

# Maratona de Programação FACENS 2017

## CADERNO DE PROBLEMAS

### Informações gerais

Este caderno contém 9 problemas; as páginas estão numeradas de 1 a 18, não contando esta folha de rosto.

Verifique se o caderno está completo.

#### A) Sobre os nomes dos programas

1) Sua solução deve ser chamada *codigo\_de\_problema.c*, *codigo\_de\_problema.cpp* ou *codigo\_de\_problema.java*, onde *codigo\_de\_problema* é especificado em cada problema. Lembre que em Java o nome da classe principal deve ser igual ao nome do arquivo.

#### B) Sobre a entrada

- 1) A entrada de seu programa deve ser lida da entrada padrão.
- 2) A entrada pode ser composta de um ou mais casos de teste, descrito em um número de linhas que depende do problema.
- 3) Quando uma linha da entrada contém vários valores, estes são separados por um único espaço em branco; a entrada não contém nenhum outro espaço em branco.
- 4) Cada linha, incluindo a última, contém exatamente um caractere final-de-linha.

#### C) Sobre a saída

- 1) A saída de seu programa deve ser escrita na saída padrão.
- 2) Quando uma linha da saída contém vários valores, estes devem ser separados por um único espaço em branco; a saída não deve conter nenhum outro espaço em branco.
- 3) Cada linha, incluindo a última, deve conter exatamente um caractere final-de-linha.



## PROBLEMA A: MULTIPLICAÇÃO DE CADEIA DE MATRIZES

**Arquivo:** mult.[cpp/c/java]

**Cor:** branco

**Limite de tempo:** 1s

### Descrição do problema

Multiplicação de cadeia de matrizes (ou problema da ordenação de cadeia de matrizes) é um problema de otimização. Dada uma sequência de matrizes, o objetivo é encontrar a forma mais eficiente de multiplicar essas matrizes. O problema não é na realidade para realizar a multiplicação, mas apenas para decidir a sequência de multiplicações das matrizes envolvidas.

Temos muitas opções, pois a multiplicação de matrizes é associativa. Em outras palavras, não importa como por entre parênteses o produto, o resultado obtido será o mesmo. Por exemplo, se tivéssemos quatro matrizes A, B, C, e D, teríamos:

$$((AB)C)D = (A(BC))D = (AB)(CD) = A(BC)D = A(B(CD)).$$

No entanto, a ordem em que se põe entre parênteses o produto afeta o número de operações aritméticas simples necessárias para calcular o produto, ou a eficiência. Por exemplo, suponha que A é uma matriz  $10 \times 30$ , B é uma matriz  $30 \times 5$ , e C é uma matriz  $5 \times 60$ . Então,

$$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ operações}$$

$$A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ operações.}$$

Aqui, foi assumido que a quantidade de operações para multiplicar uma matriz  $P \times Q$  por uma matriz  $Q \times R$  é  $P \times Q \times R$ .

O que realmente quero saber é o custo mínimo, ou o número mínimo de operações aritméticas, necessário para multiplicar as matrizes. Se estamos apenas fazendo a multiplicação de duas matrizes, só há uma maneira de multiplicar, de modo que o mínimo custo é o custo de fazer isso.

### Entrada

A entrada é composta de vários casos de testes. Cada caso de teste é composto por várias linhas. A primeira linha de um caso de teste é um inteiro N que representa a quantidade de matrizes multiplicadas. Cada uma das próximas N linhas de um caso de teste possui dois inteiros P e Q que são a quantidade de linhas e colunas da matriz, respectivamente.

Vale ressaltar que a quantidade de linhas de uma matriz é igual à quantidade de colunas da matriz anterior; a única exceção sendo a primeira matriz por não possuir matriz anterior a ela.

A entrada termina quando  $N = 0$ ; este caso não deve ser processado.

### Saída

A saída deve conter uma linha para cada caso de teste. Cada linha de saída de cada caso de teste deve conter um número T que é o menor número de operações necessárias para realizar a multiplicação das N matrizes.

**Exemplos de testes**

Entrada	Saída
2	24
2 3	12
3 4	64
3	4500
1 1	
1 5	
5 2	
3	
2 3	
3 4	
4 5	
3	
10 30	
30 5	
5 60	
0	

**Restrições**

$$1 \leq N \leq 100$$

$$1 \leq P, Q \leq 10.000$$

## PROBLEMA B: GUERRA FRIA

**Arquivo:** guerra.[cpp/c/java]

**Cor:** marrom

**Limite de tempo:** 1s

### Descrição do problema

Durante a Guerra Fria a troca de mensagens era realizada de uma forma “especial” a fim de evitar que tropas inimigas interceptassem a mesma e descobrissem segredos e estratégias importantes.

Essa forma “especial” de se enviar as mensagens foi definida pela Enigma a qual se utilizava da combinação de sistemas elétricos e mecânicos para criptografar as mensagens e consequentemente deixar elas secretas.

O processo utilizado era bastante peculiar, onde eram utilizadas somente letras (todas maiúsculas) e eram aplicadas sobre elas uma rotação (código) que era responsável por “criptografar” a mensagem original. Com isso, caso a mensagem fosse interceptada a mesma estaria totalmente ilegível para quem não tivesse conhecimento do código utilizado no processo.

Para dificultar a leitura no processo os espaços em branco foram substituídos por # (hashtags). Além disso, quando as seguintes letras existissem na palavra original elas deveriam ser substituídas por:

- A por @ (arroba);
- S por \$ (cifrão);
- E por 3 (número três);
- I por ! (exclamação);
- O por 0 (número zero).

### Entrada

A entrada é composta por vários casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha de um caso de teste contém um inteiro N que representa a rotação que será realizada. A segunda linha de um caso de teste contém a frase S que será criptografada; todas as letras de S são maiúsculas.

A entrada termina quando  $N = 0$ ; este caso não deve ser processado.

### Saída

A saída deve conter uma linha para cada caso de teste. A linha da saída de um caso de teste deve conter a frase criptografada. Caso seja informada uma rotação fora do intervalo indicado ou a palavra exceder o tamanho estipulado deve ser exibido “ERROR” como resposta.

**Exemplos de testes**

Entrada	Saída
3 FACENS 35 MENSAGEM 1 COMPUTACAO 9 ENGENHARIA 13 NAO TEM MUNDIAL 0	I@F3Q\$ ERRO D0NQVU@D@0 3WP3WQ@A!@ A@0#G3Z#ZHAQ!@Y

**Restrições**

$$1 \leq N \leq 20$$

$$1 \leq |S| \leq 51 \text{ (tamanho de S)}$$

## PROBLEMA C: TRIÂNGULO DE PASCAL

**Arquivo:** `pascal.[cpp/c/java]`

**Cor:** verde claro

**Limite de tempo:** 1s

### Descrição do problema

O Triângulo de Pascal usa uma combinação numérica para a sua formação. O mesmo fornece os coeficientes da expansão binominal, combinação usada em probabilidade em outras séries numéricas. Uma expansão muito utilizada é o produto notável.

Abaixo temos um exemplo de triângulo de pascal calculado até a sétima linha.

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

```

Onde o mesmo pode ser calculado para qualquer quantidade de linhas.

Na terceira linha do triângulo de pascal temos os termos usados para o produto notável quadrático:

$$(a + b)^2 = 1(a^2b^0) + 2(a^1b^1) + 1(a^0b^2) = a^2 + 2ab + b^2$$

A sua tarefa é criar um programa que, dado a quantidade de linhas, determine seu triângulo de pascal com o respectivo polinômio ao lado.

### Entrada

A entrada é constituída de vários casos de teste. Cada caso possui um valor N que representa a quantidade linhas do Triângulo de Pascal.

A entrada termina quando N = -1; este caso não deve ser processado.

### Saída

A saída deve conter várias linhas para cada caso de teste. A saída de um caso de teste deve ser o Triângulo de Pascal com N linhas, cada linha deve conter também o respectivo polinômio ao lado. Os valores do Triângulo de Pascal deve ser separados por espaço seguindo por um espaço, uma seta =>, outro espaço e o polinômio obtido por aquela linha. O polinômio deve ser representado por uma somatório de produtos. Cada produto deve conter as letras maiúsculas **A** e **B**, seus respectivos expoentes e o multiplicador. As somas dos produtos devem ser precedidas e sucedidas por espaços. Seguir exemplo de teste.

### Exemplos de Testes

Entrada	Saída
3	1 => 1* (A^0*B^0)
2	1 1 => 1* (A^1*B^0) + 1* (A^0*B^1)
-1	1 2 1 => 1* (A^2*B^0) + 2* (A^1*B^1) + 1* (A^0*B^2)
	1 => 1* (A^0*B^0)
	1 1 => 1* (A^1*B^0) + 1* (A^0*B^1)

### Restrições

$$1 \leq N \leq 15$$



## PROBLEMA D: LOUSA DIGITAL

**Arquivo:** lousa.[cpp/c/java]

**Cor:** azul escuro

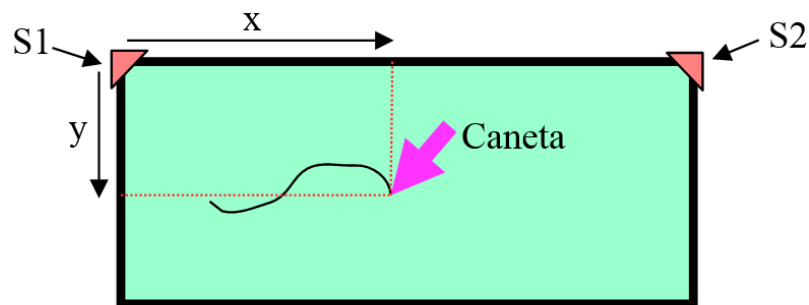
**Limite de tempo:** 1s

### Descrição do problema

Lousas Digitais são lousas em que pode-se escrever com uma caneta virtual e o traço é reproduzido na projeção, e pode ser armazenado em arquivo.

Você, um engenheiro de uma empresa de tecnologia, vai participar da equipe de desenvolvimento de uma nova Lousa Digital. A ideia dessa lousa é ser adaptada em qualquer superfície retangular em uma parede, perpendicular ao chão, onde são acoplados dois sensores de distância, um em cada canto superior do espaço retangular. Esse sensores captarão a distância da caneta. Sua tarefa é desenvolver o algoritmo que, recebendo a informação de altura e largura da superfície e das distâncias lidas pelos sensores S1 e S2, retorne as coordenadas x e y da posição da caneta na superfície. Todos os valores estarão em milímetros. Os valores de x crescem para a direita e os valores de y crescem para baixo.

Exemplo:



### Entrada

A entrada é constituída de vários casos de teste. Cada caso de teste possui 4 valores inteiros A, B, C e D. Os 4 valores são, respectivamente à ordem, a largura da lousa, a altura da lousa, a distância detectada por S1, a distância detectada por S2, todos em milímetros.

A entrada termina quando  $A = B = C = D = 0$ ; este caso não deve ser processado.

### Saída

Para cada caso de teste, a saída deve ter 2 valores inteiros em uma única linha que representam, na ordem, x e y, em milímetros. Os valores de x e y devem ser arredondados para o inteiro mais próximo. Caso os sensores indiquem um valor impossível, a saída deve ser "DEFEITO". Por exemplo, a posição da caneta deve ser dentro da área da lousa.

**Exemplos de testes**

<b>Entrada</b>	<b>Saída</b>
2000 1000 500 1500 1000 800 750 1250 2000 1000 500 500 0 0 0 0	500 0 0 750 DEFEITO

**Restrições** $0 \leq A, B, C, D \leq 10.000$

## PROBLEMA E: COORDENADA GEODÉSICA

**Arquivo:** coordenada.[cpp/c/java]

**Cor:** rosa

**Limite de tempo:** 1s

### Descrição do problema

A empresa OXGPS está contratando pessoas com conhecimentos em coordenadas geodésicas para trabalhar em um novo produto de seu catálogo. O produto consiste no rastreamento de veículos através de sua posição latitude e longitude.

Ao observar como os dados são transmitidos, constataram que a coordenada recebida pelo software está em graus, minuto e segundos, isso dificulta os cálculos que serão atendidos pelo produto.

Como parte da entrevista foi considerado o seguinte teste:

Crie um programa capaz de transformar a coordenada geodésica de graus para decimal.

Dica 1: 1° equivale a 1 hora

Dica 2: Para não existir problemas de formato a coordenada 23°27'07"S 47°27'06"W será recebida como 23g27m07sS 47g27m06sW

### Entrada

A entrada possui vários casos de teste. Cada caso de teste possui várias linhas. A primeira linha de um caso de teste possui um inteiro N que representa a quantidade de coordenadas a serem transformadas. Cada uma das próximas N linhas de um caso de teste possui dois textos que representam as coordenadas geodésicas (Latitude, Longitude) separados por espaços.

A entrada termina quando N = 0; este caso não deve ser processado.

### Saída

A saída é composta pelas coordenadas latitude e longitude em decimal separadas por espaço. O separador de decimal deve ser o ponto. Utilize 2 casas decimais, efetue o truncamento da última casa decimal, não arredonde.

### Exemplos de testes

Entrada	Saída
2	-23.45 -47.45
23g27m07sS 47g27m06sW	-23.52 -47.14
23g31m25sS 47g08m39sW	42.34 -71.06
3	34.92 104.14
42g20m55sN 71g03m38sW	29.76 -95.36
34g55m32sN 104g08m30sE	
29g45m39sN 95g21m36sW	
0	

### Restrições

$$1 \leq N \leq 100$$



**PROBLEMA F: RESPOSTA PARA VIDA, O UNIVERSO E TUDO MAIS**

**Arquivo:** resposta.[cpp/c/java]

**Cor:** amarelo

**Limite de tempo:** 1s

**Descrição do problema**

O Pensador Profundo é um computador projetado para calcular a resposta para vida, o universo e tudo mais. Após 7,5 milhões de anos, o pensador profundo calcula a resposta da vida, do universo e tudo mais: 42.

Indignados, os seres hiperinteligentes que o construíram dizem que essa não era a resposta da pergunta. O computador retruca que eles próprios não sabiam qual era pergunta, afirmando em seguida que a resposta em si (42), só poderia ser entendida quando soubessem a pergunta.

O Pensador Profundo dá aos seres hiperinteligentes cálculos de como fazer o único computador capaz de encontrar a Pergunta para a resposta 42. Esse computador seria tão grande e fenomenal que seria até confundido como um planeta. E a esse computador chamariam de Terra, a qual levaria 10 milhões de anos para completar seus objetivos.

A sua tarefa é criar um programa que faz o mesmo que o Pensador Profundo: calcular a resposta para vida, o universo e tudo mais. Logicamente, ele não pode demorar os mesmos 7,5 milhões de anos; a maratona não dura tanto tempo assim.

**Entrada**

A entrada é composta de uma única linha com o texto: "resposta para vida, o universo e tudo mais".

**Saída**

A saída deve ser composta uma linha com a resposta para vida, o universo e tudo mais.

**Exemplos de testes**

Entrada	Saída
resposta para vida, o universo e tudo mais	42



## PROBLEMA G: AVIÃO

**Arquivo:** aviao.[cpp/c/java]

**Cor:** laranja

**Limite de tempo:** 1s

### Descrição do problema

Alanito é um fiscal de aviação e precisa por várias vezes realizar reuniões e auditorias em diversos aeroportos pelo mundo para determinar se as normas de segurança estão sendo cumpridas. Alanito sabe que toda escala deve ter duração de 3 horas, e que um avião comercial viaja a uma velocidade de 1000 quilômetros por hora, e uma nova norma internacional da aviação determina que um avião não deve realizar um voo de mais de 12 horas contínuas, ou seja sem escalas. Alanito sabe ainda que os tempos de ida e volta dos voos devem ser os mesmos. Ele pediu sua ajuda para fazer um programa capaz de verificar se os aeroportos e companhias aéreas estão dentro da norma, cumprindo os tempos esperados para os voos. Sua tarefa é ajudá-lo nessa missão.

### Entrada

A entrada contém apenas um caso de teste que contém várias linhas. A primeira linha possui um inteiro A que representa a quantidade de aeroportos. Cada uma das próximas A linhas representa um aeroporto e possui um texto de 3 caracteres IATA e um inteiro F que representam o código IATA e o fuso horário no qual o aeroporto se localiza, respectivamente.

A próxima linha possui um inteiro V que representa a quantidade de vôos entre aeroportos. Cada uma das próximas V linhas contém dois textos de 3 caracteres IATA1 e IATA2 e um inteiro D que representam o código do aeroporto de origem, o código do aeroporto de destino e a distância em quilômetros entre eles, respectivamente.

Na sequência, há uma linha com um inteiro R que representa a quantidade de rotas que devem ser verificadas. Cada uma das próximas R linhas contém dois textos de 3 caracteres IATA3 e IATA4, um inteiro S e um inteiro C que representam o código do aeroporto de origem, o código do aeroporto destino, o horário de saída do aeroporto de origem e a hora de chegada no aeroporto destino, respectivamente. OBS: Uma hora de destino C maior que 24 indica que o voo chegará no dia seguinte.

### Saída

A saída deve conter R linhas. Cada linha deve indicar, para cada rota especificada na entrada, se é possível realizar a rota no tempo estabelecido e seguindo as normas da aviação, sendo os valores possíveis SIM ou NAO (sem acentos).

**Exemplos de testes**

Entrada	Saída
5	SIM
SFO -7	NAO
JFK -4	NAO
SCL -4	NAO
GRU -3	SIM
SYD 10	
7	
SFO SCL 9500	
GRU SCL 2600	
SYD JFK 16000	
JFK GRU 7600	
SFO JFK 4100	
SCL SYD 11300	
GRU SYD 13300	
5	
GRU SFO 9 20	
GRU SFO 9 19	
JFK SYD 1 31	
JFK SYD 1 38	
SYD SCL 10 26	

**Restrições**

$$3 \leq A \leq 20$$

$$-12 \leq F \leq 12$$

$$3 \leq V \leq 100$$

$$100 \leq D \leq 20.000$$

$$1 \leq R \leq 50$$

$$0 \leq S \leq 23$$

$$0 \leq C \leq 48$$

IATA, IATA1, IATA2, IATA3 e IATA3 são textos compostos por letras maiúsculas de A a Z



## PROBLEMA H: CAMINHO MAIS RÁPIDO

**Arquivo:** caminho. [cpp/c/java]

**Cor:** preto

**Limite de tempo:** 1s

### Descrição do problema

Atualmente muitas pessoas utilizam aplicativos de navegação baseados em GPS ao dirigir. Além de serem úteis para caminhos que não conhecemos, podem ajudar a achar alternativas para caminhos que já conhecemos. Aplicativos baseados em redes colaborativas compartilham informações entre os usuários que permitem estimar as condições de tráfego. Assim, cada via pode ter seu tempo de travessia estimado, e o melhor caminho encontrado, baseado no menor tempo total entre os caminhos alternativos. Desta forma, a partir de um mapa de caminhos alternativos, você deve determinar o menor tempo possível em que é possível chegar de uma origem a um destino.

### Entrada

O mapa dos caminhos alternativos pode ser definido como um conjunto de pontos, sendo as vias definidas como a ligação entre dois pontos. Como dados de entrada você receberá um conjunto de sequências de números inteiros. O primeiro número (N) de cada conjunto indicará quantidade de pontos do mapa. Em seguida, você receberá uma sequência de valores inteiros que preenchem uma tabela N x N. O valor do elemento E<sub>ij</sub> desta tabela representa o tempo estimado de percurso na via que liga os pontos i e j. Note que o valor do elemento E<sub>ij</sub> pode ser diferente do valor do elemento E<sub>ji</sub> pois representam sentidos diferentes da mesma via. Os valores preencherão a tabela por linhas, ou seja, E<sub>11</sub>, E<sub>12</sub>, E<sub>13</sub>, ..., E<sub>21</sub>, E<sub>22</sub>, ... O valor de um elemento i=j sempre será nulo, pois representa o tempo entre um ponto e ele mesmo. Se não houver uma via que conecte os pontos "i" e "j" diretamente, o elemento E<sub>ij</sub> será fornecido "0" (zero).

A entrada termina quando N = 0; este caso não deve ser processado.

### Saída

A saída de seu programa deve ser um número inteiro que representa o tempo mínimo gasto entre o primeiro e último ponto. Caso não exista um caminho possível entre o primeiro e último ponto você deve imprimir "CAMINHO INEXISTENTE". (em letras maiúsculas).

### Exemplos de testes

Entrada	Saída
4 0 1 0 0 0 0 1 0 0 0 0 1 2 0 0 0	3
4 0 3 0 0 0 0 0 0 0 1 0 3 3 3 0 0	CAMINHO INEXISTENTE
4 5 5 1 5 5 5 5 1 5 1 5 5 5 5 5 5	3
0	

### Restrições

$$1 \leq N \leq 100$$

$$0 \leq E_{ij} \leq 120$$



## PROBLEMA I: GUERRA INFINITA

**Arquivo:** infinity.[cpp/c/java]

**Cor:** vermelho

**Limite de tempo:** 1s

### Descrição do problema

Uma das sagas mais marcantes do universo da Marvel foi a Guerra Infinita. Nela, o titã louco Thanos forjou a Manopla do Infinito com as seis joias do infinito: espaço, mente, alma, realidade, tempo e poder. O mesmo evento está sendo planejado para o cinema, entretanto, o enredo está um pouco diferente dos quadrinhos. Sendo assim, Thanos terá que obter as joias de diferentes personagens. Para tentar manter uma certa lógica, a Marvel Studios contratou sua equipe para criar um programa para ajudar a manter o enredo razoavelmente consistente.

Cada joia possui características particulares que, na mão de Thanos, podem ser utilizadas para derrotar mais facilmente determinados heróis. Sua equipe deve retornar para Thanos a sequência de joias que ele deve obter, baseando-se nas características dos heróis ou grupos que estão guardando as joias. Os heróis/grupos guardiães das joias serão descritos utilizando 4 atributos: força, destreza, inteligência e espírito, com valores entre 0 e 1000. As características das joias, segundo essa classificação, são:

	Força	Destreza	Inteligência	Espírito
Espaço	80	80	40	0
Mente	0	0	100	60
Alma	0	0	60	100
Realidade	0	40	80	80
Tempo	50	100	20	0
Poder	100	40	20	10

Thanos também tem seus próprios atributos, que devem ser somados aos poderes das joias. A partir do momento que Thanos obtém mais joias, os atributos vão se acumulando também. A medida de facilidade em um combate é a soma das diferenças entre os atributos de Thanos e do herói ou grupo. Alguns exemplos:

$$\text{Thanos (60, 40, 80, 40)} \times \text{Capitão America (40, 30, 10, 20)} = 120$$

$$\text{Thanos (60, 40, 80, 40)} \times \text{Adam Warlock (30, 30, 70, 50)} = 60$$

$$\text{Thanos (60, 40, 80, 40)} \times \text{Galactus (120, 30, 70, 60)} = -60$$

Thanos deseja saber a melhor sequência para obter as joias. A melhor sequência é aquela que maximiza a soma das diferenças de todos os combates. Se houver mais de uma sequência que maximiza essa soma, escolha aquela com menor ordem lexicográfica.

Thanos é muito cauteloso, então ele não quer nenhum plano em que sua facilidade fique negativa a qualquer momento. O titã louco não arrisca!

### Entrada

A entrada é constituída de vários casos de testes. Cada caso de teste possui 7 linhas com 4 inteiros. A primeira linha são os atributos de Thanos, e as outras 6 os atributos dos heróis ou grupos que guardam as joias, na sequencia da tabela apresentada. A entrada termina quando os atributos de Thanos são negativos, este caso não deve ser processado.

### Saída

A saída deverá conter os seis nomes das joias ordenadas de forma que Thanos tenha a maior facilidade possível para obtê-las. Caso não seja possível obter as joias, o programa deve apresentar “Precisamos de mais poder antes dessa guerra!”.

### Exemplos de testes

Entrada	Saída
80 20 90 10	espaco poder realidade tempo alma mente
170 10 0 20	Precisamos de mais poder antes dessa guerra!
90 200 30 40	espaco poder realidade tempo alma mente
10 80 80 200	realidade poder tempo espaco alma mente
200 120 40 90	
50 150 80 20	
60 20 180 10	
80 20 90 10	
1700 10 0 20	
90 200 30 40	
10 80 80 200	
200 120 40 90	
50 150 80 20	
60 20 180 10	
60 40 80 40	
70 10 0 20	
90 20 30 40	
10 80 80 20	
200 120 40 90	
50 150 80 20	
60 20 180 10	
60 40 80 40	
50 500 80 20	
70 10 0 20	
90 20 30 40	
10 80 80 20	
200 120 40 90	
60 20 180 10	
-1 -1 -1 -1	

### Restrições

Todos os valores estão no intervalo [0; 10.000]