Segunda Seletiva para

Maratona de Programação de 2008

Universidade de São Paulo

Caderno de Problemas

Departamento de Ciência da Computação IME-USP

Sábado, 6 de setembro de 2008.

Problema A: Número de Flechas

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

Número de Flechas é um quebra-cabeça. O objetivo é determinar as orientações das flechas posicionadas no perímetro de um tabuleiro quadrado em forma de grade. As pistas do quebra-cabeça são dadas como inteiros escritos dentro de cada quadrado do tabuleiro: cada inteiro indica quantas flechas apontam para aquele quadrado. Flechas podem apontar apenas para uma das oito direções da bússola e sempre devem apontar para pelo menos um quadrado do tabuleiro.

Escreva um programa que, dado um tabuleiro de Número de Flechas, encontra uma solução válida ou diz que não existe solução. A Figura 1 exibe um exemplo que possui solução.

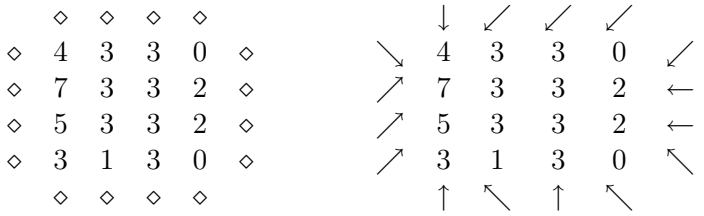

Figura 1: Um tabuleiro de Número de Flechas e sua solução.

Entrada

A entrada contém vários tabuleiros. A primeira linha da descrição de um tabuleiro contém um único inteiro N indicando o tamanho do tabuleiro. Cada uma das próximas N linhas contém N inteiros, separados por espaços, correspondendo aos números escritos nos quadrados do tabuleiro.

Há uma linha em branco após cada descrição de tabuleiro.

A entrada termina com uma linha contendo 0 ($N = 0$). Esta linha não deve ser processada.

A entrada deve ser lida da entrada padrão.

Restrições

$1 \leq N \leq 8$

Cada número do tabuleiro possui valor entre 0 e 8 (inclusive).

Saída

Para cada tabuleiro, a primeira linha da saída deve ser "Instancia #I:", onde I é o índice do tabuleiro na entrada, começando de 1.

Se o tabuleiro não tem solução, a próxima linha da saída deve ser `sem solucao`. Se o tabuleiro tem uma solução, as próximas $N + 2$ linhas da saída devem descrever a solução (todas as instâncias terão solução única). A saída deve ser formatada como uma grade $(N + 2) \times (N + 2)$. Cada célula da grade deve ter espaço para dois caracteres, alinhamento a direita e as células são separadas por espaços.

As $N \times N$ células centrais devem conter os números do tabuleiro original. As $4N$ células da borda devem conter as orientações das flechas, impressas como direções da bússola exibidas na Figura 2. Os quatro cantos devem conter apenas espaços em branco (em particular, a primeira e a última linha da saída devem conter espaços em branco no início e no final).

Há uma linha em branco após cada saída.

A saída deve ser escrita na saída padrão.

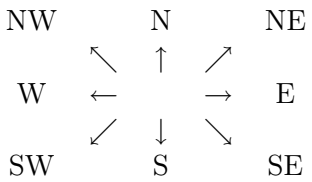

Figura 2: Direções da bússola.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 5 2 5 3 0 1 4 3 4 2 0 0 0 0 4 4 3 3 0 7 3 3 2 5 3 3 2 3 1 3 0 0	Instancia #1: S SE S E 5 2 5 W NE 3 0 1 NW E 4 3 4 W N NE NW Instancia #2: sem solucao Instancia #3: S SW SW SW SE 4 3 3 0 SW NE 7 3 3 2 W NE 5 3 3 2 W NE 3 1 3 0 NW N NW N NW

Problema B: Dando nome aos times

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

O grande *coach* de maratona Da Silva percebeu que um dos maiores problemas para os times que competem é a escolha do nome. Os times perdem muito tempo pensando em um bom nome e esquecem de treinar. Para resolver esse problema Da Silva resolveu que ele próprio escolheria os nomes dos times. No entanto, para evitar uma revolta ele pretende considerar (um pouco) a opinião de cada time.

Da Silva propôs N nomes para os N times e pediu que cada um fizesse uma lista de preferência, ordenando os nomes de acordo com suas preferências. É claro que para cada nome Da Silva já tinha sua lista de preferência dos times a receber o nome.

A escolha dos nomes será feita de forma que no final não exista um time t_1 com nome n_1 e um time t_2 com nome n_2, sendo que t_1 preferia o nome n_2 ao n_1 e Da Silva preferia dar o nome n_2 para t_1 ao invés de t_2. Caso exista mais de uma possibilidade satisfazendo esse critério, será escolhida a que melhor segue a lista de preferência de Da Silva.

Sendo um *coach* muito ocupado, com viagens pelo mundo, Da Silva quer que você escreva um programa para a escolha dos nomes dos times.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

A primeira linha de cada instância contém um inteiro N indicando o número de times. Os times são identificados por números de 1 a N.

Cada uma das próximas N linhas possui um nome de time proposto por Da Silva, um espaço e a lista de preferência de Da Silva para o nome. A lista de preferência para um nome é dada por uma permutação de $1, \ldots, N$ representando os times, com um espaço entre cada número, sendo o time mais a esquerda o de maior preferência.

Cada uma das N linhas seguintes possui a lista de preferência de cada time. A i-ésima dessas N linhas corresponde a lista do time i. A lista de preferência da cada time consiste nos N nomes de times ordenados de acordo com a preferência e separados por um espaço.

A entrada deve ser lida da entrada padrão.

Restrições

$1 \leq N \leq 300$

Um nome de time não tem mais que 15 caracteres e não contém espaços.

Saída

Para cada instância imprima N linhas, sendo que a i-ésima linha deve conter apenas o nome de time que Da Silva escolherá para o time i.

Imprima uma linha em branco após a saída de cada instância.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
2 2 cai_cai_balao 1 2 code_rupper 1 2 cai_cai_balao code_rupper code_rupper cai_cai_balao 3 2towers.bin 1 2 3 cai_cai_balao 2 3 1 code_rupper 3 1 2 cai_cai_balao code_rupper 2towers.bin code_rupper cai_cai_balao 2towers.bin 2towers.bin code_rupper cai_cai_balao	cai_cai_balao code_rupper 2towers.bin cai_cai_balao code_rupper

Comentários

Na segunda instância do exemplo existem 3 soluções possíveis:

$$M_1 = \{(1, \text{2towers.bin}), (2, \text{cai\_cai\_balao}), (3, \text{code\_rupper})\}$$

$$M_2 = \{(1, \text{cai\_cai\_balao}), (2, \text{code\_rupper}), (3, \text{2towers.bin})\}$$

$$M_3 = \{(1, \text{code\_rupper}), (2, \text{cai\_cai\_balao}), (3, \text{2towers.bin})\}$$

M_1 foi a escolhida, pois entre as 3 é a que melhor segue as preferências do *coach*.

Problema C: Sanduíche-íche-íche de atum

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

Os irmãos **I**vo **C**osta e **P**edro **C**osta estão sempre brigando. Tudo é motivo para esta dupla entrar em conflito. A última briga foi por causa de um sanduíche. Os dois estavam em uma festa de casamento e coincidentemente partiram em direção a um mesmo sanduíche de atum. Para tentar remediar a situação, a dona da festa repartiu o pão em duas metades e deu uma metade para cada um dos irmãos. O problema é que o Ivo sentiu-se prejudicado, pois alegou que a metade dada ao Pedro era muito mais recheada que a sua.

Pedimos que você e seus amigos calculem a quantidade de recheio contida no pedaço de sanduíche dado ao Pedro. Para isso, foi preciso que o *buffet* nos revelasse os segredos de uma de suas receitas mais cobiçadas, o sanduíche de atum: o pão usado é em forma de um quadrado e, portanto, pode ser dividido em $N \times N$ subquadrados de tamanhos iguais; em cada subquadrado é colocada uma determinada quantidade de recheio; o *gourmet* define uma sequência de valores $a_1, a_2, \ldots, a_N$ (dados em miligramas) e distribui o recheio no pão da seguinte forma

$$\begin{pmatrix} a_1a_1 & a_1a_2 & \cdots & a_1a_N \\ a_2a_1 & a_2a_2 & \cdots & a_2a_N \\ a_3a_1 & a_3a_2 & \cdots & a_3a_N \\ \vdots & \vdots & \ddots & \vdots \\ a_Na_1 & a_Na_2 & \ldots & a_Na_N \end{pmatrix}$$

sendo que a_ia_j corresponde a quantas miligramas de recheio serão colocadas no subquadrado de coordenadas i e j.

Da forma como o corte foi feito, no pão do Pedro tinha os recheios correspondentes aos subquadrados $a_1a_1, a_1a_2, \ldots, a_1a_N$, mais os recheios correspondentes aos subquadrados $a_2a_2, a_2a_3, \ldots, a_2a_N$, mais $a_3a_3, a_3a_4, \ldots, a_3a_N$ e assim por diante; ou seja, Pedro recebeu os recheios correspondentes a parte triangular superior do pão.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

Cada instância é composta de duas linhas. A primeira linha contém um inteiro N indicando o tamanho da sequência. A segunda linha contém N inteiros $a_1, a_2, \ldots, a_N$ separados por um espaço.

A entrada deve ser lida da entrada padrão.

Restrições

$$\begin{array}{rcccl} 1 & \leq & N & \leq & 100\,000 \\ 0 & \leq & a_i & \leq & 100\,000\,000 \quad \text{para } i = 1, 2, \ldots, N \end{array}$$

Saída

Para cada instância, imprima a quantidade de recheio que tem no pão do Pedro. Tal valor deve ser impresso módulo 1300031.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
2 4 1 2 3 4 3 666 666 666	65 61274

Problema D: Calculadora bugada!

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

A **A**bsolute **C**o**M**puters (ACM) é uma grande companhia da área de informática. Criando tanto aplicativos de larga escala, como sistemas gerenciadores de banco de dados, quanto sistemas de controle de estoque, a empresa vinha crescendo a cada ano em percentuais impressionantes. Isso até seu último lançamento, quando prometeram algo totalmente inovador: a possibilidade do usuário de ter uma calculadora no seu próprio computador. "Não há nada de novo nisso!", você pode pensar. Mas se o fez é só porque não me deixou terminar de explicar tudo. Senta aí e termina de ler tudo primeiro!

Uma das novas *features* dessa calculadora é que todas as operações podem ser definidas através de um *script*. Ou seja, você pode criar desde simples somatórias até complicadas equações diferenciais que a calculadora fará as contas e lhe dará o resultado correto. Para demonstrar o poder da linguagem de *script* a ACM fez uma apresentação em um evento internacional. Nessa apresentação iria exibir exemplos de utilizações avançadas da calculadora ao público.

Inicialmente o encarregado para o desenvolvimento dos complexos *scripts* seria o diretor de projeto. Porém ele resolveu tirar férias algumas semanas antes do evento, e passou a tarefa para o chefe de desenvolvimento. Esse, muito ocupado com seus jogos de golfe, repassou para o desenvolvedor sênior. Como o desenvolvedor sênior estava com muitos bugs (em outros projetos) para corrigir, repassou para o programador júnior. Já o programador júnior achou que aquilo era só mais um trote, e resolveu aproveitar a "piada" para aprontar com o estagiário. Aí o estagiário fez o que sabe de melhor (vocês já sabem o quê). Resultado: o que era para ser um script complexo para demonstrar o poderio do software tornou-se em um simples exemplo de potenciação. E pior, com três falhas grotescas:

1. o único número aceito como base da potência era 1331
2. a base numérica era 1300031
3. e ele esqueceu totalmente do "vai um". Segue abaixo um exemplo de multiplicação (na base 16) com esse bug.

$$\begin{array}{cccc} & & 4 & 5 \\ \times & & 3 & 1 \\ \hline & & 4 & 5 \\ + & 12 & 15 & \\ \hline & 12 & 3 & 5 \end{array}$$

Mas nem tudo foi perdido. Meu professor adorou esse novo tipo de operação e passou como tarefa simulá-la. Como não tenho um estagiário, vai ter que ser você mesmo para me ajudar.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

Cada instância terá uma linha com dois inteiros N e P.

A entrada deve ser lida da entrada padrão.

Restrições

$$\begin{array}{ccccc} 1 & \leq & N & \leq & 10\,000 \\ 0 & \leq & P & \leq & 100 \end{array}$$

Saída

Para cada instância, imprima o valor, computado pelo script do estagiário, do P-ésimo dígito (começando em 0 — da direita para esquerda) de 1331 elevado a N.

A saída deve ser escrita na saída padrão.

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

Comentários

A culpa não é do estagiário! A culpa é do Carlinhos! :–)

Problema E: Setun

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

Setun era um computador ternário balanceado, desenvolvido em 1958 na Universidade de Moscou. Ele foi um dos primeiros computadores ternários, usando lógica de três valores em vez da mais comum lógica de dois valores.

Historiadores russos querem construir um emulador de Setun para rodar os programas escritos para tal computador. No entanto, eles precisam de um programa que converta os dados para um formato que o Setun entenda e pediram a sua ajuda.

Escreva um programa que, dado um inteiro, imprime a representação desse inteiro em ternário balanceado.

Números em ternário balanceado são escritos usando dígitos ternários, ou *trits*. Os três *trits* possíveis são +, 0 e −. Como no sistema numérico decimal, o *trit* mais a direita tem valor 1, mas, diferentemente do sistema numérico decimal, mover um *trit* uma posição a esquerda multiplica seu valor por 3.

O valor de um número ternário balanceado é obtido multiplicando o valor da posição de cada *trit* pelo seu sinal. Por exemplo, $++0- = 27 + 9 - 1 = 35$ e $-+0 = -9 + 3 = -6$.

Entrada

Cada linha da entrada contém um inteiro N.

A entrada termina com *end-of-file* (EOF).

A entrada deve ser lida da entrada padrão.

Restrições

$-999\,999\,999 \leq N \leq 999\,999\,999$

Saída

Para cada instância imprima a representação em ternário balanceado de N em uma única linha.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
35 -6	++0- -+0

Problema F: Corrida para o ICPC

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

O prefeito, cujo apelido é "Guima", de uma pequena cidade do interior de São Paulo decidiu organizar uma corrida para decidir quais cidadãos terão direito às vagas remanescentes no ICPC (Incontro Cemanal das Pessoa Carentes), evento no qual são distribuídas cestas básicas, entre outras coisas, aos participantes. Guima designou você e sua equipe para definir qual será o trajeto da corrida.

Na cidadezinha existem exatamente M ruas e N pontos turísticos. Cada uma das M ruas liga um par de pontos turísticos distintos. Todas as ruas são de mão dupla. Tudo muito simples!

Para dificultar a sua vida (ele adora fazer isso), Guima fez algumas exigências com relação ao trajeto. A quantidade de pontos turísticos distintos contidos no trajeto deve ser ímpar (no mínimo três). O trajeto deve ser fechado (começar e terminar no mesmo lugar), para que os espectadores assistam à largada e à chegada da corrida. Para não tornar enfadonha a jornada dos aspirantes a atletas, o trajeto não pode passar mais do que uma vez por um mesmo ponto turístico (com exceção do ponto inicial que coincide com o ponto final). O comprimento do trajeto deve ser o menor possível, afinal se trata de uma corrida de amadores. O comprimento de um trajeto é a soma dos comprimentos das ruas que os atletas terão que percorrer.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

Cada instância é composta de diversas linhas. A primeira linha contém dois inteiros positivos N e M, sendo que N corresponde ao número de pontos turísticos da cidade, e M corresponde ao número de ruas. Os pontos da cidade são identificados por $1, 2, \ldots, N$. Cada uma das próximas M linhas contém uma tripla de inteiros positivos separados por um espaço, digamos u, v e c, indicando que a rua que liga os pontos turísticos u e v tem comprimento c.

A entrada deve ser lida da entrada padrão.

Restrições

$$
\begin{array}{rcrcl}
1 & \leq & N & \leq & 100 \\
1 & \leq & M & \leq & \frac{N^2-N}{2} \\
1 & \leq & u, v & \leq & N \\
1 & \leq & c & \leq & 999
\end{array}
$$

Saída

Para cada instância imprima um inteiro contendo o comprimento do trajeto, conforme especificado; ou a palavra `impossivel`, caso não exista um trajeto que atenda aos requisitos **impostos** pelo pseudodemograta Guima.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
2 5 10 1 2 1 2 3 1 3 4 1 4 5 1 5 1 1 1 4 4 1 3 4 2 5 4 2 4 4 3 5 4 4 4 1 2 666 3 4 666 1 3 666 2 4 666	5 impossivel

Problema G: O General

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

A Segunda Guerra Mundial foi o confronto que causou mais vítimas na história da humanidade. Iniciando-se em 1939, ela opôs, de um lado, o Eixo, formado por Alemanha, Itália e Japão, e de outro os Aliados, que contavam, entre outros, com China, França, Inglaterra, Estados Unidos e Rússia. O Brasil se integrou aos Aliados em 1943, grupo esse que saiu vencedor em 1945.

Entre os principais nomes da guerra estavam Adolph Hitler, Benito Mussolini, Joseph Stalin e Winston Churchill. Porém, outras personalidades também tiveram seu destaque no período. Alguns desses passaram desconhecidos por várias décadas, mas documentos recentemente encontrados, mostraram que a atuação dessas pessoas tiveram um grande impacto no resultado final. Um desses ex-desconhecidos é o General da Gringolândia, Charles Eduard Blacksmith.

Charles era uma pessoa muito peculiar. Ele exigia que seus soldados sempre formassem uma fila, de tal maneira que pudesse vê-los todos ao mesmo tempo, para receber as ordens. Por ter uma estatura abaixo da média (usando um pouco de eufemismo), essa fila deveria estar em ordem crescente de altura dos soldados. Para evitar confusão, Little Charles, como era conhecido, sempre comandava seus soldados na hora deles se organizarem na fila. Porém, logo ele notou que para grandes batalhões essa era uma tarefa que dava muito trabalho. Para evitar a fadiga, ele, então, decidiu contratar terceiros para organizar a fila no seu lugar. Sua tarefa é, dada a fila inicial, com a altura de cada soldado, encontrar o menor número de trocas para organizá-la. Contudo, Charles impôs a seguinte condição: um soldado na posição i da fila só pode trocar de lugar com o soldado que está na posição $i + k$ da fila. (um jeito bem peculiar, não?)

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

Cada instância terá duas linhas. Na primeira haverá dois inteiros N e K que indicam o número de soldados e a distância entre dois soldados que podem trocar de posições, respectivamente. Na segunda haverão N inteiros separados por um espaço, com a altura de cada soldado, na ordem em que estão na fila inicialmente.

A entrada deve ser lida da entrada padrão.

Restrições

$1 \leq N \leq 100\,000$
$1 \leq K \leq N$
$1 \leq a_i \leq 1\,000\,000$ Altura do soldado i, onde $i = 1, 2, \ldots, N$.

Saída

Para cada instância imprima um inteiro com o menor número de trocas necessárias para ordenar os soldados, ou `impossivel` caso não seja possível ordenar os soldados levando em conta a condição imposta por Little Charles.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
3 4 2 3 4 1 2 4 1 4 3 2 1 4 2 7 3 6 5	2 6 impossivel

Problema H: Atirador de elite

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

Problemas e mais problemas! Assim foi a Seletiva para Maratona de Programação dos Alunos do IME-USP. Enquanto o big-boss "Guima" conquistava o Egito, seus seguidores passavam muito sufoco e angústia durante a prova. Assim que retornou, Guima conversou com os alunos e as principais reclamações foram: a demora nas respostas e os estouros repentinos dos balões. Muitos alunos eram bichos (alunos do primeiro ano ... *hehe*) e estavam com muito medo de ter perdido o problema. Eles realmente achavam que era preciso fazer tudo do zero para ganhar um novo balão.

Imediatamente, após a reunião, o grande Faraó Guima pediu as fitas de segurança do laboratório, e constatou que havia um atirador de elite no canto da sala. Em intervalos de tempo o atirador disparava um tiro contra os balões. Após assistir ao vídeo inteiro, Guima percebeu que o atirador sempre dava um tiro que estourava a maior quantidade de balões possível.

Agora, o grande Cacique Guima quer que você e sua equipe escrevam um programa que dadas N coordenadas (x, y), determine o número de balões que o atirador consegue acertar com um único tiro. Um fato importante é que as balas do atirador não fazem curva.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

Cada instância é composta de diversas linhas. A primeira linha de cada instância contém um inteiro N — o número de balões. Seguido por N linhas que descrevem N coordenadas de balões. Cada coordenada de balão é dada por dois inteiros x e y separados por um espaço.

A entrada deve ser lida da entrada padrão.

Restrições

$$\begin{array}{rcccl} 1 & \leq & N & \leq & 1\,000 \\ -100\,000 & \leq & x, y & \leq & 100\,000 \end{array}$$

Dois balões quaisquer não possuem a mesma coordenada.

Saída

Para cada instância imprima o número de balões que o atirador consegue acertar com um tiro.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
2 9 0 0 1 1 1 2 1 3 2 1 2 2 2 3 3 2 3 3 5 1 1 2 1 3 1 4 1 3 2	4 4

Problema I: Diagrama de Voronoi

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

O diagrama de Voronoi é muito famoso na matemática e computação. Nomeado em homenagem a Georgi Voronoi, seu objetivo é, dado um conjunto S de pontos no plano, determinar, para cada ponto p, a região $V(p)$ onde todos os pontos dali estão mais próximos de p do que de qualquer outro ponto de S.

É muito fácil encontrar aplicações para o diagrama. Suponha, por exemplo, que você tenha o mapa de uma cidade, e a localização de todos os cyber-cafés. Assim, toda vez que você estiver em algum lugar da cidade, e precisar verificar se algum amigo mandou um e-mail novo, pode facilmente achar o mais próximo. Outra possibilidade seria se em vez de cyber-cafés tivéssemos hospitais ou postos de saúdes, mas claramente o primeiro exemplo é mais importante.

É possível estender o conceito do diagrama para mais dimensões. Estamos, aqui, interessados no caso tri-dimensional. Como a implementação é trivial, os juízes fizeram as deles, enquanto jogavam uma partidinha de War (que o exército azul ganhou). Para o término do programa faltou apenas uma função que eles não entraram em acordo. É sobre calcular o volume de um sólido formado por quatro pontos A, B, C e D. Um dos juízes, disse que a fórmula é Área da Base $\times$ Altura/3. O outro falou que é Altura $\times$ Área da Base/3. Sua tarefa é fazer um programa que leia os pontos A, B, C e D e imprima o volume do sólido formado por eles.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias.

Cada instância terá quatro linhas. Cada linha conterá as coordenadas X, Y e Z dos pontos A, B, C e D, respectivamente. As coordenadas são separadas por um espaço.

A entrada deve ser lida da entrada padrão.

Restrições

$$\begin{array}{ccccccc} 1 & \leq & X, Y, Z & \leq & 1\,000 & & \\ A & \neq & B & \neq & C & \neq & D \end{array}$$

Saída

Para cada instância, imprima, com precisão de até 6 casas decimais, o volume do sólido.

A saída deve ser escrita na saída padrão.

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

`1`
`0 0 0`
`10 10 0`
`20 0 0`
`10 5 10`
	`333.333333`

Comentários

Aviso: Na maratona você não tem acesso à internet. Logo, não estrague a brincadeira pesquisando a fórmula na internet.