acm International Collegiate Programming Contest 2012 IBM event sponsor

Maratona de Programação da SBC 2012

Sub-Regional Brasil do ACM ICPC

15 de Setembro de 2012

Caderno de Problemas e Soluções

Promoção:

Sociedade Brasileira de Computação

Patrocínio:

Fundação Carlos Chagas

v1.0

Problema A

Concurso de Contos

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

Machado gosta muito de escrever. Já escreveu muitos contos, resenhas, relatos de viagens que fez, além de um pequeno romance. Agora Machado quer participar de um concurso de contos, que tem regras muito rígidas sobre o formato de submissão do conto.

As regras do concurso especificam o número máximo de caracteres por linha, o número máximo de linhas por página, além de limitar o número total de páginas. Adicionalmente, cada palavra deve ser escrita integralmente em uma linha (ou seja, a palavra não pode ser separada silabicamente em duas linhas). Machado quer escrever um conto com o maior número de palavras possível, dentro das regras do concurso, e precisa de sua ajuda.

Dados o número máximo de caracteres por linha, o número máximo de linhas por página, e as palavras do conto que Machado está escrevendo, ele quer saber o número mínimo de páginas que seu conto utilizaria seguindo as regras do concurso.

Entrada

A primeira linha de um caso de teste contém três inteiros N, L e C, que indicam, respectivamente, o número de palavras do conto de Machado, o número máximo de linhas por página e o número máximo de caracteres por linha. O conto de Machado é inovador e não contém nenhum caractere além de letras maiúsculas e minúsculas e espaços em branco, sem letras acentuadas e sem cedilha. A segunda linha contém o conto de Machado, composto de N palavras separadas por espaços em branco; há espaço em branco somente entre duas palavras, e entre duas palavras há exatamente um espaço em branco.

Saída

Para cada caso de teste imprima uma única linha, contendo um único número inteiro, indicando o número mínimo de páginas que o conto de Machado ocupa, considerando as regras do concurso.

Restrições

- $2 \leq N \leq 1000$
- $1 \leq L \leq 30$
- $1 \leq C \leq 70$
- $1 \leq$ comprimento de cada palavra $\leq C$

Exemplos

Exemplo de entrada

```
14 4 20
Se mana Piedade tem casado com Quincas Borba apenas me daria uma esperanca colateral
16 3 30
No dia seguinte entrou a dizer de mim nomes feios e acabou alcunhando me Dom Casmurro
5 2 2
a de i de o
5 2 2
a e i o u
```

Saída para o exemplo de entrada

```
2
1
3
3
```

Solução

Em primeiro lugar, como o comprimento de cada palavra é menor ou igual ao comprimento de cada linha, é sempre possível escrever o conto obedecendo às restrições do concurso: por exemplo, Machado pode escrever uma palavra em cada linha.

Observe também que não importam exatamente **quais** são as palavras usadas por Machado, mas apenas o número de caracteres de cada uma. Isso permite simplificar o problema: não é preciso armazenar todas as palavras da entrada, mas apenas seus comprimentos (por exemplo, em um vetor de 1000 inteiros).

O comprimento de uma linha de N palavras, de comprimentos $c_1, c_2, \ldots, c_N$ é $c_1 + 1 + c_2 + 1 + \cdots + 1 + c_N$, onde os $\cdots + 1 + \cdots$ correspondem aos espaços em branco entre palavras consecutivas. Em particular, adicionar uma palavra de comprimento c_{N+1} a uma linha gasta $c_{N+1} + 1$ caracteres. Isso nos dá um algoritmo para contar o número total de *linhas* necessárias:

1. Inicialmente, temos só uma linha, cujo comprimento x é igual ao comprimento da primeira palavra, c_1.
2. Para cada palavra p_i: será verdade que $x + 1 + c_i \leq C$?
3. Se sim, a palavra p_i cabe na linha; faça $x \leftarrow x + 1 + c_i$;
4. Se não, a palavra p_i *não* cabe na linha: precisaremos de mais uma linha no conto. Essa linha, pelo menos inicialmente, só contém a palavra p_i: faça $x \leftarrow c_i$ e aumente o contador de linhas.

A ideia crucial aqui é a do passo 4: não importa quantas palavras já escrevemos ou faltam escrever: *só interessa o comprimento da última linha do conto*! Assim, não precisamos saber exatamente, o tempo todo, a disposição das palavras nas páginas; basta saber quanto espaço temos (ou no caso, quanto espaço já usamos) na última linha até então escrita, que é o papel da variável x.

E agora que sabemos quantas linhas o conto ocupa, como fazemos para contar o número de páginas? Se gastamos k linhas, precisamos de $\lceil \frac{k}{L} \rceil$ páginas, já que podemos colocar só L linhas por página (e não

podemos contar frações de página). Na maioria das linguagens de programação, a divisão de inteiros encontra o *piso*, e não o *teto*, da divisão, mas isso pode ser contornado usando a identidade

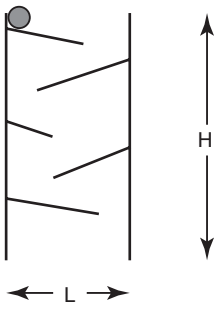
$$\left\lceil \frac{k}{L} \right\rceil = \left\lfloor \frac{k + (L-1)}{L} \right\rfloor$$

Problema B

Tobogan de bolinhas

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

Uma fábrica quer produzir um tobogan de brinquedo como o da figura abaixo, composto de duas hastes de madeira sustentando aletas que se alternam nas duas hastes. Uma bolinha de aço é solta na aleta mais alta do tobogan; sob efeito da gravidade, a bolinha desliza pelas aletas, terminando por sair do brinquedo.



O projeto do brinquedo, contendo as especificações do tamanho, posição e inclinação das hastes e de cada aleta, foi feito pelo dono da fábrica, e milhares de unidades já estão sendo confeccionadas na China. O gerente da fábrica foi incumbido de comprar as bolinhas de aço, mas antes de fazer o pedido das milhares de bolinhas quer saber o diâmetro máximo da bolinha, para que esta não pare no meio do brinquedo.

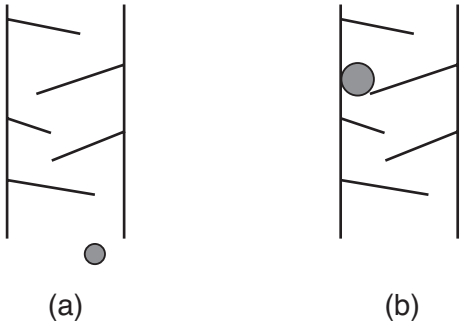


Figura 1: Dois exemplos: em (a) a bolinha chega ao final, e (b) a bolinha para no meio do brinquedo e não chega ao final.

O gerente da fábrica quer que você escreva um programa que, dadas as especificações do brinquedo, determine o diâmetro máximo da bolinha para que esta não pare no meio do brinquedo.

Entrada

A primeira linha de um caso de teste contém um inteiro N indicando o número de aletas do brinquedo. A segunda linha contém dois inteiros L e H, indicando respectivamente a distância entre as hastes e a altura das hastes do brinquedo. A haste esquerda do brinquedo está na posição 0 do eixo de coordenadas X, de forma que a haste direita está na posição L do eixo X.

Cada uma das N linhas seguintes descreve uma aleta. As aletas são descritas da mais alta para a mais baixa, de forma alternada em relação à haste na qual a aleta está conectada. A aleta mais alta

do brinquedo (a primeira a ser descrita) tem a extremidade ligada à haste esquerda; a segunda aleta mais alta (a segunda a ser descrita) tem a extremidade ligada à haste direita, assim alternadamente. As aletas ímpares têm a extremidade ligada à haste esquerda, as aletas pares têm a extremidade ligada à haste direita.

Cada aleta é descrita em uma linha contendo três números inteiros Y_i, X_f e Y_f, separados por um espaço em branco. (X_f, Y_f) indica a coordenada do final da aleta; para aletas ímpares a coordenada do início da aleta é $(0, Y_i)$, e para aletas pares a coordenada do início da aleta é (L, Y_i).

Para todas as aletas $Y_i > Y_f$ (ou seja, há um declive entre o início e o final da aleta), e o comprimento da aleta é menor do que a largura do brinquedo. Além disso, para duas aletas consecutivas A e B, $Y_{fA} >= Y_{iB}$ (ou seja, o final da aleta A tem altura maior do que ou igual ao início da aleta B).

Considere que as aletas são muito finas, de forma que a sua espessura pode ser desconsiderada, e que a sua largura é sempre maior do que o diâmetro da bolinha (ou seja, a bolinha sempre tem espaço lateral para deslizar pela aleta).

Saída

Para cada caso de teste imprima uma linha contendo um único número, com exatamente duas casas decimais, indicando o maior diâmetro de bolinha tal que esta consiga percorrer todo o brinquedo.

Restrições

- $1 \leq N \leq 10^3$
- $1 \leq L \leq 10^3$
- $1 \leq H \leq 10^3$
- $0 < X_f < L$
- $0 \leq Y_i \leq H$, $0 \leq Y_f \leq H$ e $Y_i > Y_f$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
3 6 10 9 3 8 6 2 5 4 3 1 3 5 10 9 3 7 7 2 4 2 3 0	2.00 1.41

Solução

Devido à configuração das aletas, basta determinar a distância horizontal entre o ponto da extremidade livre de cada aleta para a haste oposta, e entre esse mesmo ponto e a aleta seguinte na sequência

de aletas (distância ponto – segmento de reta). A resposta é o menor valor entre todas as distâncias calculadas.

Problema C

Cartões

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

Dois jogadores, Alberto e Wanderley, disputam um jogo. Um conjunto com um número par de cartões contendo números inteiros é disposto sobre uma mesa, um ao lado do outro, formando uma sequência. Alberto começa, e pode pegar um dos dois cartões das pontas. Wanderley então pode pegar um dos dois cartões das pontas e novamente Alberto pode pegar um cartão das pontas, e assim por diante, até Wanderley pegar o último cartão.

Alberto, o primeiro a jogar, tem como objetivo maximizar o número total de pontos que ele consegue, somando os valores dos cartões escolhidos. Wanderley, o segundo jogador, quer atrapalhar o Alberto e fazer com que ele consiga o menor número de pontos possível. Em suma, ambos querem fazer o melhor possível, Alberto querendo maximizar sua soma e Wanderley querendo minimizar a soma de Alberto.

Você deve escrever um programa que, dada a sequência de cartões, determine o maior número de pontos que Alberto consegue obter.

Entrada

Cada caso de teste é descrito em duas linhas. A primeira linha contém um inteiro, N, que indica o número de cartões sobre a mesa. A segunda contém N inteiros, que descrevem a sequência de cartões.

Saída

Para cada caso de teste seu programa deve imprimir uma única linha, contendo um único inteiro, o maior número de pontos que Alberto consegue obter.

Restrições

- $2 \leq N \leq 10^4$
- N é par
- cada um dos N inteiros pode ser representado com 32 bits.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
4 0 -3 5 10 4 0 -3 5 7 4 47 50 -3 7	10 7 57

Solução

Sejam $t_1, t_2, \ldots, t_n$ os n inteiros, n par. Basta fazer uma pd da seguinte forma:

$$f(i,j) = \begin{cases} \max(t_i, t_j), \text{ se } j = i+1 \\ \max(t_i + \min(f(i+1, j-1), f(i+2, j)), t_j + \min(f(i+1, j-1), f(i, j-2))). \end{cases}$$

A resposta será $f(1, n)$.

Dados os limites, as somas não passam de 10^{14} em valor absoluto.

Problema D

Coral Perfeito

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

A Maestrina do coral está planejando o espetáculo que apresentará na famosa Semana Brasileira de Corais. Ela pensou em preparar uma nova música, definida da seguinte maneira:

- cada um dos integrantes do coral inicia cantando uma nota, e somente muda de nota quando determinado pela Maestrina;
- ao final de cada compasso, a Maestrina determina que exatamente dois integrantes alterem a nota que cantam: um integrante passa a cantar a nota imediatamente acima da nota que cantava, e o outro integrante passa a cantar a nota imediatamente abaixo da nota que cantava;
- a música termina ao final do primeiro compasso em que todos os integrantes do coral cantam a mesma nota.

A Maestrina já tem várias ideias de como distribuir as notas no início da música entre os integrantes do coral, de forma a criar o efeito desejado. No entanto, ela está preocupada em saber se, dada uma distribuição de notas entre os integrantes, é possível chegar ao final da música da forma desejada (todos cantando a mesma nota) e, caso seja possível, qual o número mínimo de compassos da música. Você pode ajudá-la?

Entrada

A primeira linha de um caso de teste contém um inteiro N indicando o número de integrantes do coral. As notas serão indicadas por números inteiros. A segunda linha contém N números inteiros, indicando as notas iniciais que cada integrante deve cantar. As notas são dadas em ordem não decrescente de altura.

Saída

Para cada caso de teste imprima uma linha contendo um único número inteiro indicando o número mínimo de compassos que a música terá. Se não é possível terminar a música com todos os integrantes cantando a mesma nota, imprima o valor -1.

Restrições

- $2 \leq N \leq 10^4$
- $-10^5 \leq nota_i \leq 10^5$ para $0 \leq i \leq N-1$
- $nota_i \leq nota_{i+1}$ para $0 \leq i \leq N-2$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
3 1 2 3 4 3 6 9 12 6 1 2 3 4 5 723 5 10 10 10 10 10	2 -1 601 1

Solução

Primeiramente, você deve notar que somente será possível todos os cantores chegarem a uma mesma nota se a média dos valores dados na entrada for um número inteiro. Se a média não for um número inteiro, a resposta é `-1`.

Considere agora os cantores que iniciaram em notas com valores abaixo da média. Para cada cantor nessa situação, o número de compassos que ele necessitará para chegar à nota média é a diferença entre o valor de sua nota inicial e a nota média. Como a cada passo apenas um dos cantores que iniciaram em notas abaixo da média diminui a sua distância da nota média, o tempo total para que os cantores que iniciaram em notas abaixo da média cantem a nota média é a somatória das diferenças entre a média e cada nota menor do que a média.

Problema E

Elevador Espacial

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

A China está construindo um elevador espacial, que permitirá o lançamento de sondas e satélites a um custo muito mais baixo, viabilizando não só projetos de pesquisa científica como o turismo espacial.

No entanto, os chineses são muito supersticiosos, e por isso têm um cuidado muito especial com a numeração dos andares do elevador: eles não usam nenhum número que contenha o dígito "4" ou a sequência de dígitos "13". Assim, eles não usam o andar 4, nem o andar 13, nem o andar 134, nem o andar 113, mas usam o andar 103. Assim, os primeiros andares são numerados 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16, ...

Como o elevador espacial tem muitos andares, e eles precisam numerar todos os andares do elevador, os chineses pediram que você escrevesse um programa que, dado o andar, indica o número que deve ser atribuído a ele.

Entrada

Cada caso de teste consiste de uma única linha, contendo um inteiro N que indica o andar cujo número deve ser determinado.

Saída

Para cada caso de teste, imprima uma linha contendo um único número inteiro indicando o número atribuído ao N-ésimo andar.

Restrições

- $1 \leq N \leq 10^{18}$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
1 4 11 12 440	1 5 12 15 666

Solução

O problema é determinar o N-ésimo inteiro positivo que não contém as sequências de dígitos "4" ou "13" na sua representação decimal.

Por inclusão/exclusão, é fácil calcular quantos números da forma "$1xx\ldots x$" e "$xx\ldots x$" são botões válidos de elevador em função apenas da quantidade de 'x'; esses valores podem ser pré-computados e compilados num vetor no código do programa.

A partir daí é fácil calcular quantos números válidos existem, por exemplo, de 1 a 2816: podemos contar os inteiros válidos nos intervalos $[0, 999]$, $[1000, 1999]$, $[2000, 2099]$, $[2100, 2199]$, $\ldots$, $[2700, 2799]$, $[2800, 2809]$, $[2810, 2810]$, $[2811, 2811]$, $\ldots$, $[2816, 2816]$.

Problema F

Perdido na Noite

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

Numa cidade da Nlogônia, o sistema viário é composto de N rotatórias e $N-1$ ruas, sendo que cada rua liga duas rotatórias distintas. Utilizando o sistema viário, é possível ir de qualquer rotatória para qualquer outra rotatória da cidade.

A cidade possui apenas dois hotéis: um barato, localizado na rotatória B, e um caro, localizado na rotatória C. Um turista veio à cidade para celebrar o aniversário de um amigo, cuja festa está sendo realizada em um clube localizado na rotatória A. Como o turista não fez reserva em nenhum dos hotéis e a noite está agradável, após a festa ele decidiu passear a pé pelas ruas e rotatórias até encontrar um dos hotéis (ele também decidiu hospedar-se no primeiro hotel que encontrar).

Seu plano foi dificultado porque como ele não conhece a cidade e bebeu um pouco além da conta, todas as ruas lhe parecem iguais. Assim, ele decidiu usar a seguinte estratégia: a cada rotatória ele escolhe, com probabilidade uniforme, uma das ruas que saem da rotatória, e usa essa rua para ir a uma outra rotatória, até chegar à rotatória onde um dos hotéis está localizado. Note que como o turista não consegue distinguir as ruas, pode ocorrer de ele escolher a mesma rua pela qual chegou à rotatória.

Você deve escrever um programa que, dadas a descrição do sistema viário, a localização A da festa de aniversário, a localização B do hotel barato e a localização C do hotel caro, determine a probabilidade de o turista chegar ao hotel barato antes de chegar ao hotel caro.

Entrada

A primeira linha de um caso de teste contém quatro inteiros N, A, B e C, indicando respectivamente o número de rotatórias do sistema viário, a rotatória onde a festa de aniversário foi realizada, a rotatória onde o hotel barato está localizado, e a rotatória onde o hotel caro está localizado. Cada uma das $N-1$ linhas seguintes contém dois inteiros X e Y, indicando que existe uma rua que liga as rotatórias X e Y.

Saída

Seu programa deve imprimir uma única linha, contendo a probabilidade de o turista chegar ao hotel barato antes de chegar ao hotel caro, com 6 casas decimais.

Restrições

- $3 \leq N \leq 100$
- $B \neq C$, $A \neq B$, $A \neq C$.
- $1 \leq A, B, C \leq N$
- $1 \leq X, Y \leq N$.
- $X \neq Y$.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
4 1 2 3 1 4 2 4 3 4 5 3 1 5 1 2 2 3 3 4 4 5	0.500000 0.500000

Solução

Note que, pela descrição, o sistema viário pode ser modelado como uma árvore T onde as rotatórias são os vértices e as ruas são as arestas.

Três soluções possíveis para o problema são:

- Seja P o caminho entre B e C, de comprimento $l_{B,C}$. Seja w o vértice mais próximo de A que faz parte de P (possivelmente $A = w$). Seja l_w a distância entre B e w. Então a resposta é dada por $l_w/l_{B,C}$.
- Considerar a árvore T como uma cadeia de Markov, adicionando self-loops de probabilidade 1 em B e C. Encontrar a distribuição estacionária.
- Analogia com circuitos: considerando cada aresta como um resistor de 1 ohm, aplicando-se uma tensão de 1V entre B e C, a tensão entre um vértice x e C nos dá a probabilidade de chegar antes em B que em C.

Problema G

Grid de Largada

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

Na Nlogônia, vai ser realizada a sensacional final mundial da fórmula 17. Os competidores se alinham na largada e disputam a corrida. Você vai ter acesso aos grids de largada e de chegada. A questão é determinar o número mínimo de ultrapassagens que foram efetuadas durante a competição.

Entrada

Cada caso de teste utiliza três linhas. A primeira linha de um caso de teste contém um inteiro N indicando o número de competidores. Cada competidor é identificado com um número de 1 a N. A segunda linha de cada caso tem os N competidores, em ordem do grid de largada. A terceira linha de cada caso tem os mesmos competidores, porém agora na ordem de chegada.

Saída

Para cada caso de teste imprima uma linha contendo um único número inteiro, que indica o número mínimo de ultrapassagens necessárias para se chegar do grid de largada ao grid de chegada.

Restrições

- $2 \leq N \leq 24$

Exemplos

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

Solução

Para descrever a solução deste problema, vamos inicialmente definir um conceito importante para o problema de ordenação. Uma *inversão* em uma sequência de elementos $A[1..N]$ é um par (i, j) para o qual $i < j$ e $A[i] > A[j]$. O número de inversões de uma sequência indica portanto o número de pares de elementos que estão "fora de ordem". O número de inversões é muito usado como medida da ordenação de uma sequência, e é importante na análise da complexidade de algoritmos de ordenação.

Considere os grids de largada e chegada como vetores. Considere ainda que o vetor de chegada representa a ordenação correta dos elementos. Então, o número mínimo de ultrapassagens é igual ao

número de *inversões* do vetor de chegada em relação ao vetor de largada, ou seja, o número de trocas (ultrapassagens) necessárias para "ordenar corretamente" o vetor de largada.

O algoritmo mais simples para contar o número de inversões é aplicar diretamente a definição de inversão: para cada elemento $A[j]$ do vetor, conte quantos elementos existem tais que $A[i] > A[j]$ e $i < j$. Note que esse algoritmo tem complexidade N^2, mas pode ser utilizado considerando os limites deste problema.

Algoritmos de ordenação padrão também podem ser modificados para contar o número de inversões de um vetor. Considerando os limites do problema, qualquer algoritmo de ordenação pode ser usado como solução. Por exemplo, pode-se utilizar o método da bolha (*bubble sort*), que é muito simples de implementar e no pior caso tem complexidade $O(N^2)$, mas no melhor caso tem complexidade $O(N)$. Utilize um contador de trocas, inicialmente zero. Em cada passo, compare cada par de elementos vizinhos do vetor, efetuando a troca (e atualizando o contador) se necessário. O algoritmo termina quando em um passo não houver necessidade de troca, indicando que o vetor já está ordenado. A resposta para o problema é o valor do contador de trocas. Note que o algoritmo basicamente "simula" as ultrapassagens da corrida.

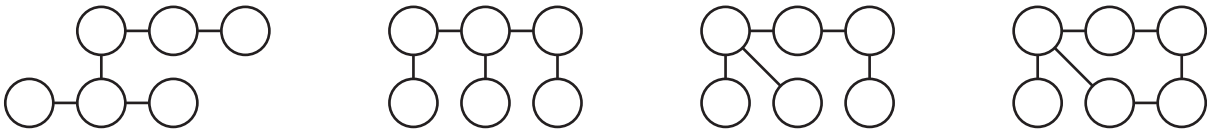
Problema H

Combate ao câncer

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

Pesquisadores da Fundação Contra o Câncer (FCC) anunciaram uma descoberta revolucionária na Química: eles descobriram como fazer átomos de carbono ligarem-se a qualquer quantidade de outros átomos de carbono, possibilitando a criação de moléculas muito mais complexas do que as formadas pelo carbono tetravalente. Segundo a FCC, isso permitirá o desenvolvimento de novas drogas que poderão ser cruciais no combate ao câncer.

Atualmente, a FCC só consegue sintetizar moléculas com ligações simples entre os átomos de carbono e que não contêm ciclos em suas estruturas: por exemplo, a FCC consegue sintetizar as moléculas (a), (b) e (c) abaixo, mas não a molécula (d).



(a) (b) (c) (d)

Devido à agitação térmica, uma mesma molécula pode assumir vários formatos. Duas moléculas são *equivalentes* se for possível mover os átomos de uma das moléculas, sem romper nenhuma das ligações existentes nem criar novas ligações químicas, de forma que ela fique exatamente igual à outra molécula. Por exemplo, na figura acima, a molécula (a) não é equivalente à molécula (b), mas é equivalente à molécula (c).

Você deve escrever um programa que, dadas as estruturas de duas moléculas, determina se elas são equivalentes.

Entrada

A primeira linha de um caso de teste contém um inteiro N indicando o número de átomos nas duas moléculas. Os átomos são identificados por números inteiros de 1 a N. Cada uma das $2N - 2$ linhas seguintes descreve uma ligação química entre dois átomos: as primeiras $N - 1$ linhas descrevem as ligações da primeira molécula; as $N - 1$ últimas descrevem as ligações químicas da segunda molécula. Cada linha contém dois inteiros A e B indicando que existe uma ligação química entre os átomos A e B.

Saída

Para cada caso de teste seu programa deve imprimir uma única linha, contendo um único caractere: `S` se as moléculas são equivalentes ou `N` caso contrário.

Restrições

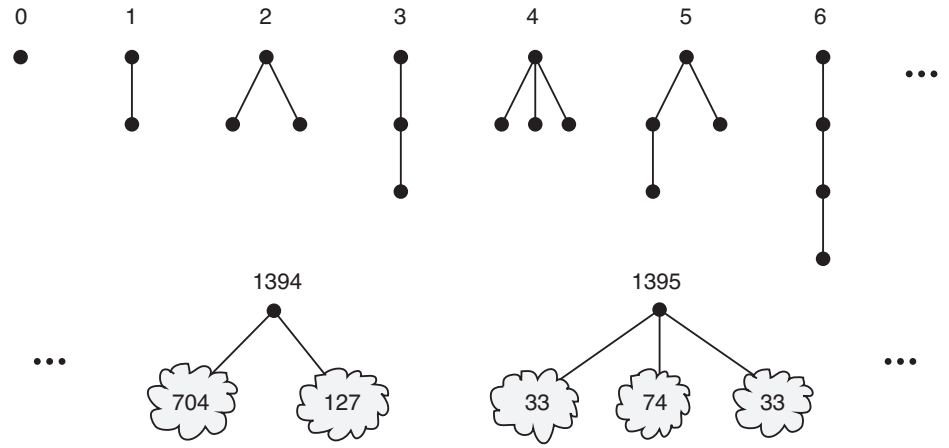
- $2 \leq N \leq 10^4$
- $1 \leq A, B \leq N$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
7 1 2 2 3 3 4 4 5 6 2 7 3 1 2 2 3 3 4 4 5 6 2 7 4 6 1 2 1 3 2 4 2 5 3 6 2 4 5 3 6 4 1 4 5 6	N S

Solução

Obviamente, duas árvores *enraizadas* são isomorfas se e somente se existe uma bijeção entre os filhos de cada raiz de forma que cada par de subárvores correspondente também seja isomorfo. Testar todas as ordens possíveis dos filhos para um eventual isomorfismo é impraticável, mas note que o conjunto das árvores enraizadas é enumerável, i.e. podemos atribuir a cada árvore enraizada um número inteiro.



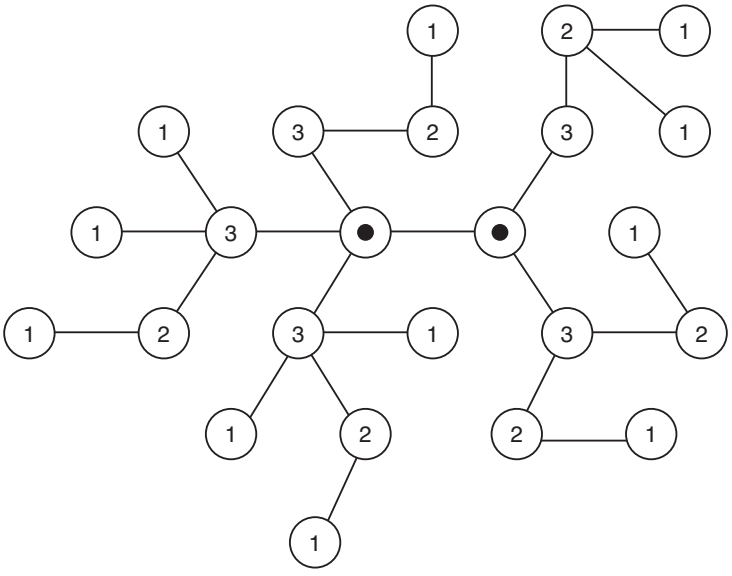
Isso resolve o problema da ordem dos filhos: duas árvores enraizadas são isomorfas se e somente se ambas as raízes tem o mesmo multiconjunto de filhos; *essa* comparação pode ser feita ordenando as árvores em ordem crescente de inteiro associado.

A enumeração das árvores não precisa ser feita completamente (isso seria computacionalmente inviável): crie um dicionário que mapeia multiconjuntos de inteiros a inteiros, representando o identificador de cada árvore, dados os filhos de sua raiz.

$$
\begin{aligned}
\{\} &\rightarrow 0 \\
\{0\} &\rightarrow 1 \\
\{0,0\} &\rightarrow 2 \\
\{1\} &\rightarrow 3 \\
\{0,0,0\} &\rightarrow 4 \\
\{0,1\} &\rightarrow 5 \\
\{3\} &\rightarrow 6 \\
\{0,0,0,0\} &\rightarrow 7 \\
&\dots \\
\{127,704\} &\rightarrow 1394 \\
\{33,33,74\} &\rightarrow 1395 \\
&\dots
\end{aligned}
$$

Se processarmos os vértices em ordem decrescente de nível, ao processar um dado vértice, todos os seus filhos já tiveram identificadores atribuídos a eles; se o multiconjunto dos filhos já existir no dicionário, reusamos o identificador; caso contrário, criamos um novo identificador para aquela árvore.

O que fazer no caso em que a árvore *não* é enraizada? Toda árvore tem no máximo dois vértices chamados *centros*: para achá-los, marque todas as folhas da árvore e depois remova-as; repita o processo até obter o grafo vazio. Os últimos vértices removidos são os centros da árvore (exercício: por que um grafo não pode ter mais de dois centros?).



Enraizando a primeira árvore em um de seus centros, basta testar se a primeira árvore (agora enraizada) é isomorfa à segunda árvore enraizada em algum de seus centros. Como a segunda árvore tem no máximo dois centros, isso só introduz um fator constante no tempo de execução do algoritmo.

Problema I

Integral

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

Dado um inteiro positivo n, denotaremos por $[n]$ o intervalo real $\{x : 0 \leq x \leq n\}$. Uma função $f : [n] \Rightarrow \mathcal{R}$ é parcialmente especificada, sendo fornecidos valores de f apenas em pontos de um subconjunto S de $[n]$.

O conjunto S satisfaz as seguintes propriedades:

1. Os pontos em S são todos inteiros.
2. Os extremos 0 e n de $[n]$ estão ambos em S.

A função f satisfaz as seguintes propriedades:

1. Os valores de f nos pontos inteiros de $[n]$ são inteiros.
2. Para cada ponto inteiro x em $[n] \setminus S$ (ou seja, nos pontos inteiros de $[n]$ que não estão em S), a função f é monótona no intervalo $[x-1, x+1]$. Em outras palavras, pelo menos uma das desigualdades $f(x-1) \leq f(x) \leq f(x+1)$ ou $f(x-1) \geq f(x) \geq f(x+1)$ é satisfeita.
3. Para cada ponto não inteiro x em $[n]$, o valor de $f(x)$ é dado pela interpolação linear de $f(\lfloor x \rfloor)$ e $f(\lceil x \rceil)$, isto é, $f(x) = (x - \lfloor x \rfloor) f(\lfloor x \rfloor) + (\lceil x \rceil - x) f(\lceil x \rceil)$.

Temos ainda a liberdade de especificar os valores de f nos pontos inteiros de $[n] \setminus S$ (note no entanto que S pode conter todos os pontos inteiros de $[n]$). Gostaríamos de utilizar essa flexibilidade para fazer com que $\int_0^n f(x)dx = y$, isto é, a área sob $f(x)$ entre os extremos 0 e n seja igual a y, um valor dado.

Seu problema então é decidir se isso é possível ou não.

Entrada

A primeira linha de um caso de teste contém três inteiros, N, M e Y, respectivamente a amplitude do intervalo, o tamanho do conjunto S e o valor de y. Cada uma das M linhas seguintes descreve a função em um ponto de S, contendo dois inteiros X e F, representando $f(X) = F$. Os valores de X não estão necessariamente em ordem crescente.

Saída

Para cada caso de teste, determine se existe atribuição de valores a $f(x)$ para os pontos inteiros $x \in [n] \setminus S$ tal que $\int_0^n f(x)dx = y$, isto é, a área sob $f(x)$ entre os extremos 0 e n seja igual a y. Em caso negativo, seu programa deve imprimir uma linha contendo apenas o caractere '`N`'. Em caso afirmativo, seu programa deve imprimir uma linha contendo o caractere '`S`', seguido dos valores de $f(x)$ para os pontos inteiros $x \in [n] \setminus S$, em ordem crescente de valores de x. O caractere inicial e os valores seguintes, se houver, devem ser separados por um espaço em branco. Caso mais de uma solução seja possível, imprima aquela que for lexicograficamente menor.

Restrições

- $1 \leq N \leq 10^6$
- $0 \leq X \leq N$, X inteiro, $\forall X \in S$

- $0 \leq F \leq 10^6$, F inteiro
- $0 \leq Y \leq 10^9$, Y inteiro
- $\int_0^n f(x)dx \leq 10^9$ para qualquer atribuição de valores a $f(x)$ para $x \in [n] \setminus S$ satisfazendo as restrições do enunciado.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
5 6 10 0 2 1 2 5 2 2 2 3 2 4 2 5 2 10 0 0 5 10 2 2 5 0 1 2 2 10 3 18 0 2 6 4 10 0 2 2 1 0 0 2 1	S S 0 0 0 5 N S 2 2 2 2 2 1 1 1 N

Solução

Inicialmente testa-se todas as variáveis no seu valor mínimo e no seu valor máximo para verificar se o valor de y está entre os dois. Em caso afirmativo, seta-se todas para o valor máximo e, em ordem lexicográfica, vamos setando o valor das variáveis para seu valor mínimo. Quando o valor da integral ficar menor que o valor de y, sabe-se que esta variável é a última a ser ajustada e escolhe-se o valor adequado.

Problema J

Palavras

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

Dados dois conjuntos de palavras formadas por zeros e uns, você deve escrever um programa para determinar se existem concatenações de palavras de cada um dos conjuntos que gerem uma mesma palavra. Por exemplo, se um conjunto A contém as palavras `010` e `11` e outro conjunto B contém as palavras `0` e `101`, então a palavra `01011010` pode ser formada tanto por contatenações de palavras de A como por contatenações de palavras de B:

$$\texttt{010} \cdot \texttt{11} \cdot \texttt{010} = \texttt{01011010} = \texttt{0} \cdot \texttt{101} \cdot \texttt{101} \cdot \texttt{0}$$

Entrada

A primeira linha de um caso de teste contém dois inteiros, N_1 e N_2, que indicam respectivamente o número de palavras do primeiro e do segundo conjunto de palavras. Cada uma das N_1 linhas seguintes contém uma palavra do primeiro conjunto. Cada uma das N_2 linhas seguintes contém uma palavra do segundo conjunto.

Saída

Para cada caso de teste seu programa deve imprimir uma única linha, contendo um único caractere. Se for possivel encontrar uma concatenação de uma ou mais palavras do primeiro conjunto que seja igual a uma concatenação de uma ou mais palavras do segundo conjunto então o caractere deve ser `S`, caso contrário deve ser `N`.

Restrições

- $1 \leq N_1, N_2 \leq 20$
- cada palavra tem no mínimo um caractere e no máximo 40 caracteres, todos zeros e uns.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
2 2 010 11 0 101 3 1 1 00 000 01 1 1 00 000	S N S

Solução

Seja S_1 e S_2 os dois conjuntos dados. Então, para $i = 1, 2$, $L_i = S_i^+$, ou seja, L_i é o conjunto das palavras formadas pelo produto de uma ou mais palavras de S_i, com possíveis repetições. Uma solução consiste em construir os dois autômatos finitos, A_1 e A_2, que reconhecem as linguagens L_1 e L_2, respectivamente. Em seguida, determina-se se a interseção das linguagens formadas pelos dois autômatos é ou não vazia.

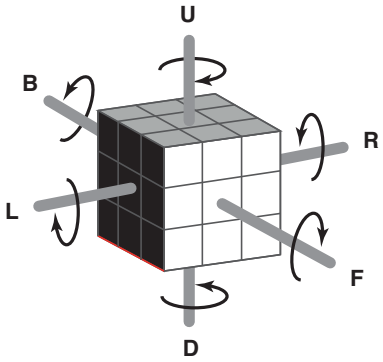
Problema K

Ciclo de Rubik

Arquivo: ***rubik.[c|cpp|java]***

Provavelmente todos conhecem o Cubo de Rubik, um passatempo 3-D desafiador, que tem cada uma das seis faces cobertas com nove etiquetas, cada etiqueta de uma cor (azul, amarelo, laranja, branco, verde e vermelho). No estado inicial, todas as nove etiquetas de uma face têm a mesma cor. Um mecanismo engenhoso permite que cada face seja rotacionada independentemente, fazendo com que as cores das etiquetas nas faces possam ser misturadas.

Cada uma das faces do Cubo de Rubik é denotada por uma letra: `F`, `B`, `U`, `D`, `L`, e `R`, como ilustrado na figura abaixo.



A rotação de uma face é denominada de um *movimento*. Para descrever os movimentos utilizamos as letras identificadoras das faces:

- uma letra maiúscula representa um giro de 90^o no sentido horário da face correspondente;
- uma letra minúscula representa um giro de 90^o no sentido anti-horário da face correspondente.

Por exemplo, `F` representa um giro de 90^o no sentido horário da face `F`; `r` representa um giro de 90^o no sentido anti-horário da face `R`. Uma sequência de movimentos é denotada por uma sequência de letras identificadoras de faces. Assim, `rDF` representa um giro de 90^o no sentido anti-horário da face `R`, seguido de um giro de 90^o no sentido horário da face `D`, seguido de um giro de 90^o no sentido horário da face `F`.

Uma propriedade interessante do Cubo de Rubik é que qualquer sequência de movimentos, se aplicada repetidas vezes, faz com que o cubo retorne ao estado original (estado que tinha antes da primeira aplicação da sequência). Por exemplo, após quatro aplicações da sequência `B` o cubo retorna ao estado original.

Você deve escrever um programa que, dada uma sequência de movimentos, determine o menor número de aplicações completas dessa sequência para que o cubo retorne ao seu estado original.

Entrada

Cada caso de teste é descrito em uma única linha, que contém a sequência de movimentos.

Saída

Para cada caso de teste seu programa deve imprimir uma única linha, contendo um único inteiro, indicando o menor número de aplicações completas da sequência para que o cubo retorne ao seu estado original.

Restrições

- Cada sequência tem no mínimo um movimento e no máximo 80 movimentos.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**

`Rr`
`LLL`
`dl`
`RUUdBd`
	`1`
	`4`
	`105`
	`1260`

Solução

Uma sequência de movimentos no cubo de Rubik é uma permutação dos $9 \times 6 = 54$ quadradinhos coloridos nas faces do cubo (nem todas as permutações são atingíveis, mas isto não é importante). O período de uma dessas permutações é o m.m.c. dos ciclos desta permutação.

A maior dificuldade deste problema é a implementação dos movimentos de cada face — cada rotação altera as posições de 20 dos 54 quadradinhos do cubo!

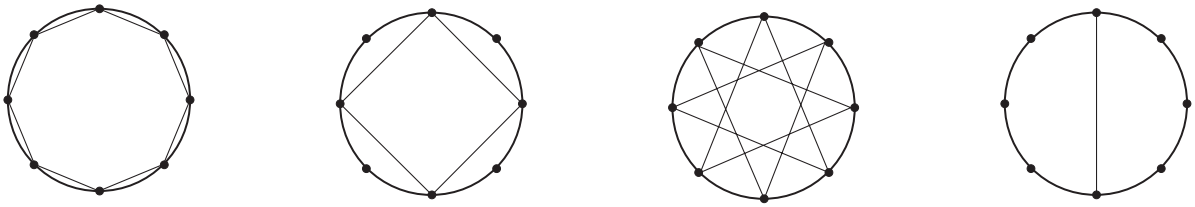
Problema L

Estrela

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

Fernando ganhou um compasso de aniversário, e agora sua diversão favorita é desenhar *estrelas*: primeiro, ele marca N pontos sobre a circunferência, dividindo-a em N arcos iguais; depois, ele liga cada ponto ao k-ésimo ponto seguinte, até voltar ao ponto inicial.

Dependendo do valor de k, Fernando pode ou não atingir todos os pontos marcados sobre a circunferência; quando isto acontece, a estrela é chamada de *completa*. Por exemplo, quando $N = 8$, as possíveis estrelas são as mostradas no desenho abaixo; as estrelas (a) e (c) são completas, enquanto as estrelas (b) e (d) não o são.



(a) (b) (c) (d)

Dependendo do valor de N, pode ser possível desenhar muitas estrelas diferentes; Fernando pediu que você escrevesse um programa que, dado N, determina o número de estrelas completas que ele pode desenhar.

Entrada

Cada caso de teste contém de uma única linha, contendo um único inteiro N, indicando o número de arcos no qual a circunferência foi dividida.

Saída

Para cada caso de teste, seu programa deve imprimir uma única linha contendo um único inteiro, indicando o número de estrelas completas que podem ser desenhadas.

Restrições

$3 \leq N < 2^{31}$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
3 4 5 18 36 360 2147483647	1 1 2 3 6 48 1073741823

Solução

O número de estrelas completas é igual à metade do número de inteiros k coprimos com n no intervalo $0 < k < n$.

Dois números a e b são ditos coprimos se não há nenhum divisor em comum a ambos diferente de 1. Para notar que será necessário ver usar coprimos, note que, se você pegar algum k que não seja coprimo a n, você conseguirá gerar um ciclo de tamanho $\frac{n}{k}$, em outras palavras, em $\frac{n}{k}$ passos, consegue-se chegar ao ponto 0, onde o próximo ponto seria novamente o k, e, portanto, haveria uma repetição.

Primeiro, fatoramos $N = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_r^{e_r}$ em r fatores primos distintos $p_1, p_2, \ldots, p_r$, onde $r \geq 1$, $e_1, e_2, \ldots, e_r \geq 1$.

Posteriormente, vamos lembrar alguns conceitos de teoria dos números. Euler já definira uma função que, para um número n, descreve quantos coprimos menores ou iguais a n existem. Esta função é definida como $\varphi(n)$, também conhecida como a função totiente de Euler. Para acharmos o número de coprimos, temos que lembrar de duas propriedades importantes desta função:

- $\varphi(p^k) = p^{k-1} \times (p-1)$ para $k \geq 1$ e p primo
- $\varphi(a * b) = \varphi(a) \times \varphi(b)$, para a e b coprimos

Usando isto, temos então que $\varphi(n) = (p_1 - 1)p_1^{e_1-1} \cdot (p_2 - 1)p_2^{e_2-1} \cdot \ldots \cdot (p_r - 1)p_r^{e_r-1}$.

Por fim, temos que notar que, ao escolher o número de coprimos, para metade deles, haverá repetição de estrelas completas distintas. Uma prova direta para isto é notar que, para um k coprimo a n, então $n - k$ também será coprimo a n (numa demonstração prática, seria como se você, ao invés de ter começado puxando k para mais, puxasse k para menos, o que daria $n - k$, que seria o último passo da iteração começando com k a mais).

Temos, portanto, que a resposta é $\frac{\varphi(n)}{2}$.