Dólar	2,22	1	0,73
Euro	3,03	1,37	1

se convertermos 1 real em euro, depois em dólares e depois em reais novamente teremos ao final 1 * 0,33 * 1,37 * 2,22 = 1,003662 reais. Evidentemente, se isso acontece, eis uma “máquina de fazer dinheiro”.

O controle das taxas de conversão de forma que isso seja evitado é conhecido no mercado financeiro como o Problema da Arbitragem. A bolsa de valores precisa de um programador que verifique constantemente se há uma oportunidade de arbitragem dado um conjunto de taxas de conversão. Você aceita essa missão?

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N ($0 \leqslant N \leqslant 1000$) que representa quantas são as moedas envolvidas. As N linhas subsequentes de cada caso de teste contém N números fracionários t_{ij} ($0 \leqslant t_{ij} \leqslant 1000$), com no máximo duas casas decimais, que representam as taxas de conversão da moeda i para a moeda j.

O final da entrada é indicado por N = 0.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha contendo “S” caso haja oportunidade de arbitragem ou “N” caso contrário.

Exemplo de entrada	**Saída para exemplo de entrada**
3 1 0.45 0.33 2.22 1 0.73 3.03 1.37 1 3 1 0.40 0.30 2.00 1 0.70 3.00 1.30 1 4 1 0.25 5 2 4 1 10 100 0.20 0.10 1 50 0.5 0.01 0.02 1 3 1 0.30 5 2.5 1 9 0.1 0.08 1 0	S N S N

Problema B

Jogo do Bicho

Nome do arquivo: bicho.c, bicho.cpp, bicho.java

Em um país muito distante, as pessoas são viciadas em um jogo de apostas bastante simples. O jogo é baseado em números e é chamado jogo do bicho. O nome do jogo deriva do fato que os números são divididos em 25 grupos, dependendo do valor dos dois últimos dígitos (dezenas e unidades), e cada grupo recebe o nome de um animal. Cada grupo é associado a um animal da seguinte forma: o primeiro grupo (burro) consiste nos números 01, 02, 03 e 04; o segundo grupo (águia) á composto dos números 05, 06, 07 e 08; e assim em diante, até o último grupo contendo os números 97, 98, 99 e 00.

As regras do jogo são simples. No momento da aposta, o jogador decide o valor da aposta V e um número N ($0 \leqslant N \leqslant 1000000$). Todos os dias, na praça principal da cidade, um número M é sorteado ($0 \leqslant M \leqslant 1000000$). O prêmio de cada apostador é calculado da seguinte forma:

• se M e N tem os mesmos quatro últimos dígitos (milhar, centena, dezena e unidade), o apostador recebe V × 3000 (por exemplo, N = 99301 e M = 19301);

• se M e N têm os mesmos três últimos dígitos (centena, dezena e unidade), o apostador recebe V × 500 (por exemplo, N = 38944 e M = 83944);

• se M e N têm os mesmos dois últimos dígitos (dezena e unidades), o apostador recebe V × 50 (por exemplo, N = 111 e M = 552211);

• se M e N têm os dois últimos dígitos no mesmo grupo, correspondendo ao mesmo animal, o apostador recebe V × 16 (por exemplo, N = 82197 e M = 337600);

• se nenhum dos casos acima ocorrer, o apostador não recebe nada.

Obviamente, o prêmio dado a cada apostador é o máximo possível de acordo com as regras acima. No entanto, não é possível acumular prêmios, de forma que apenas um dos critérios acima deve ser aplicado no cálculo do prêmio. Se um número N ou M com menos de quatro dígitos for apostado ou sorteado, assuma que dígitos 0 devem ser adicionados na frente do número para que se torne de quatro dígitos; por exemplo, 17 corresponde a 0017.

Dado o valor apostado, o número escolhido pelo apostador, e o número sorteado, seu programa deve calcular qual o prêmio que o apostador deve receber.

Entrada

A entrada contém vários casos de teste. Cada caso consiste em apenas uma linha, contendo um número real V e dois inteiros N e M, representando respectivamente o valor da aposta com duas casas decimais ($0.01 \leqslant V \leqslant 1000.00$), o número escolhido para a aposta ($0 \leqslant N \leqslant 1000000$) e o número sorteado ($0 \leqslant M \leqslant 1000000$). O final da entrada é indicado por uma linha contendo V = M = N = 0.

Saída

Para cada um dos casos de teste seu programa deve imprimir uma linha contendo um número real, com duas casas decimais, representando o valor do prêmio correspondente a aposta dada.

Exemplo de entrada	**Saída para exemplo de entrada**
32.20 32 213929 10.50 32 213032 2000.00 340000 0 520.00 874675 928567 10.00 1111 578311 0 0 0	515.20 5250.00 6000000.00 0.00 500.00

Problema C

Caminho de Soma Máxima

Nome do arquivo: caminho.c, caminho.cpp ou caminho.java

Começando pelo topo do triângulo abaixo e movendo-se por valores adjacentes da linha seguinte até chegar a base, o valor da soma máxima é 23. Considerando como adjacente o valor imediatamente abaixo ou uma posição à direita do último valor somado.

3

7 *4*

2 **4** *6*

8 5 **9** *3*

Neste triângulo a soma é obtida pelo caminho 3 + 7 + 4 + 9 = 23, que é um caminho de soma máxima. Encontre o valor da soma máxima do topo à base dos triângulos de entrada.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N, a quantidade de linhas do triângulo ($1 \leqslant N \leqslant 100$). Cada uma das N linhas subsequentes contém cada uma das linhas do triângulo. Cada linha possui valores inteiros separados por um espaço em branco, de um inteiro na primeira linha a N inteiros na última linha.

O final da entrada é indicado por N = 0.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha contendo um único inteiro que é a soma máxima do topo à base do triângulo da entrada.

Exemplo de entrada	**Saída para exemplo de entrada**
4 3 7 4 2 4 6 8 5 9 3 3 1 1 7 9 1 1 0	23 11

Problema D

IsaSeq

Nome do arquivo: isaseq.c, isaseq.cpp ou isaseq.java

Isabela tem 7 anos e gosta muito de brincar com um joguinho que tem muitas letras e números. Ela é muito inteligente e gosta de pedir aos adultos para que criem desafios para ela resolver. Certo dia, ela estava organizando seu brinquedo e começou a construir várias sequências intercalando letras e números e sempre iniciando e terminando com letra. Essas sequências foram chamadas de SeqInicial e podemos considerar exemplos válidos as sequências: A2S5E2B5S9H4K5W, A2B, U5E2J5D6G5K4C, X1X1X. Isabela então criou uma regra para criar sua sequência IsaSeq, de maneira que é possível transformar qualquer SeqInicial em IsaSeq e vice-versa. A IsaSeq sempre começa com uma letra e termina com um número.

Isabela usou o seguinte raciocínio para a montagem da sequência IsaSeq. Procure a primeira subsequência [letra1 número letra2] cujo número é a primeira ocorrência do maior dígito presente na sequência (da esquerda para a direita). Em seguida, transforme-a em IsaSeq, isto é [letra1 letra2 número] e substitua essa sequência por uma letra especial. Repita esse processo até que sobre apenas uma letra especial. Então substitua todas as letras especiais pelas IsaSeq. Veja o exemplo a seguir:

SeqInicial: A1B2C2D3E

A1B2C2D3E
  D3E => DE3 => X
A1B2C2X
  B2C => BC2 => Y
A1Y2X
  Y2X => YX2 => Z
A1Z
  A1Z => AZ1 => W
W

Para construir a IsaSeq:

W = AZ1
 => AZ1
Z = YX2
 => AYX21
Y = BC2
X = DE3
 => ABC2DE321

Ela quer mostrar para você que ela consegue transformar qualquer IsaSeq em SeqInicial. Para isso, você deve conseguir transformar uma SeqInicial em IsaSeq.

Entrada

A entrada contém vários casos de teste. Cada um deles é dado em exatamente uma linha contendo um conjunto não vazio da sequência SeqInicial (letra número letra número ... letra). O tamanho máximo de SeqInicial é 10^4 caracteres. Após o último caso de teste segue uma linha contendo um zero.

Saída

Para cada caso de teste, a saída será uma linha que representa a IsaSeq da SeqInicial de entrada.

Exemplo de entrada	**Saída para exemplo de entrada**
A1B2C2D3E A1B3C3D4E3F Y0V1I 0	ABC2DE321 ABC3DE43F31 YVI10

Problema E

Eleição em Ecaterimburgo

Nome do arquivo: eleicao.c, eleicao.cpp ou eleicao.java

Ecaterimburgo, Rússia, é uma cidade com um curioso sistema de votação. Em uma eleição em que haja V vagas para um cargo, cada eleitor tem direito a fazer V votos, ordenados em sua ordem de preferência. Assim, se, por exemplo, há 3 vagas de senador, cada eleitor vota em até 3 nomes. Serão eleitos os candidatos que tiverem o maior número de votos, sem importar em que posição da preferência do eleitor está o candidato. Apenas quando há empate no número de votos se torna relevante a ordem dada pelos eleitores. Ganha aquele candidato que tiver mais indicações em primeiro lugar. Se persistir o empate, em segundo lugar, e assim por diante. Caso dois ou mais candidatos que estejam em posição de serem eleitos tenham exatamente o mesmo número de indicações em todas as posições, todos são eleitos (podendo inclusive exceder o número de vagas). Candidatos com zero votos podem ser eleitos se ainda existir vagas disponíveis.

Entrada

A entrada é composta por diversas instâncias. Cada instância começa com o número N de eleitores ($1 \leqslant N \leqslant 10^5$), o número K de candidatos e V de vagas ($1 \leqslant V \leqslant K \leqslant 100$). A seguir vêm N linhas com os votos de cada um dos eleitores. Em seu voto, o eleitor i indicará o número Li ($1 \leqslant Li \leqslant 100$) de candidatos em quem votará, e os índices destes candidatos na sua ordem de preferência.

Índices de candidatos fora do intervalo [1, K] significam votos em branco apenas para a opção de preferência correspondente. Se indicar mais que V votos, os últimos serão desconsiderados. Um eleitor nunca indica o mesmo candidato mais de uma vez.

O final da entrada é indicado por uma linha contendo N = K = V = 0.

Saída

Para cada instância da entrada seu programa deverá imprimir, em uma única linha, a lista de candidatos eleitos ordenada pela classificação dos candidatos na eleição. No caso de dois candidatos possuírem a mesma classificação, o de menor índice vem antes.

Exemplo de entrada	**Saída para exemplo de entrada**
5 3 2 2 1 3 2 2 3 2 1 3 2 2 3 1 1 3 6 3 3 1 5 3 3 1 0 3 3 1 4 5 0 0 0	3 1 1 5 3

Problema F

Criptografia Fuleco

Nome do arquivo: fuleco.c, fuleco.cpp ou fuleco.java

Um dos organizadores da Copa do Mundo 2014 decidiu usar um sistema criptográfico baseado em ECC- *Elliptic Curve Cryptography* nos cartões inteligentes de autenticação das equipes nos eventos. Como estes cartões são muitos simples, ele precisa de uma implementação que utilize um tamanho de chave menor que os suportados pelas bibliotecas criptográficas disponíveis. Sua função como membro da equipe de TI da Copa Brasil 2014 é implementar parte deste algoritmo, que receberá o nome de CRIFU (Criptografia Fuleco).

As operações aritméticas realizadas pelo algoritmo ocorrem sobre pontos da curva elíptica definida. O conjunto de pontos válidos para o algoritmo é dado por um conjunto finito de pontos sobre a curva mais um ponto no infinito, neste caso definido por O=(0,0). Para este conjunto de pontos é definida a operação de soma tal que:

Seja P, Q e G pontos quaisquer da curva e O (ponto no infinito), valem as propriedades:

- P = Q + G é um ponto da curva.
- Q + G = G + Q
- P + Q + G = (P + Q) + G = P + (Q + G)
- O é o elemento neutro da adição: Q + O = O + Q = Q
- Existe o inverso aditivo G = -Q tal que Q + G = G + Q = O

Sua equipe deve implementar a operação de multiplicação de um ponto (G = (Xg,Yg)) da curva por um escalar (n), obtendo um outro ponto da curva (R = (Xr,Yr)). Esta operação é definida como a soma sucessiva do ponto G (gerador).

Exemplo: Se n = 3, então R = 3G = G + G + G

A soma de dois pontos quaisquer (Q e G) para a curva elíptica utilizada por CRIFO, resulta em outro ponto da curva (R = Q + G), sendo definida abaixo.

Considere,

- R= (Xr,Yr) , Q = (Xq,Yq), G =(Xg,Yg) e o ponto no infinito O = (0,0).
- que as coordenadas do ponto na curva (X,Y), o escalar n e todas as operações realizadas sobre estes operandos são operações sobre um corpo finito primo $GF(Z_p)$. Ou seja, Xi,Yi e n são inteiros com $0 \leqslant Xi,Yi,n \leqslant p-1$, para qualquer coordenada ou escalar multiplicativo, e as operações são realizadas modulo um inteiro primo p, com $p < 10^7$.
- que se Q = (Xq,Yq), então –Q = (Xq,-Yq) mod p, ou seja, -Q = (Xq , (p-Yq) mod p) para $0 \leqslant Yq \leqslant p-1$.
- que nG = O para n = 0.

A operação de soma de pontos é definida por R = Q + G:

$$R = \begin{cases} (Xq,Yq) & \text{se } G = O \\ (Xg,Yg) & \text{se } Q = O \\ O & \text{se } Q = -G \text{ (ou } G = -Q) \\ (Xr,Yr) \text{ com } \lambda = (Yg - Yq) / (Xg-Xq) \bmod p & \text{se } Q \neq \pm G \text{ e } Q,G \neq O \\ (Xr,Yr) \text{ com } \lambda = (3Xq^2 + \mathbf{a}) / (2Yq) \bmod p & \text{se } Q = G \text{ e } Q,G \neq O \text{ e } Yq \neq 0 \end{cases}$$

$$Xr = (\lambda^2 - Xq - Xg) \bmod p$$

$$Yr = (\lambda (Xq - Xr) - Yq) \bmod p$$

Observação:

- **a** é um coeficiente da curva e será fornecido,
- o cálculo de λ é uma operação modular. Observe a divisão modular: $X / Y = X*(Y)^{-1}$. $(Y)^{-1}$ é o inteiro menor que p que multiplicado por Y resulta em (1 mod p), ou seja, $Y * (Y)^{-1} \bmod p = 1 \bmod p$. Ex.: $7 * 2 \bmod 13 = 1 \bmod p$, então 2 é o inverso multiplicativo de 7 e vice-versa.

Exemplo:

Dados: n = 3; a = 3; p = 13; G=(2,10)

R = 3 G = G + G + G = (G + G) + G

$\lambda = (3Xg^2 + a)/(2Yg) \bmod p = (3*4+3)/(2*10) \bmod 13 = 2/7 \bmod 13 = 2*2 \bmod 13 = 4$

$Xq = (\lambda^2 - Xg - Xg) \bmod p = (16-2-2) \bmod 13 = 12$

$Yq = (\lambda (Xg-Xq) - Yg) \bmod p = (4*(2-12)-10) \bmod 13 = (-50) \bmod 13 = 2$

Segue-se o mesmo raciocínio para obter R = (12,2) + G

$\lambda = (Yg - Yq)/(Xg-Xq) \bmod p = (10-2)/(2-12) \bmod 13 = 8/3 \bmod 13 = 8*9 \bmod 13 = 7$

$Xr = (\lambda^2 - Xq - Xg) \bmod p = (49-12-2) \bmod 13 = 9$

$Yr = (\lambda (Xq-Xr) - Yq) \bmod p = (7*(12-9)-2) \bmod 13 = 6$

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha contém o escalar multiplicativo ***n***. A segunda linha contém quatro inteiros, separados por espaço, sendo na ordem o parâmetro da curva ***a***, o inteiro primo ***p*** e as coordenadas do ponto G, ***X*** e ***Y***. Os casos de teste terminam quando o valor **n** for igual a zero.

Saída

A saída deve fornecer em uma única linha as coordenadas ***X*** e ***Y*** do ponto R = nG para cada caso de teste.

Exemplo de entrada	**Saída para exemplo de entrada**
2 3 13 2 10 3 3 13 2 10 9 3 13 2 10 5 10 13 3 6 6 10 13 3 6 10 10 13 3 6 0	12 2 9 6 0 0 10 0 1 9 0 0

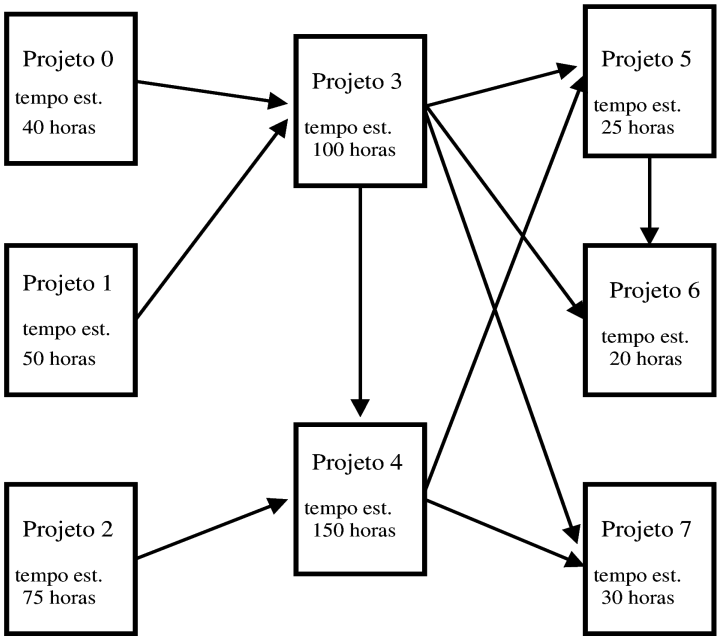
Problema G

Gerente de Projetos

Nome do arquivo: gerente.c, gerente.cpp ou gerente.java

Um gerente de projetos acabou de receber uma excelente notícia de sua chefe: assim que entregar todos os projetos sob sua responsabilidade no momento será promovido e seu salário dobrará! Imediatamente o gerente de projetos ficou ansioso para saber quando será a promoção e pediu (para ontem) uma estimativa, de cada equipe de cada projeto, do número de horas necessárias para concluir seu projeto. Suas equipes responderam rapidamente com suas estimativas, no entanto, lembraram que existem algumas relações de dependência entre os projetos gerenciados por ele, de forma que cada equipe só consegue começar quando todos os projetos de que dependem estiverem concluídos.

A figura abaixo mostra um exemplo de um conjunto de projetos com os respectivos tempos estimados para conclusão e os projetos dos quais dependem. No exemplo da figura o tempo mínimo para a conclusão completa dos projetos (e para o gerente conseguir a promoção) é de 345 horas, dado pela soma dos tempos de conclusão dos projetos 1, 3, 4, 5 e 6 que dependem sequencialmente um do outro, nessa ordem. Qualquer outra sequência de dependências leva a um tempo total de conclusão menor.



Ao perceber que determinar o tempo mínimo até a promoção é mais complicado do que imaginou inicialmente, o gerente de projetos foi lamentar com seus outros colegas gerentes sua ansiedade, e descobriu que todos eles estavam na mesma situação. Eles resolveram então se juntar e contratar você para determinar esse tempo mínimo para cada um deles.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois inteiros N e M, separados por um espaço em branco, que representam,

respectivamente, quantos são os projetos ($1 \leqslant N \leqslant 10.000$) e quantas são as relações são as relações de dependência ($1 \leqslant M \leqslant 100.000$). Cada projeto é representado por um inteiro entre 0 e N-1. A segunda linha de cada caso de teste contém N inteiros t_i ($1 \leqslant t_i \leqslant 10.000$) que representam os tempos estimados de conclusão de cada projeto. Cada uma das M linhas subsequentes de cada caso de teste contém dois inteiros u e v ($0 \leqslant u, v \leqslant N-1$), separados por um espaço em branco, indicando que o projeto v depende do projeto u para poder ser concluído.

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

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha contendo um único inteiro que é o tempo mínimo para o gerente concluir seus projetos atuais e obter a sua promoção.

Exemplo de entrada	**Saída para exemplo de entrada**
8 10 40 50 75 100 150 25 20 30 0 3 1 3 2 4 3 4 3 5 3 6 3 7 4 5 4 7 5 6 5 4 10 20 30 40 50 0 1 1 2 3 1 1 4 6 8 20 70 60 30 100 10 0 1 0 3 1 2 1 4 2 5 3 4 4 2 4 5 1 0 9384 0 0	345 110 260 9384