Maratona de Programação FACENS 2022

CADERNO DE PROBLEMAS

Informações gerais

Este caderno contém 11 problemas; as páginas estão numeradas de 1 a 22, 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 de A até J para C, C++ e Python 2/3 é 1 segundo. O limite de tempo do exercício K é 2 segundos.

O limite de tempo para Java é 5 segundos para os exercícios de A até J; 6 segundos para o exercício K (sinceramente, Java demora muito para compilar!)

A: Porque as árveres somos nozes

Arquivo: arvore.[cpp/c/java/py]
Cor: dourado

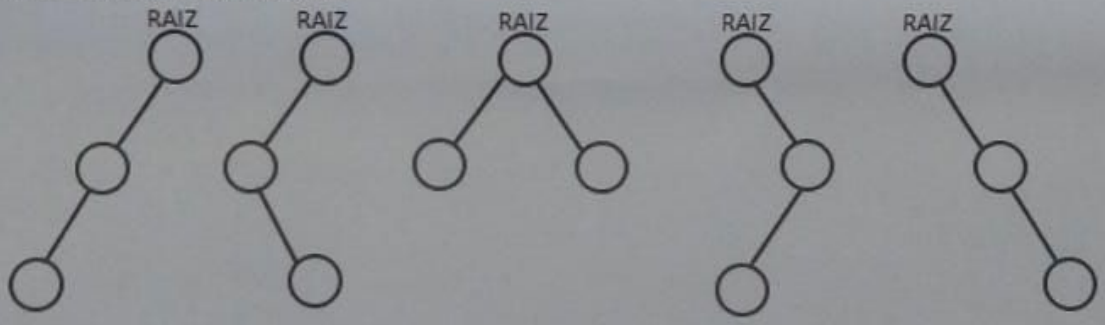
Eugenio gostava de contar! Ele pensou em uma estrutura um tanto curiosa e decidiu contar de quantas maneiras ele poderia construir tal estrutura com um determinado tamanho.

Para construir essa estrutura, ele primeiro cria uma bola inicial (chamada raiz). A partir da raiz, ele pode adicionar até duas bolas (ou vértices); uma à esquerda e outra à direita. A partir de cada um desses vértices, ele pode adicionar até outros dois vértices; sempre a direita e/ou a esquerda. Ele repete esse processo até que a estrutura (também conhecida como árvore binária) tenha N vértices (tamanho N).

Só existe um jeito de criar uma árvore binária de tamanho 1: criar apenas a raiz. Existem 2 maneiras de criar uma árvore binária de tamanho 2:

- raiz e vértice a esquerda, ou;
- raiz e vértice a direita.

Quando falamos de árvore de tamanho 3, existem 5 possibilidades. A *imagem a seguir* mostra essas 5 possibilidades.



Como você pode perceber, a quantidade de possibilidades cresce um tanto rápido. Dado um tamanho T, você deve calcular essa quantidade de possibilidades.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por uma linha com um inteiro T, o tamanho da árvore binária.

Saída

Para cada caso de teste, exiba uma linha contendo a quantidade de maneiras de construir uma árvore binária de tamanho T.

Restrições

$1 \leq N, T \leq 100$

Continua no verso >

Exemplo

Entrada	Saída
4 1 2 3 7	1 2 5 429

B: Ursos e doces

Arquivo: doce.[cpp/c/java/py]
Cor: preto

Os ursos adoram doces e jogos que envolvam comê-los. Limak e Bob jogam o seguinte jogo: Limak come 1 doce, então Bob come 2 doces, Limak come 3, Bob 4 e assim por diante. Quando um deles não consegue comer a quantidade que deveria, ele perde.

Limak consegue comer no máximo L doces no total e Bob consegue comer B doces no total. Quem vai ganhar o jogo? Exiba Limak ou Bob.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por uma linha com dois inteiros L e B.

Saída

Para cada caso de teste, exiba uma linha contendo o nome do vencedor.

Restrições

$1 \leq N \leq 1.000$
$1 \leq L, B \leq 1.000$

Exemplo

Entrada	Saída
10 3 2 4 2 1 1 1 2 1 3 9 3 9 11 9 12 9 1000 8 11	Bob Limak Limak Bob Bob Limak Limak Bob Bob Bob

Primeiro caso de teste: com L = 3 e B = 2, Limak come 1 doce, Bob come 2. Limak deveria comer 3 doces mas isso o faria comer 4 (1 + 3) doces no total. Isso é impossível pois ele pode comer no máximo 3 doces, então ele perde. Bob é o vencedor!

Segundo caso de teste: L = 4 e B = 2, Limak come 1 doce, Bob come 2, Limak come 3 doces (1 + 3 = 4 doces no total, o que é permitido pois não ultrapassa o valor de L). Agora Bob deveria comer 4 doces mas ele não pode comer mais nenhum (ele já comeu 2 doces). Bob perde e Limak ganha.

Continua no verso >

Oitavo caso de teste: L = 9 e B = 12, o jogo segue:
Limak come 1 doce.
Bob come 2 doces.
Limak come 3 (4 no total).
Bob come 4 (6 no total).
Limak 5 (9 no total).
Bob 6 (12 no total).
Limak deveria comer 7 doces mas ele não pode (7 + 9 = 16 > 9)
Bob ganha.

C: É mais de 8000!

Arquivo: maisoitomil.[cpp/c/java/py]
Cor: amarelo

No universo de Dragon Ball, a força dos guerreiros é representada pelo chamado poder de luta. Para descobrir a força de um oponente, os guerreiros de Freeza usam um equipamento que consegue analisar o poder de luta, chamado rastreador.

Após a luta contra Goku, o rastreador de Vegeta ficou com defeito. O rastreador agora mostra o valor do poder de luta criptografado. O equipamento mostra o valor 4 quando o poder de luta é 24. Para um poder de luta de 120, o rastreador exibe 5.

Vegeta está ameaçando destruir a Terra se o rastreador não for consertado. Será que vocês conseguem descobrir como descriptografar o equipamento?

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por uma linha com um inteiro C, o valor criptografado.

Saída

Para cada caso de teste, exiba uma linha com o poder de luta descriptografado.

Restrições

$1 \leq N \leq 1.000$
$1 \leq C \leq 25$

Exemplo

Entrada	Saída
3 4 5 6	24 120 720

D: Quick Maths

Arquivo: matematica.[cpp/c/java/py]
Cor: verde claro

Um músico e matemático famoso está com problemas em polinômios. Será que você consegue ajudar?

Você conhece as R raízes r_i de um polinômio da forma

$$L(x) = \prod_{i=1}^{R} (x - r_i)$$

As raízes são distintas entre si, ou seja, não existem raízes repetidas.

Para ajudar o músico, você deve responder a P perguntas numeradas de 1 a P. Para cada pergunta P_j, você receberá um valor W_j e deverá responder se $L(W_j)$ é positivo, negativo ou zero.

Entrada

A entrada consiste de apenas um caso de teste. A primeira linha do caso de teste contém os inteiros R e P, as quantidades de raízes e perguntas, respectivamente. A segunda linha contém R inteiros R_i: as raízes de $L(x)$.

Cada uma das próximas P linhas contém um inteiro W_j - o valor a ser avaliado.

Saída

Para cada pergunta P_j, exiba uma linha contendo:

- "+", se $L(W_j) > 0$;
- "-", se $L(W_j) < 0$;
- "0", se $L(W_j) = 0$.

Restrições

$1 \leq R, P \leq 100.000$
$-1.000.000.000 \leq R_i, W_j \leq 1.000.000.000$

Exemplo

Entrada	Saída
4 6 1 100 5 3 -2 2 4 80 107 5	+ - + - + 0

Nesse caso de teste, $L(X) = (x - 1)(x - 100)(x - 5)(x - 3)$

Primeira pergunta: $x = -2$
$P(-2) = (-2 - 1)(-2 - 100)(-2 - 5)(-2 - 3) = 10710 > 0$, logo "+"

Segunda pergunta: $x = 2$
$P(2) = (2 - 1)(2 - 100)(2 - 5)(2 - 3) = -294 < 0$, logo "-"

Sexta pergunta: $x = 5$
$P(5) = (2 - 5)(5 - 100)(5 - 5)(5 - 3) = 0$, logo "0"

E: Mínima pizza comum

Arquivo: pizza.[cpp/c/java/py]
Cor: verde escuro

Você vai dar uma festa para seus A amigos. Você decidiu comprar pizzas para a festa.

Cada pizza tem exatamente F fatias. Você sabe que seus amigos ficam tristes se um ganha mais fatias que o outro. Além disso, você fica triste se sobram fatias. De forma mais formal, se você comprar P pizzas então todos ficam felizes se, e somente se, for possível distribuir igualmente as $F * P$ fatias entre os A amigos.

Você precisa descobrir o menor valor para P de forma que todos fiquem felizes. Vale lembrar que você não come pizza.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por uma linha com os valores de A e F.

Saída

Para cada caso de teste, exiba uma linha com o menor valor de P que faça todos felizes.

wh

fazer um laço para adicionar mais uma pizza até o valor de fatias * pizza ser divisível por amigos

Restrições

$1 \leq N \leq 200.000$
$1 \leq A, F \leq 1.000.000.000$

Exemplo

Entrada	Saída
3 2 2 2 3 4 2	1 2 2

Primeiro caso de teste: uma pizza tem 2 fatias e você vai convidar 2 amigos, então você pode comprar apenas 1 pizza e dar uma fatia para cada amigo.

Segundo caso de teste: uma pizza tem 3 fatias e você vai convidar 2 amigos, então você não consegue distribuir fatias de 1 pizza sem haver sobras. Se você comprar 2 pizzas, você pode dar 3 fatias para cada amigo.

Terceiro caso de teste: uma pizza tem 2 fatias e você vai convidar 4 amigos. Se você comprar 2 pizzas então você consegue dar 1 fatia para cada amigo.

F: Poligonando

Arquivo: poligonando.[cpp/c/java/py]
Cor: roxo

Um Engenheiro Agrimensor tem um negócio próprio onde ele precisa calcular áreas de alguns terrenos para seus clientes. Ele gostaria de ter um aplicativo para ajudar a calcular essas áreas.

Você deve desenvolver a lógica por trás das cortinas, onde toda a mágica acontece, sem se preocupar com a parte monótona e chata da interface com o usuário, que será de responsabilidade de outra pessoa. Para isso, sua camada de código deve receber uma lista de pares de coordenadas x e y do terreno que o Engenheiro irá capturar com seu GPS moderno e introduzir no sistema e, na sequência, deve calcular a área do terreno e apresentar na tela.

Nessa primeira versão, só serão considerados terrenos na forma de polígonos planos e convexos. Detalhe, o Engenheiro não tem nenhuma obrigação de informar os pontos em uma ordem específica. O sistema deve cuidar disso para ele.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso de teste é composto por várias linhas. A primeira linha de um caso de teste contém um inteiro P, a quantidade de pares ordenados. Cada uma das próximas P linhas contém dois inteiros X e Y; uma coordenada do polígono.

Saída

Para cada caso de teste, exiba uma linha com a área do polígono com precisão de 1 casa decimal.

Restrições

$1 \leq N \leq 1.0000$
$3 \leq P \leq 20$
$-1.000 \leq X, Y \leq 1.000$

Exemplo

Entrada	Saída
3 4 0 0 0 1 1 0 1 1 3 0 0 200 -100 200 100 4 5 5 -5 -5 -5 5 5 -5	1.0 20000.0 100.0

G: Robô

Arquivo: robo.[cpp/c/java/py]
Cor: azul escuro

Um chefe de cozinha comprou um robô que vai entregar os pratos para os clientes. Ele começou a testar o robô fazendo-o se movimentar de um lado para o outro em linha reta.

Inicialmente, o robô está em uma posição X e deve executar C comandos descritos em uma string S de tamanho C. Cada carácter de S é 'E' or 'D' indicando que o robô deve se movimentar uma unidade para a esquerda (decrementar X em 1) ou uma unidade para a direita (incrementar X em 1), respectivamente.

Quantos pontos distintos são visitados pelo robô após executar todos os comandos? Um ponto P é considerado visitado se a posição do robô antes ou depois de executar um comando é P.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por duas linhas. A primeira linha de um caso de teste contém dois inteiros C e X. A segunda linha contém a string S de tamanho C.

Saída

Para cada caso de teste, exiba uma linha contendo a quantidade de pontos visitados pelo robô.

Restrições

$1 \leq N \leq 100$
$1 \leq C \leq 100$
$-1.000.000 \leq X \leq 1.000.000$
S é composta por C caracteres, cada um sendo 'E' ou 'D'

Exemplo

Entrada	Saída
2 6 10 DDEEEE 2 0 EE	5 3

Primeiro caso de teste: o robô segue o caminho 10 > 11 > 12 > 11 > 10 > 9 > 8, portanto passou por 5 pontos: 8, 9, 10, 11 e 12.

Segundo caso de teste: o robô segue o caminho 0 > -1 > -2, portanto passou por 3 pontos: -2, -1 e 0.

H: Pulo do sapo

Arquivo: sapo.[cpp/c/java/py]
Cor: laranja

Existem S sapos numerados de 1 a S em uma linha. Cada sapo S_i começa na posição i, tem peso P_i e, cada vez que você encosta nele, ele dá um pulo com distância D_i para a direita. Todos os sapos têm pesos distintos, ou seja, não existem dois sapos com um mesmo peso.

Você pode encostar em quantos sapos quiser (inclusive zero), quantas vezes quiser em cada sapo e em qualquer ordem. Os sapos não interferem uns com os outros, ou seja, pode haver mais de um sapo em uma mesma posição.

A sua tarefa é ordenar os sapos por peso de forma crescente encostando o menor número possível de vezes neles. A ordenação só é válida se não houver dois sapos em uma mesma posição.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por três linhas. A primeira linha de um caso de teste contém um inteiro S. A segunda contém S inteiros que representam os pesos de cada um dos sapos: P_1, P_2, P_3, ..., P_S. A terceira linha contém S inteiros que representam as distâncias que cada sapo pula: D_1, D_2, D_3, ..., D_S.

Saída

Para cada caso de teste, exiba uma linha contendo a menor quantidade de vezes necessárias que você precisa encostar nos sapos para ordená-los por peso.

Restrições

$1 \leq N \leq 20.000$
$2 \leq S \leq 4$
$1 \leq P_i \leq S$
$1 \leq D_i \leq 5$

Exemplo

Entrada	Saída
3 3 3 1 2 1 4 5 3 3 2 1 1 1 1 4 2 1 4 3 4 1 2 4	3 6 5

Primeiro caso de teste: você pode encostar no primeiro sapo 3 vezes.

Segundo caso de teste: você pode encostar no primeiro sapo 4 vezes e 2 vezes no segundo sapo (4 + 2 = 6).

I: Quem usa táxi?

Arquivo: taxi.[cpp/c/java/py]
Cor: branco

Na sua cidade existem dois serviços de táxi que você pode utilizar. O primeiro serviço cobra X reais. O segundo serviço cobra Y reais.

Dados os valores de X e Y, qual serviço de táxi você vai utilizar?

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por uma linha com dois inteiros X e Y.

Saída

Para cada caso de teste, exiba uma linha contendo o serviço que você vai utilizar: PRIMEIRO, SEGUNDO ou QUALQUER.

Restrições

$1 \leq N \leq 100$
$1 \leq S \leq 1.000$

Exemplo

Entrada	Saída
3 30 65 42 42 90 50	PRIMEIRO QUALQUER SEGUNDO

FACILIMO

J: Xadrez, é o que tem pra hoje

Arquivo: xadrez.[cpp/c/java/py]
Cor: vermelho

A distância do tabuleiro de xadrez para dois pontos (X_1, Y_1) e (X_2, Y_2) em um plano cartesiano é definido por $max(|X_1 - X_2|, |Y_1 - Y_2|)$.
$|P|$ é o valor absoluto de P, ou seja, $|-4| = 4$

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por uma linha com quatro inteiros X_1, Y_1, X_2 e Y_2.

Saída

Para cada caso de teste, exiba uma linha contendo a distância do tabuleiro de xadrez.

Restrições

$1 \leq N \leq 1.000$
$1 \leq X_1, Y_1, X_2, Y_2 \leq 8$

Exemplo

Entrada	Saída
3 2 4 5 1 5 5 5 3 1 4 3 3	3 2 2

Primeiro caso de teste: distância entre $(2, 4)$ e $(5, 1)$ é
$max(|2 - 5|, |4 - 1|) = max(|-3|, |3|) = 3$

Segundo caso de teste: distância entre $(5, 5)$ e $(5, 3)$ é
$max(|5 - 5|, |5 - 3|) = max(|0|, |2|) = 2$

Terceiro caso de teste: distância entre $(1, 4)$ e $(3, 3)$ é
$max(|1 - 3|, |4 - 3|) = max(|-2|, |1|) = 2$

K: O chamado de Xululu

xululu.c

Arquivo: zululu.[cpp/c/java/py]
Cor: azul claro

Uma terrível entidade cósmica está adormecida, contudo, como seu poder é imensurável, ecos de seus sonhos mais profundos alcançam a humanidade e atraem seres humanos com algum tipo de conexão com o oculto.

H.P. Lovecraft temendo que Xululu não acorde a tempo.

Estas pessoas passam a compartilhar destes sonhos e devotam a sua vida a esta entidade. Os sonhos apresentam a dominação completa da Terra e de outros planetas, onde toda a humanidade é escravizada e passa a entoar um cântico uníssono fantasmagórico... "Salve Xululu!".

Xululu, apesar de estar adormecido, sente os passos da humanidade, as conquistas, as guerras, as mortes, a perdição. Ele enxerga que deve agir logo, caso contrário, a humanidade se exterminará em passos rápidos. Os sonhos do Grande Deus Adormecido já não são suficientemente rápidos para alcançar o número suficiente de devotos para que o ritual máximo seja realizado e, por fim, que o terror cósmico governe para sempre.

Aquele Que Se Levantará Novamente planeja utilizar as mesmas armas de destruição em massa que a humanidade criou para alcançar uma quantidade maior de devotos. Xululu enxerga como os humanos criam algoritmos baseados em estatística, matemática e inteligência artificial para otimizar o vício, ou aquilo que eles chamam de engajamento. O algoritmo escuta e analisa o comportamento de seus usuários e os rotula como membro de um grupo de pessoas com ideias parecidas. A partir desses grupos, a inteligência artificial começa a recomendar o mesmo tipo de conteúdo para determinados grupos. No primeiro momento, não parece haver nada de errado, mas Xululu sabe das limitações humanas e como esta raça não consegue enxergar as consequências de seus próprios atos. Na prática, o tipo de conteúdo que mais gera engajamento é aquele que mais divide e polariza. Além disso, o que é espalhado dentro de um grupo é a "verdade do grupo", não importando os fatos e conteúdos que circulam em outros meios. Estas armas podem influenciar nações e alterar o curso da história, basta que você apresente uma realidade alternativa para cada grupo, mostrando aquilo que o usuário quer ver e o fazer acreditar que o grupo dele é o único correto. Os humanos não imaginam que criaram a ferramenta perfeita para que Xululu acelere seu despertar. Em meio a tantas dificuldades que a população vem enfrentando, o Grande Deus Adormecido será apresentado como a única solução!

O Grande Sonhador chama seus súditos por meio de novos sonhos e dá ordens. Os devotos devem reunir as mentes mais brilhantes que compactuam com Sua Alteza e criar uma arma que compreenda e gere textos nas línguas humanas, com o objetivo de apresentar Xululu como a única alternativa para o inferno que se instaurou na Terra. Baseando-se em técnicas de Processamento de Linguagem Natural, a IA deve conseguir processar textos e entendê-los, desde o nível mais básico de compreensão, que é o conhecimento da forma e estrutura, até o mais alto nível de estruturação linguística, a pragmática! A IA aprenderá desde o básico da gramática até a retórica e argumentação de Aristóteles e, dessa forma, dissuadir os humanos de suas crenças e levá-los a aceitar Aquele Que Aguarda Dormindo como seu único e eterno líder escravista.

Você, caro maratonista, quando aceitou fazer parte desta competição, também aceitou fazer parte do culto ao Grande Sonhador! Salve Xululu! E agora, como seu primeiro projeto, deve provar seu valor, ou ser tomado pelo terror! Você deve criar um algoritmo capaz de calcular o total de caracteres corretamente tokenizados, considerando a ordem sequencial correta, ou seja, a sequência deve ser a correta, não necessariamente a posição. A tokenização busca segmentar os tokens de uma sentença, ou seja, separar as palavras individuais. Por exemplo, na sentença "Salve Xululu!" a tokenização deve separar a exclamação do nome próprio para gerar a string "Salve Xululu !". No momento você não terá que se preocupar em tratar todos os fenômenos linguísticos de todas as línguas, apenas deverá iniciar pela avaliação de textos já tokenizados.

Entrada

A primeira linha da entrada contém um inteiro N, o número de casos de teste. Cada caso teste é composto por duas linhas, cada uma com uma string S1 e S2 de no máximo 1000 caracteres. A primeira string será o resultado da tokenização gerada pela IA criada pelos súditos. Já a segunda string será a string tokenizada corretamente, dessa forma, se a tokenização estiver completamente correta, as duas strings devem ser iguais.

Saída

Para cada caso de teste, exiba uma linha com a quantidade de caracteres corretamente tokenizados atentando a sequência (repetições devem ser contadas).

se ele achar uma sequência de dois caracteres, os dois estão corretos

Restrições

$1 \leq N \leq 10$

Exemplo

Entrada	Saída
3 Xululu Xululu Salve Xululu! Salve Xululu ! abdxca dcab	6 13 3