Maratona de Programação FACENS 2024

CADERNO DE PROBLEMAS

Informações gerais

Este caderno contém 11 problemas; as páginas estão numeradas de 1 a 19, não contando esta folha de rosto.

Verifique se o caderno está completo.

1) Sobre os nomes dos programas

Sua solução deve ser chamada arquivo.c, arquivo.cpp, arquivo.py ou arquivo.java, onde "arquivo" é especificado em cada problema. Lembre que em Java o nome da classe principal deve ser igual ao nome do arquivo.

2) Sobre a entrada

A entrada de seu programa deve ser lida da entrada padrão.

A entrada pode ser composta de um ou mais casos de teste, descritos em um número de linhas que depende do problema.

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. Cada linha, incluindo a última, contém exatamente um caractere final-de-linha.

3) Sobre a saída

A saída de seu programa deve ser escrita na saída padrão.

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.

Cada linha, incluindo a última, deve conter exatamente um caractere final-de-linha.

3) Sobre o limite de tempo

O limite de tempo dos exercícios para C, C++ e Python 2/3 é 1 segundo.

O limite de tempo para Java é 5 segundos (sinceramente, Java demora muito para compilar!)

A: MULTIPLICAÇÃO DE CADEIA DE MATRIZES

Arquivo: `mult.[cpp/c/java/py]`

Cor: branco

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 × 30, B é uma matriz 30 × 5, e C é uma matriz 5 × 60. Então,

(AB)C = (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operações

A(BC) = (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operações.

Aqui, foi assumido que a quantidade de operações para multiplicar uma matriz P x Q por uma matriz Q x R é P*Q*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 2 3 3 4 3 1 1 1 5 5 2 3 2 3 3 4 4 5 3 10 30 30 5 5 60 0	24 12 64 4500

Restrições

$1 \leq N \leq 100$

$1 \leq P, Q \leq 10.000$

B: GUERRA FRIA

Arquivo: `guerra.[cpp/c/java/py]`

Cor: marrom

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 utilizada 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$ ERROR D0NQVU@D@0 3WP3WQ@A!@ A@0#G3Z#ZHAQ!@Y

Restrições

1 ≤ N ≤ 20

1 ≤ |S| ≤ 51 (tamanho de S)

C: TRIÂNGULO DE PASCAL

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

Cor: verde claro

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 triangulo 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 prgrama 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 2 -1	1 => 1*(A^0*B^0) 1 1 => 1*(A^1*B^0) + 1*(A^0*B^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

$1 \leq N \leq 15$

D: LOUSA DIGITAL

Arquivo: `lousa.[cpp/c/java/py]`

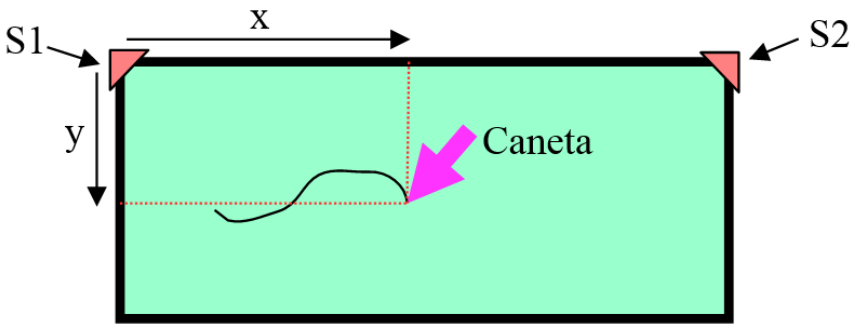
Cor: azul escuro

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

Entrada	**Saída**
2000 1000 500 1500 1000 800 750 1250 2000 1000 500 500 0 0 0 0	500 0 0 750 DEFEITO

Restrições

0 ≤ A, B, C, D ≤ 10.000

PROBLEMA E: COORDENADA GEODÉSICA

Arquivo: `coordenada.[cpp/c/java/py]`

Cor: rosa

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: 1º equivale a 1 hora

Dica 2: Para não existir problemas de formato a coordenada 23º27'07"S 47º27'06"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 23g27m07sS 47g27m06sW 23g31m25sS 47g08m39sW 3 42g20m55sN 71g03m38sW 34g55m32sN 104g08m30sE 29g45m39sN 95g21m36sW 0	-23.45 -47.45 -23.52 -47.14 42.34 -71.06 34.92 104.14 29.76 -95.36

Restrições

1 ≤ N ≤ 100

F: RESPOSTA PARA VIDA, O UNIVERSO E TUDO MAIS

Arquivo: `resposta.[cpp/c/java/py]`

Cor: amarelo

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`

G: AVIÃO

Arquivo: `aviao.[cpp/c/java/py]`

Cor: laranja

Descrição do problema

Alanito é um fiscal de aviação e precisa por várias vezes realizar reuniões e auditorias em diversas 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 kilometros por hora, e uma nova norma internacional da aviação determina que um avião não deve realizar um voo de mais de 12 horas continuas, 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 aerorporto 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 aerporto de destino e a distância em kilometros 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 caracateres 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 SFO -7 JFK -4 SCL -4 GRU -3 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	SIM NAO NAO NAO SIM

Restrições

3 ≤ A ≤ 20

-12 ≤ F ≤ 12

3 ≤ V ≤ 100

100 ≤ D ≤ 20.000

1 ≤ R ≤ 50

0 ≤ S ≤ 23

0 ≤ C ≤ 48

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

H: CAMINHO MAIS RÁPIDO

Arquivo: `caminho.[cpp/c/java/py]`

Cor: preto

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 ser 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 Eij desta tabela representa o tempo estimado de percurso na via que liga os pontos i e j. Note que o valor do elemento Eij pode ser diferente do valor do elemento Eji pois representam sentidos diferentes da mesma via. Os valores preencherão a tabela por linhas, ou seja, E11, E12, E13, ..., E21, E22, ... 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 Eij 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 4 0 3 0 0 0 0 0 0 0 1 0 3 3 3 0 0 4 5 5 1 5 5 5 5 1 5 1 5 5 5 5 5 5 0	3 CAMINHO INEXISTENTE 3

Restrições

$1 \leq N \leq 100$

$0 \leq Eij \leq 120$

I: GUERRA INFINITA

Arquivo: `infinity.[cpp/c/java/py]`

Cor: vermelho

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:

Thanos (60, 40, 80, 40) x Capitão America (40, 30, 10, 20) = 120

Thanos (60, 40, 80, 40) x Adam Warlock (30, 30, 70, 50) = 60

Thanos (60, 40, 80, 40) x 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 10 170 10 0 20 90 200 30 40 10 80 80 200 200 120 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	espaco poder realidade tempo alma mente Precisamos de mais poder antes dessa guerra! espaco poder realidade tempo alma mente realidade poder tempo espaco alma mente

Restrições

Todos os valores estão no intervalo [0; 10.000]

J: MEGA SENA

Arquivo: `megasena.[cpp/c/java/py]`

Cor: azul claro

Descrição do problema

O Paulo gosta de apostar na Mega Sena, mas não gosta de ficar conferindo os jogos que faz. Ele gosta de apostar em mais do que 6 dezenas no mesmo bilhete para maximizar suas chances de acertar a sena. Então ele precisa de um programa que faça a conferência por ele e indique quantas quadras, quinas e a sena que ele acertou em cada aposta. Para apostar na Mega Sena deve-se pegar um cartão de apostas e marcar ao menos 6 dezenas e no máximo 15 dezenas (uma dezena é um valor entre 1 e 60). Marcando mais dezenas você concorre com todas as combinações de 6 dezenas possíveis formadas com as dezenas marcadas. Não é possível marcar, nem sortear, dezenas repetidas no mesmo cartão. Não é possível ganhar mais de uma vez com cada combinação de 6 dezenas.

Entrada

A entrada é composta por várias linhas. A primeira linha contém 6 números inteiros, sem repetição, que foram os números sorteados. A seguir uma linha com um inteiro N (1<=N<=100) que indica a quantidade de jogos que Paulo fez. As próximas N linhas contêm, cada uma, uma aposta feita por Paulo. A aposta deve ter pelo menos 6 dezenas e pode ter até 15 dezenas ou até o usuário digitar 0.

Saída

Para cada aposta digitada o programa deve mostrar quantas quadras, quinas e sena que Paulo acertou. A saída de uma aposta deve ser composta por três linhas:

- A primeira linha deve exibir o texto “Quadras:” seguido de um espaço em branco e então a quantidade de quadras que Paulo acertou naquela aposta.
- A segunda linha deve exibir o texto “Quinas:” seguido de um espaço em branco e então a quantidade de quinas que Paulo acertou naquela aposta.
- A terceira linha deve exibir o texto “Sena:” seguido de um espaço em branco e então a quantidade de senas que Paulo acertou naquela aposta.

Exemplos de testes

Entrada	**Saída**
15 32 45 1 5 58 4 15 32 45 1 5 58 0 9 18 27 36 45 54 0 5 10 15 20 25 30 35 40 45 50 55 60 0 2 45 32 23 59 58 44 41 12 5 1 26 17 34 53	Quadra: 0 Quinas: 0 Sena: 1 Quadras: 0 Quinas: 0 Sena: 0 Quadras: 0 Quinas: 0 Sena: 0 Quadras: 225 Quinas: 10 Sena: 0

K: DISNEY WARS - EPISÓDIO XXIX

Arquivo: `disney.[cpp/c/java/py]`

Cor: cinza

Descrição do problema

Darth Sion ressurgiu através de um ritual Sith, que permite trazer de volta a vida um ser através do lado negro da força.

Como sempre, os Siths se reergueram, e planejam o domínio completo da galáxia através da construção de uma nova estrela da morte.

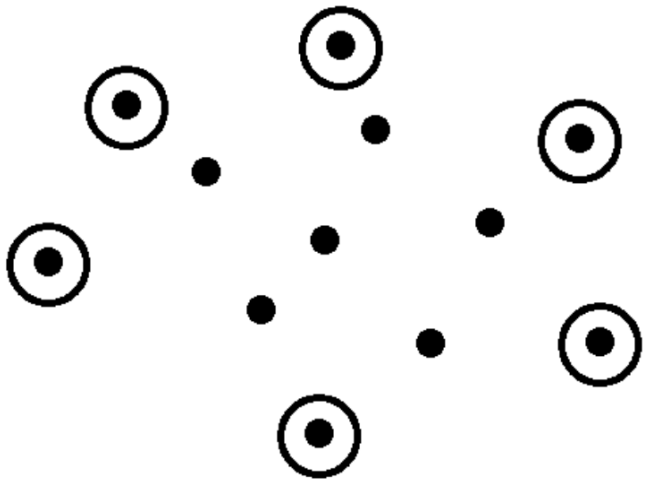
Darth Sion resolveu inovar. Desta vez, a estrela da morte será efetivamente uma estrela, e usará toda sua energia para disparar e destruir um sistema inteiro.

Após várias tentativas frustradas, inclusive uma na qual a estela da morte original foi destruída pelo Jedi mais pastel da história (afinal ele nunca venceu uma batalha) e outra na qual ela foi destruída pelo Apocalipse, quero dizer, Poe Dameron, Darth Sion resolveu proteger melhor sua arma de destruição em massa.

Buscando desta vez finalmente conseguir o domínio total, Darth Sion resolveu apelar para seu aprendiz, você, que tem habilidades da força de programação (ou não).

Para isso, eles mapearam todos os planetas no sistema que contém esta estrela. A ideia que todos os planetas que estejam nas fronteiras externas do sistema possuam um posto de vigilância.

Por desacreditar no seu poder, Darth Sion já simplificou o mapa do sistema deixando o mesmo em duas dimensões, conforme exemplo abaixo, no qual os pontos circundados são os locais aonde os postos de vigilância serão construídos.



Após lhe passar esta tarefa, Darth Sion questiona sua capacidade, saindo da ala da nave com a seguinte frase:

- I have studied you, and found nothing but weakness

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 é composta por um inteiro N (3 ≤ N ≤ 100) que indica a quantidade de planetas do sistema solar. A segunda linha de um caso de teste é composto pelas coordenadas inteiras (X, Y) (0 ≤ X,Y ≤ 500) dos N planetas dispostos no sistema solar.

As entradas se encerram quando N=0.

Saída

Para cada caso de teste, seu programa deve exibir uma linha contendo as coordenadas dos planetas nos quais serão construídos os postos de vigilância. Estas coordenadas devem ser separadas por um espaço em branco e ordenadas por X e então por Y.

Exemplos de testes

Entrada	**Saída**
7 1,10 2,3 10,10 5,5 10,1 1,1 7,2 0	1,1 1,10 10,1 10,10