I Escola Regional de Informática do Rio de Janeiro (ERI-RJ 2010)

Meia Maratona de Programação

Caderno de Questões

Problema A
(Hipertenusa)

fonte: `hipertenusa.c`, `hipertenusa.cpp` *ou* `hipertenusa.java`

Descrição

Joe Nísio atualmente cursa a disciplina Estudando Retas e Inteiros. Nessa disciplina, são abordadas as relações entre números inteiros e a geometria inerente a eles. Na primeira aula foram estudados triângulos pitagóricos, triângulos retângulos que possuem seus três lados, os dois catetos e a hipotenusa, como números inteiros. Além disso, o professor Pablo Topázio mostrou que números que são hipotenusas de algum triângulo retângulo satisfazem uma série de propriedades.

Joe, muito interessado e criativo, se fez a seguinte pergunta: quais seriam os números inteiros que são o quadrado da hipotenusa de pelo menos dois triângulos retângulos distintos com catetos inteiros? Ele batizou um número que satisfaz essa propriedade de hipertenusa. Por exemplo, 50 é uma hipertenusa, uma vez que $50 = 1^2 + 7^2 = 5^2 + 5^2$.

Interessado em obter uma resposta para essa pergunta, ele conversou com Hiago Bota e Pérola Silvia, seus companheiros de time da maratona de triangulação (uma competição exótica onde times de três pessoas devem resolver problemas sobre triângulos em uma determinada quantidade de tempo), e pediu-os para ajudá-lo a escrever um programa respondesse ao questionamento. Ao descobrir que não sabem resolver esse problema e receosos que algo parecido acabe aparecendo na próxima maratona de triangulação, eles pediram que você os ajudasse.

Entrada

A entrada é composta por diversas linhas de entrada. A primeira linha contém um inteiro $T \leq 1000000$, o número de casos de teste. Cada uma das T linhas seguintes contém um número $1 \leq N \leq 10000000$, o número que deve ser checado se é uma hipertenusa ou não.

Saída

Para cada linha da entrada deve ser impresso "sim", caso o número seja uma hipertenusa e "nao" caso contrário.

Exemplo

Entrada	**Saída**
4 37 43 50 51	nao nao sim nao

Problema B
(Robôs Inteligentes)

fonte: `robos.c`, `robos.cpp` *ou* `robos.java`

Descrição

No Encontro de Robôs Inteligentes do Rio de Janeiro, haverá uma pequena competição de programação de robôs. Nessa competição, cada time participante deverá programar seu robô para percorrer um labirinto cheio de tesouros. O time vencedor será aquele cujo robô conseguir coletar o maior número de tesouros.

Vários labirintos diferentes já foram criados, porém ninguém sabe ao certo quantos tesouros são coletáveis em cada um deles. O organizador dessa competição, o Sr. Benício Prantos, está muito ocupado procurando patrocínio e divulgando o evento, e por isso pediu a sua ajuda para contar os tesouros.

Dado um mapa de um labirinto com a posição do robô e as posições de "tesouros" espalhados pelo labirinto, o seu programa deve imprimir quantos tesouros podem ser coletados a partir da posição inicial. Um tesouro pode ser coletado apenas se o robô consegue alcançar sua posição através de movimentos para norte, sul, leste ou oeste, em qualquer quantidade ou ordem, passando apenas por posições livres. A borda mais externa do labirinto sempre conterá paredes para que, dessa forma, o robô nunca se perca, saindo do labirinto.

Entrada

A entrada é composta por diversas instâncias. Cada uma das instâncias começa com dois inteiros $3 \leq m, n \leq 1000$, respectivamente o comprimento e a largura de um labirinto. As m linhas seguintes descrevem o labirinto e contêm n caracteres cada uma. Cada caractere '#' representa uma parede, ou seja, uma posição que não pode ser ocupada pelo robô, ao contrário das posições ocupadas pelo caractere '.' (ponto). Um caractere '$' representa um tesouro e um caractere 'I' representa a posição inicial do robô. Haverá exatamente um caracter 'I' em cada instância. A entrada termina com um par de inteiros m, n com $m = n = 0$. Este caso não deve ser processado.

Saída

Para cada instância da entrada, deve ser impressa uma única linha contendo a string "C/T", onde C é a quantidade de tesouros coletáveis e T é a quantidade total de tesouros no mapa.

Exemplo

Entrada	**Saída**
7 11 ########### #..I...#.$# #.###..$..# #.#.###.$.# #.#.#.$.$.# #.....#.#.# ########### 7 13 ############# #$$#........# ####..#####.# #.I.$.#.$.#.# ####..#####.# #$$#........# ############# 0 0	5/5 1/6

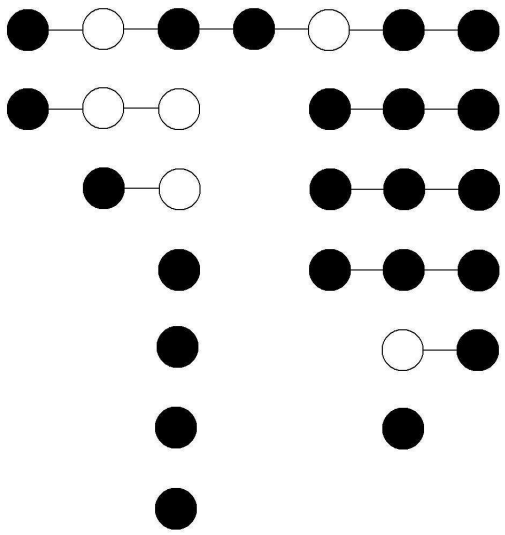
Problema C
(Jogo da Virada)

fonte: `virada.c`, `virada.cpp` *ou* `virada.java`

Descrição

A filial carioca da Entidade Recreativa Internacional, ERI-RJ, está promovendo um velho jogo: o Jogo da Virada, um jogo de quebra-cabeça que exige um pouco de raciocínio. Para distribuí-lo, é necessário criar uma série de configurações iniciais do jogo. O ideal é que as configurações fossem tão diversas quanto possível, mas todas as configurações deveriam ser resolvíveis. Infelizmente, nenhum membro da ERI-RJ é muito bom no jogo e têm sido difícil decidir, para cada configuração, se existe uma solução. Como você é um bom resolvedor de problemas, você é a única esperança de se agilizar o proceso de verificação das configurações.

O jogo consiste de peças circulares dispostas em uma linha reta. Cada uma das peças circulares possui duas cores, preto e branco, cada cor em um dos lados. A cada lance do jogo, uma peça deve ser removida. Entretando, uma peça só pode ser removida caso esteja com o lado preto para cima. Ao removermos uma peça, cada uma de suas duas peças vizinhas (se existirem e ainda não tiverem sido removidas), são viradas, de forma que sua cor seja alterada. Uma configuração é dita "válida"se existe uma solução, um seja, uma sequência de passos que consiga remover todas as peças. Sua tarefa é criar um programa que diga, para cada configuração, se ela é válida. Na figura, é possível visualizar uma sequência de passos que resolve uma configuração inicial. Esta figura representa a configuração do primeiro caso de teste no exemplo.



Entrada

A entrada consiste de um conjunto de configurações, com cada uma delas ocupando duas linhas. A primeira delas contém um único inteiro, $1 \leq N \leq 100000$, dizendo o número de peças compondo uma configuração. A segunda delas contém N números separados por espaço. Um número 0 representa que a peça está com o lado branco para cima e 1 representa o lado preto. O conjunto de configurações termina $N = 0$ e essa configuração não deve ser processada.

Saída

Para cada configuração, deve ser impressa uma única linha na saída, contendo o número 1 caso a configuração seja válida e 0 em caso contrário.

Exemplo

Entrada	Saída
7 1 0 1 1 0 1 1 1 1 2 0 0 0	1 1 0

Problema D
(Frações Equivalentes)

fonte: `fracoes.c`, `fracoes.cpp` *ou* `fracoes.java`

Descrição

Um professor da disciplina Estudando Racionais e Irracionais, ERI, dá aula sobre frações para seus alunos. A próxima aula será sobre frações equivalentes e ele gostaria de ter um programa para os alunos brincarem com os números. Como ele não sabe programar, ele pediu que você fizesse um programa que decidisse se duas frações são equivalentes. Para te ajudar na tarefa ele lhe deu uma dica de como verificar se duas frações são equivalentes. Dizemos que $\frac{x}{y}$ e $\frac{z}{w}$ são equivalentes se $x \times w = y \times z$.

Entrada

A primeira linha da entrada contém um inteiro positivo N ($N \leq 1000$). Cada uma das N linhas seguintes contém 4 números inteiros positivos, x, y, z e w, todos maiores ou iguais a 1 e menores ou iguais a 100.

Saída

Para cada quádrupla x, y, z, w, você deve imprimir uma linha contendo "SIM" caso as frações sejam equivalentes e "NAO" (sem acento) em caso contrário.

Exemplo

Entrada	**Saída**
3 1 2 3 4 1 2 2 4 50 50 100 100	NAO SIM SIM

Problema E
(Faces e Arestas)

fonte: `faces.c`, `faces.cpp` *ou* `faces.java`

Descrição

No evento Esculturas Reconhecidas Internacionalmente, vários artistas enviam seus projetos de escultura para serem construídos em tamanho grande e apreciados por todos os visitadores.

Para que isso seja possível, são impostas algumas limitações às esculturas aceitas no evento:

Todas as esculturas podem ser construídas a partir de polígonos planos (triângulos, quadrados, etc). As esculturas serão construídas a partir de uma estrutura de arame, e sobre essa estrutura serão fixadas as placas planas nos formatos dos polígonos. Todas as esculturas são "fechadas", de modo que não seja possível ver a estrutura interna delas. Para o próximo evento, todas as esculturas que serão expostas já foram submetidas, restando apenas encomendar o material necessário para construí-las. Este material é formado por placas cortadas no formato dos polígonos desejados e por barras retas que formam a estrutura interna de arame.

Pedro, responsável pelas encomendas, recebeu a lista com todas as placas necessárias para cada escultura, porém perdeu a lista que continha a quantidade de barras necessárias para cada placa. Em vez de requisitar outra lista (o que pode demorar alguns dias), ele pensou se poderia reconstruir a lista de barras a partir da lista de polígonos.

Entrada

A entrada será composta por vários casos de teste. A primeira linha de cada caso de teste contém um número inteiro $1 \leq n \leq 100$, que indica quantos tipos diferentes de polígonos serão encomendados. As n linhas seguintes contêm, cada uma, dois números inteiros a, b ($2 \leq a \leq 100$, $1 \leq b \leq 1000$), que indicam respectivamente a quantidade de lados do polígono e quantos polígonos desse tipo são necessários.

A entrada termina com uma linha com $n = 0$, que não deve ser processada.

Saída

Para cada caso de teste deve ser impresso uma única linha, contendo o número de barras metálicas para construir a estrutura de arame da escultura.

Exemplo

Entrada	**Saída**
4 3 1 4 23 10 2 9 1 1 4 6 0	62 12

Problema F
(Contando Múltiplos)

fonte: `contando.c`, `contando.cpp` *ou* `contando.java`

Descrição

Dada uma lista de k números primos e dois números inteiros m e n, quantos números inteiros positivos menores ou iguais a m são divisíveis exatamente por n dos números primos dados?

Entrada

A entrada será composta por vários casos de teste. A primeira linha de cada caso de teste contém os três números inteiros $2 \leq k \leq 20$, $1 \leq m \leq 100000000$ e $1 \leq n \leq k$. A segunda e última linha de cada caso de teste contém k números primos, $p_1, p_2, ..., p_k \leq 1000000$. A entrada termina com uma linha com $k = m = n = 0$, que não deve ser processada.

Saída

Para cada caso de teste, deve ser impressa a quantidade de inteiros positivos menores ou iguais a m divisíveis por exatamente 2 dos inteiros.

Exemplo

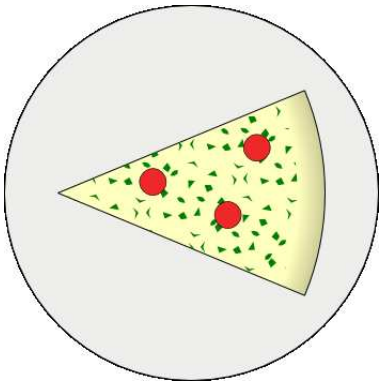
Entrada	**Saída**
5 10 2 2 3 5 7 13 5 30 2 2 3 5 7 13 0 0 0	2 11

Problema G
(Rodízio de Pizza)

fonte: `pizza.c`, `pizza.cpp` *ou* `pizza.java`

Descrição

O os alunos do curso comunitário de culinária "Experimentando Receitas Italianas" tiveram uma ideia para aproveitar os pratos feitos no curso: será feito um evento beneficente para crianças carentes. Para isso, serão feitas pizzas para serem servidas no evento. As pizzas serão cortadas em 8 fatias idênticas e servidas em pratos do próprio curso. Entretanto, as pizzas de diferentes alunos são de diferentes tamanhos. Além disso, o pratos do curso são oriundos de doações, tendo-se assim diversos modelos distintos. Por razões de higiene, uma fatia só pode ser servida em um prato se ela couber completamente naquele prato. Como a quantidade de pizzas será grande, você foi contratado para desenvolver um programa que, a partir de uma lista contendo raio de cada pizza e os raios dos pratos, diga qual a maior quantidade de pizzas que pode ser servida de maneira adequada.



Entrada

A entrada é composta por diversas instâncias do problema. Cada instância é composta por 3 linhas. A primeira linha contém 2 números inteiros, m e n, $1 \leq m \leq 1000000$, $1 \leq n \leq 1000000$ as quantidades de pizzas e pratos, respectivamente. A segunda linha contém m números inteiros, indicando os raios das pizzas. A terceira linha contém n números inteiros, contendo os tamanhos dos pratos. A entrada termina quando $m = n = 0$. Essa instância não deve ser processada.

Saída

Para cada instância de entrada deve ser impresso uma única linha contendo um inteiro referente à quantidade de fatias de pizza que podem ser servidas de maneira adequada.

Exemplo

Entrada	**Saída**
2 10 5 5 3 3 5 5 5 5 5 5 5 5 2 10 50 5 3 3 5 5 5 5 5 5 5 5 0 0	10 2

Problema H
(Código de Barras)

fonte: `barras.c`, `barras.cpp` *ou* `barras.java`

Descrição

Códigos de barra são largamente utilizados em diversos setores da indústria e comércio. Um dos códigos mais utilizados é o UPC, Universal Product Code, que possibilita o armazenamento de 12 dígitos decimais em cada código. Sua representação é feita utilizando um código binário e leva em consideração diversos fatores relacionados à confiabilidade do valor lido. Barras verticais representam bits 1 e espaços em branco representam bits 0.

Cada código de barras é composto por 5 partes. A primeira delas contém um código de controle 101. A segunda parte, chamada padrão esquerdo, codifica os 6 primeiros dígitos do valor representado. Cada um desses dígitos é representado por uma string de 7 bits. A seguir, há uma outra string de controle, 01010, que sinaliza o centro do código de barras. A quarta parte, denomidada padrão direito, codifica os outros 6 dígitos. Finalmente, a quinta e última parte contém também uma string de controle 101, como a string original. Cada número codificado resulta em uma string de $3 + 6 \times 7 + 5 + 6 \times 7 + 3 = 95$ bits. Os códigos de cada dígito decimal para o padrão esquerdo e o padrão direito são fornecidos na tabela a seguir.

Tabela 1: Padrões de codificação de dígitos.

Dígito	Padrão esquerdo	Padrão direito
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

Neste problema, dado um código de barras, você deverá decodificá-lo e fornecer o número representado ou informar que o código foi lido incorretamente. Observe que não há distinção entre início e fim do código, então seu programa deve ser capaz de decodificar o código de barras mesmo que este tenha sido lido de cabeça para baixo.

Entrada

A entrada é descrita por um inteiro positivo N na primeira linha, indicando o número de códigos de barra a serem decodificadas. Cada uma das N linhas seguintes contém uma string de 95 caracteres formada apenas pelos caracteres 0 e 1. Observação: no exemplo, há uma quebra de linha na string apenas para melhor visualização.

Saída

Para cada código de barra, deve ser impresso o número de 12 dígitos codificado. Caso haja algum erro na string, deve ser impressa a string XXXXXXXXXXXX.

Exemplo

Entrada	Saída
3 10101101110110001010111100010110100011011110101010 1 01110011001101011100111001010010001110100101 10101101110110001010111100010110100011011110100010 1 11110011001101111100111001000010001110100101 10100101110001001010011100111010110011001110101010 1 01111011000101101000111101010001101110110101	856943414089 XXXXXXXXXXXX 856943414089