I Simulado para Ingressantes 2015

Caderno de Problemas

Universidade de São Paulo

e

Universidade Federal do ABC

Sexta-feira, 24 de abril de 2015.

Instruções

- A competição tem duração de 5 horas;
- Faltando 1 hora para o término da competição, o placar será congelado, ou seja, o placar não será mais atualizado;
- Faltando 15 minutos para o término da competição, os times não receberão mais a resposta de suas submissões.

- A entrada de cada problema deve ser lida da entrada padrão (teclado);
- A saída de cada problema deve ser escrita na saída padrão (tela);
- Siga o formato apresentado na descrição da saída, caso contrário não é garantido que seu código será aceito;
- Na saída, toda linha deve terminar com o caracter '`\n`';
- O nome do arquivo de códigos em Java deve ser **exatamente** como indicado abaixo do nome de cada problema. Para C/C++ é recomendado usar o nome indicado;
- Para códigos em Java, o nome da classe principal deve ser **igual** ao nome do arquivo.

Respostas das submissões		
Not answered yet	-	Paciência
YES	-	Código aceito. Parabéns!
NO	Compilation error	Erro de compilação
	Wrong answer	Errado. Pode tentar de novo.
	Time limit exceeded	Seu programa demora muito para dar a resposta (certa ou errada)
	Runtime error	Erro em tempo de execução (ex.: *segmentation fault*)
	Problem name mismatch	Leia as duas últimas instruções
	Presentation error	Não está imprimindo no formato exigido no enunciado
	If possible, contact staff	Não sei, você conseguiu fazer algo inesperado

Problema A: Bactérias

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

Os pesquisadores do Instituto de Pesquisas Bacteriais (IPB) estão investigando a taxa de crescimento de diversos tipos de bactérias diferentes, essas bactérias obedecem a duas regras básicas:

- Ao final de cada dia, todas bactérias se subdividem para dar origem a novas bactérias. O número de bactérias geradas por cada uma é dependente de de seu tipo, esse número é definido por a_i para o i-ésimo tipo de bactéria. Se um tipo de bactérias tem $a_i = 2$, o número de bactérias nos primeiros dias será: $1, 2, 4, 8, \ldots$
- As bactérias param de se multiplicar depois de um número definido de dias, determinado por b_i para o i-ésimo tipo de bactéria.

Os pesquisadores do IPB já conseguiram descobrir o número de subdivisões e número de dias que elas crescem para os N tipos de bactérias. Agora eles precisam da sua ajuda, eles querem saber o tipo de bactéria que gera o maior número de bactérias ao final de seu período de crescimento.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A primeira linha de cada instância é composta por um inteiro N, o número de tipos de bactérias existentes. Cada uma das N linhas seguintes contém dois inteiros: a_i e b_i, contendo o número de subdivisões e número de dias de crescimento do i-ésimo tipo de bactéria, respectivamente.

Todos os valores finais de bactérias serão distintos.

Saída

A saída de cada instância deve ser composta por um inteiro P, que é o índice do tipo de bactéria com maior número de bactérias ao final do período de crescimento. Os tipos de bactéria são indexados começando em 1.

Restrições

- $3 \leq N \leq 10.000$
- $1 \leq a_i \leq 100.000$
- $1 \leq b_i \leq 10.000$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
2 2 5 3 3 3 100 5 200 3 5 100 2 2 1000 1000 2	1 3 1

Problema B: Cavaleiros da Távola Redonda

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

O rei Artur se encontra em uma difícil situação. A Inglaterra medieval tem crescido muito em tamanho e número de habitantes para que possa ser governada pelo rei sem precisar sair de Camelot. Além disso, o rei Artur está velho, e por isso, pretende delegar esta tarefa aos cavaleiro da Távola Redonda. A Távola Redonda é composta por: Sir Lancelot, Sir Gawain, Sir Gearing, Sir Percival, Sir Bors, Sir Lamorak, Sir Kay, Sir Gareth, Sir Bedivere, Sir Gaheris, Sir Galahad e Sir Tristan.

A intenção do rei Artur é que cada cavaleiro seja responsável por um conjunto de habitantes. Essa divisão deve satisfazer as seguintes restrições:

- um cavaleiro deve ficar cuidando do rei e de Camelot;
- o mesmo número de pessoas deve ser atribuída a cada cavaleiro, caso contrário, algum deles pode pensar que o rei não é justo e tem preferência por alguns cavaleiros.

Então, o rei Artur chamou o mago Merlin para que ele lhe responda se é possível fazer essa divisão do reino. Como Merlin não é bom com os números, mas sim com a mágica, usou ela para viajar no tempo e pedir sua ajuda.

A sua tarefa é, dado o número N de pessoas na Inglaterra medieval, responder se é possível fazer uma divisão entre os cavaleiros da Távola Redonda que satisfaz as restrições mencionadas anteriormente.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A única linha de cada instância é composta por um inteiro N, o número de habitantes da Inglaterra medieval.

Saída

A saída de cada instância deve estar composta por uma linha contendo um único caráter. Imprima "`Y`", se que é possível fazer a divisão do reino satisfazendo as restrições do enunciado, caso contrário, imprima "`N`".

Restrições

- $2 \leq N \leq 10^{10000}$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
121 1009000000900	Y N

Problema C: Imparmiopia

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

Recentemente o renomado e gentil alemão Dr. Hans Wondabar Chucrutes descobriu uma nova e intrigante patologia oftalmológica que ele batizou de imparmiopia. O portador da imparnopia tem modificações em todos os genes ímpares, o que o faz enxergar apenas números ímpares.

O Dr. Hans Chucrutes descobriu essa patologia ao pedir que seu sobrinho Herr Schweinsteiger, carinhosamente chamado de Schweini, procurasse o número de um delivery de *Sauerkraut* na lista telefônica, mas o pequeno sobrinho falava o número telefônico incompleto. Então o Dr. Hans Chucrutes criou o novo teste optométrico chamado de imparnoscopia de oclusão reflexiva imparóptica sensível a imparidade exclusiva ou inclusiva. Esse teste é muito simples, o médico dá ao paciente um número inteiro, e o paciente deve dizer a quantidade de dígitos que o número possui. Se ele falar a quantidade exata de dígitos, então ele não é portador da imparnopia, mas se ele falar a quantidade exata de dígitos impares do número, então ele é um imparmiope.

Mas há um problema neste teste. E se o médico também for imparmiope? Por este motivo, a empresa *Donaudampfschifffahrtselektrizitätenhauptbetriebswerkbauunterbeamtengesellschaft* te contratou para fazer um programa de computador que ajude o Dr. Hans Chucrutes dizendo qual a quantidade de dígitos impares em um número inteiro.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância consiste de uma linha contendo apenas um número inteiro N.

Saída

Para cada instância, imprima uma única linha, apresentando a quantidade de dígitos ímpares do inteiro N.

Restrições

- $0 \leq N \leq 2^{32} - 1$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
1234567 24680 13579	4 0 5

Problema D: *Free shop*

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

Free shops são um grande atrativo para os que viajam de avião. Eles são conhecidos por vender produtos com um grande desconto, mas nem sempre isso é verdade.

Mariana é comissária de voo da empresa multinacional Poli Aviação Ltda e deseja trazer da sua próxima viagem AiFones afim de revendê-los no Brasil.

A viagem de Mariana parará em N aeroportos na viagem (incluindo o de partida e o de chegada), numerados de 1 a N, sendo que cada aeroporto tem um único *Free shop*.

O i-ésimo *Free shop* tem um coeficiente c_i, indicando a diferença entre o preço cobrado pelo AiFone no *Free shop* e no shopping do lado de sua casa.

Ou seja, se $c_i > 0$, o AiFone no *Free shop* é mais caro que no shopping, se $c_i < 0$, o AiFone no *Free shop* é mais barato que no shopping e se $c_i = 0$, o AiFone no *Free shop* tem o mesmo preço que no shopping.

Como Mariana fica pouco tempo em cada aeroporto, ela consegue comprar somente um AiFone em cada *Free shop*.

Além disso, ela deseja comprar todos os AiFones em sequência, ou seja, comprando um AiFone nos aeroportos $a - 1$ e $a + 1$, ela obrigatoriamente compra um no aeroporto a, podendo escolher em qual aeroporto iniciar e terminar as compras.

Você deve escrever um programa para ajudar Mariana a determinar qual o maior lucro que ela pode obter. Note que, se não houver uma opção em que ela economize, ela pode optar por não comprar nada durante a viagem.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância é composta por um inteiro N, o número aeroportos que Mariana parará, seguido de N inteiros, indicando o coeficiente de cada *Free shop*.

Saída

A saída de cada instância deve ser composta por uma única linha, contendo um único inteiro dizendo qual é o maior lucro que Mariana pode obter.

Restrições

- $2 \leq N \leq 100.000$
- $-1.000 \leq c_i \leq 1.000$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
6 10 -2 -3 6 -4 0 4 -2 -2 1 -5 3 4 6 8	5 8 0

Problema E: Competição Pirata

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

Sabe-se que piratas, quando não estão lutando contra autoridades, outros piratas ou desbravando novos mundos, eles bebem rum.

Muitos especialistas classificam a bebida como um estimulante mais eficiente que café, energético e pó de guaraná combinados.

Recentemente historiadores encontraram relatos de uma competição realizada por piratas em suas paradas, para conquistar respeito diante de seus companheiros.

A competição segue as seguintes regras:

- dois piratas jogam um contra o outro, sendo que o campeão prévio começa;
- ambos os jogadores compartilham, inicialmente, N canecos de rum;
- cada pirata, na sua vez, pode tomar de 1 a K canecos de rum;
- o pirata que não tiver mais canecos para tomar na sua vez perde.

O duelo mais aclamado de todos os tempos foi entre Colombo, o descobridor, e Carolina, a sagaz. Sabe-se que, como Colombo era o campeão, ele começou. Mas ninguém sabe o desfecho...

Você foi contratado por estes historiadores para criar um programa de computador que descubra quem ganhou. Como tanto Colombo quanto Carolina eram exímios bebedores de rum, pode-se assumir que eles jogam da melhor maneira possível sempre.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância é composta por dois inteiros: N, o número de canecos iniciais, e K, o número máximo de canecos que cada pirata pode tomar na sua vez.

Saída

A saída de cada instância deve ser composta por uma única linha, contendo o nome do vencedor daquela competição: `‘‘Colombo’’` ou `‘‘Carolina’’`.

Restrições

- $1 \leq N \leq 1.000.000$
- $1 \leq K \leq 1.000.000$

Exemplos

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

`5 3`
`30 5`
`3 4`
	`Colombo`
	`Carolina`
	`Colombo`

Problema F: IntegraMIX

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

O IME pode ser um lugar muito estressante. São provas, trabalhos e um milhão de listas para fazer, você precisa de muita dedicação e de ajuda de Goku para conseguir fazer tudo. Por isso, alguns alunos costumam ir em festas e eventos para aliviar a tensão. João é um desses alunos e quer muito ir no IntegraMIX desse ano. Mas, ele tem um problema: é um universitário falido que não tem dinheiro para pagar o ingresso. Como João não pode vender nenhum órgão, pois precisa deles para se formar, foi pedir dinheiro ao seu pai que é professor de matemática. O pai de João lançou o seguinte desafio: "Eu vou te dar uma cadeia S, contendo apenas letras. Se você achar uma cadeia s tal que $s^t = S$ (onde s^t é a concatenação de t cópias de s) eu vou te dar $t * 10$ reais".

Como João está mesmo precisando de dinheiro, ele pediu a você para que fizesse um programa de computador que calculasse essa string s de forma que ele ganhe o máximo possível. Por exemplo, seja $S = aaaaaa$. Se $s = aaa$ temos que $t = 2$, se $s = aa$ temos que $t = 3$, e se $s = a$ temos $t = 6$. Logo, João responderá $s = a$ para seu pai, pois com ela t atinge o maior valor possível.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância é composta por uma única linha. Essa linha contém uma cadeia S que consiste em letras minúsculas.

Saída

Para cada instância imprima uma única linha contendo um número que representa o tamanho da cadeia s tal que $s^t = S$ e t é máximo.

Restrições

- $1 \leq |S| \leq 100.000$

Exemplos

Exemplo de entrada	Saída para o exemplo de entrada
`abab` `ccc`	`2` `1`

Problema G: Velha

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

Alguns alunos do IME gostam muito de jogar jogos na vivência quando não estão em aula. O problema é que eles jogam jogos muito emocionantes e, por isso, fazem muito barulho comemorando suas vitórias e lamentando suas derrotas, o que atrapalha levemente nas aulas que acontecem perto (ou longe) dali. Como esse assunto já vem gerando muita discussão, os IMEanos resolveram procurar outros jogos menos emocionantes, como, por exemplo, o jogo da velha.

O problema é que, de tanto jogar seus jogos habituais, eles esqueceram como se joga todos os outros jogos, até os mais básicos, e estão reaprendendo, mas tem certas dificuldades em definir quando o jogo acabou e quem ganhou (ou não) no final do jogo.

Assim, ficou para você a tarefa de pesquisar sobre esse jogo super complexo e criar um programa que dada uma configuração de jogo da velha (um tabuleiro 3×3), diz se algum jogador já ganhou. Após árdua pesquisa você descobriu que, no jogo da velha, o jogador `x` ganha quando existem 3 `x` alinhados, o jogador `o` ganha quando existem 3 `o` alinhados. Lembrando que vale alinhar `x` ou `o` em qualquer uma das linhas, colunas ou das duas diagonais do tabuleiro.

Entrada

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

Cada instância consiste em 3 linhas representando as 3 linhas do jogo da velha. Cada linha consiste de 3 caracteres, "`.`" se a célula não estiver ocupada, "`x`" se for ocupada por um `x`, ou "`o`" se for ocupada por um `o`.

Saída

Para cada caso de teste imprima uma linha contendo "`Y`" se alguem já venceu o jogo, ou "`N`" caso contrário.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
2 oo. xx. xx. ox. .o. ..o	N Y

Problema H: Ideais principais

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

Lua é uma menina muito curiosa e inteligente. Hoje, ela está estudando um pouco de álgebra, mais especificamente, o conceito de ideais principais. Um conjunto de inteiros I é um **ideal principal** sobre $\mathbb{Z}$ se satisfaz a seguinte condição

$$I = a\mathbb{Z} = \{ar : r \in \mathbb{Z}\}, \text{ para algum } a \in \mathbb{Z}.$$

Dados dois ideais principais A e B, Lua gostaria de saber se $A \subseteq B$ ou $B \subseteq A$. Como ela é muito esperta, ela já sabe essa resposta. Porém, Lua quer testar vocês agora.

A sua tarefa é, dados dois inteiros a e b, que representam os ideais principais $a\mathbb{Z}$ e $b\mathbb{Z}$, respectivamente, responder se um deles contém ao outro.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A única linha de cada instância é composta por dois inteiros a e b, que representam os ideais principais $a\mathbb{Z}$ e $b\mathbb{Z}$, respectivamente.

Saída

A saída de cada instância deve estar composta por uma linha contendo um único caráter. Imprima "`Y`", se $a\mathbb{Z} \subseteq b\mathbb{Z}$ ou $b\mathbb{Z} \subseteq a\mathbb{Z}$, caso contrário, imprima "`N`".

Restrições

- $1 \leq a, b \leq 2^{31} - 1$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
3 5 1024 2 3 13 3 15	N Y N Y

Problema I: Lista de problemas ótima

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

Preparar uma boa lista de problemas para os treinos da Maratona de Programação para iniciantes não é uma tarefa tão simples quanto parece. Os problemas da lista devem ser suficientemente desafiadores para os iniciantes aprenderem coisas novas, mas não podem ser excessivamente difíceis a ponto de assustá-los e desincentivar a participação na competição.

Após vários anos tentando equilibrar esses dois fatores, percebeu-se que a quantidade de problemas na lista afeta bastante os chamados nível de satisfação e nível de dificuldade da lista. Mais precisamente, ambos são, respectivamente, funções da forma

$$n_s(x) = a_s x^2 + b_s + c_s \qquad \text{e} \qquad n_d(x) = a_d x^2 + b_d + c_d,$$

onde x é a quantidade de problemas na lista.

Uma possível explicação para isso é que os alunos costumam interpretar listas de problemas (ou exercícios) com poucos ou muitos problemas da mesma forma. Poucos problemas podem indicar que cada problema tem nível de dificuldade grande e, portanto, a lista é difícil. Muitos problemas indica que serão cobradas vários tópicos diferente e, portanto, a lista também é difícil. Bom, essa foi a explicação que um aluno deu, os demais não quiseram responder.

Para decidir o número de problemas nas próximas listas, foi criada uma nova medida: a *desincentivabilidade* da lista. A desincentivabilidade, d, é apenas a diferença entre o nível de dificuldade e o nível de satisfabilidade, ou seja $d(x) = n_d(x) - n_s(x)$. Sua tarefa é encontrar o número de problemas que minimiza a desincentivabilidade de uma lista de problemas.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância consiste de uma linha contendo seis inteiros, a_d, b_d, c_d, a_s, b_s e c_s, representando, respectivamente, os níveis de dificuldade e satisfação de uma lista de problemas.

Saída

Para cada instância, imprima uma linha com um inteiro x representando o número de problemas que minimiza a desincentivabilidade de uma lista. Se não existir tal x, imprima "`sem min`". Nos casos em que existir um valor mínimo, é garantido que x será um número inteiro.

Restrições

- $-1.000 \leq a_d, b_d, c_d \leq 1.000$
- $-1.000 \leq a_s, b_s, c_s \leq 1.000$
- $a_d \neq a_s$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
`4 6 2 2 -2 -1` `2 9 2 3 2 -3`	`-2` `sem min`

Problema J: Miojo[1]

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

João é um fanático por miojos; ele os adora, e, como era de se esperar, ele levou vários pacotes quando foi acampar com seus colegas. Como João só gosta de miojos feitos com o tempo exato, ele se desesperou ao perceber que havia esquecido seu relógio em casa.

Por sorte, ele conseguiu, no caminho, comprar duas ampulhetas de durações diferentes. Por exemplo, se o miojo precisa de 3 minutos para ficar pronto, e João tiver uma ampulheta de 5 minutos e outra de 7, uma possível forma de cozinhar o miojo é:

1. João começa virando as duas ampulhetas ao mesmo tempo.
2. Quando a areia da ampulheta de 5 minutos se esgotar, João torna a virá-la.
3. João começa a preparar o miojo quando a areia da ampulheta de 7 minutos acabar.
4. João tira o miojo do fogo quando a ampulheta de 5 minutos acabar novamente.

Dessa forma, o miojo ficará 3 minutos no fogo (do minuto 7 ao minuto 10). Assim, apesar do miojo levar apenas três minutos para ser cozido, ele precisa de 10 minutos para ficar pronto.

Faça um programa que, dado o tempo de preparo do miojo, e os tempos das duas ampulhetas (ambos maiores que o tempo do miojo), determina o tempo mínimo necessário para o miojo ficar pronto. Você pode supor que sempre é possível cozinhar o miojo no tempo correto.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A entrada é composta por uma única linha, que contém três inteiros T, A e B, representando o tempo necessário para o preparo do miojo, o tempo da primeira ampulheta e o tempo da segunda ampulheta respectivamente.

Saída

Para cada instância, seu programa deve produzir uma única linha na saída, contendo o tempo mínimo para o preparo do miojo.

Restrições

- $0 \leq T \leq 10.000$
- $T < A, B \leq 40.000$

Exemplos

Exemplo de entrada	Saída para o exemplo de entrada
`3 5 7` `14 15 22`	`10` `44`

[1]Problema originalmente apresentado no treino para OBI de 2006

Problema K: Jenyffer, Rute, Raquel, Cláudia e Narcisa

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

A coisa mais importante na vida de qualquer IMEano é o café. Estudiosos dizem que 90% dos alunos saem pelo menos uma vez por aula para comprar café de uma das máquinas do IME-USP. E é por isso que essas máquinas são tão importantes.

Atualmente, o IME-USP tem 5 máquinas de café conhecidas, a Jenyffer, a Rute, a Raquel, a Cláudia e a Narcisa, que trazem consigo um sorriso cativante capaz de alegrar até uma complexa manhã de trancaço[2].

Para a felicidade geral do IMEano, muitas outras máquinas serão colocadas no IME-USP, deixando-nos com um total de N máquinas de café. O problema é que, agora, ficar passando por cada máquina para ver se ainda existe café nela vai se tornar uma tarefa muito trabalhosa. Então, o IME-USP disponibilizará uma cota diária de café para cada um em cada máquina.

Sabendo que você quer comprar M copos de café em determinado dia e que você vai sempre querer esgotar sua cota em uma máquina antes de ir comprar em outra, sua função é descobrir qual é a quantidade mínima de máquinas de café que você vai visitar no dia. Por exemplo, imagine que ainda existam 5 máquinas de café e você quer comprar 20 copos em um dia. Se suas cotas de café para as máquinas forem 4, 1, 17, 3 e 4, você vai ter que visitar no mínimo duas máquinas para conseguir comprar todos esses cafés.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância é dada por duas linhas. A primeira contém os inteiros N e M, o número de máquinas de café e a quantidade de café que você quer comprar, respectivamente. A linha seguinte contém N inteiros indicando as cotas diárias de cada máquina.

Saída

Para cada instância, imprima uma linha contendo a quantidade mínima de máquinas de café que você precisará visitar. É importante que, se não houver cotas suficientes para comprar a quantidade de café que você quer, seu mundo desmoronou. Nesse caso, imprima "`NOOOO!`".

Restrições

- $1 \leq N \leq 1.000$
- $0 \leq M \leq 1.000$
- $0 \leq$ cota por máquina ≤ 50

Exemplos

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

`5 20`
`4 1 17 3 4`
`4 7`
`0 1 2 3`
	`2`
	`NOOOO!`

[2]Forma de protesto no qual os protestantes bloqueiam a entrada principal da USP.