Maratona de Programação FACENS 2018

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: SOMA DOS PRIMOS

Arquivo: `soma.[cpp/c/java]`

Cor: vermelho

Limite de tempo: 1s

Descrição do problema

Seu primo, Joãozinho, está com dificuldades para aprender a somar frações e pediu sua ajuda. Ele possui uma lista de exercícios gigantesca e você está sem paciência pois se lembra quando ele veio pedir ajuda para contar números primos.

Todos os exercícios que ele possui são no padrão a/b + c/d, sendo a, b, c e d números naturais maiores do que zero.

Ele quer a resposta da soma para saber se acertou o exercício.

Cuidado: ele deseja saber a resposta na forma simplificada, ou seja, se a soma resultar em 2/4, o valor exibido deve ser 1/2.

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro N que representa a quantidade de exercícios que Joãozinho deseja resolver.

As próximas N linhas contém quatro números naturais a, b, c e d.

Nota: é garantido que o MMC entre b e d é menor do que 3*10^18.

Saída

Para cada um dos exercícios, seu programa deve produzir uma linha no padrão e/f, onde "e" representa o numerador e "f" o denominador do resultado.

Exemplos de testes

Entrada	**Saída**
3 1 2 3 4 1 2 1 2 10 4 7 9	5/4 1/1 59/18

Restrições

10 ≤ N ≤ 10^5

1 ≤ a, b, c, d ≤ 10^12

PROBLEMA B: UMA REDE SOCIAL

Arquivo: `feicibuqui.[cpp/c/java]`

Cor: branco

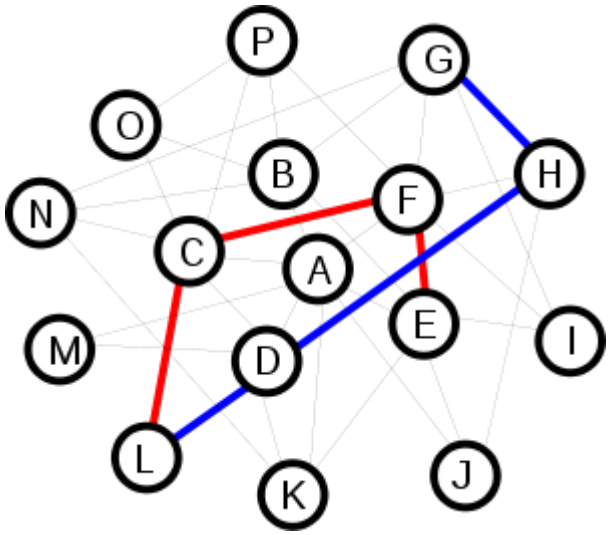
Limite de tempo: 1s

Descrição do problema

Com o crescimento explosivo do Feicibuqui, os administradores vislumbram validar a Teoria dos Seis Graus de Separação proposta por Stanley Milgram.

Essa teoria diz que são necessários no máximo seis laços de amizade para que duas pessoas quaisquer estejam ligadas.

A imagem abaixo exemplifica as relações de amizades entre diversas pessoas e as ligações entre os pares de pessoas (G; L) e (E; L).



Nesta imagem, as pessoas são representadas por letras e as relações de amizades por linhas.

Os desenvolvedores da rede social precisam de um mecanismo para avaliar se existe uma ligação para alguns pares de pessoas antes de testar tal teoria.

Para isso, eles precisam representar as pessoas por números e realizar dois tipos de operações:

- Criar uma relação de amizade entre um par de pessoas (criar um traço)
- Testar se existe uma ligação (sequências de amizades ligando-as) entre um par de pessoas

Você acha que consegue criar tal mecanismo?

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro N que representa a quantidade de testes que serão realizados com o seu programa.

A primeira linha de um caso de teste contém dois inteiros V e Q que representam a quantidade de pessoas e operações realizadas daquele teste, respectivamente.

Cada uma das próximas Q linhas de um caso de teste contém três inteiros A, B e C que representam a operação que será realizada (1 para criar uma relação de amizade e 2 para testar se existe uma ligação) e as pessoas (B e C) sobre as quais a operação será realizada, respectivamente.

Saída

Para cada operação do tipo 2 (testar se existe uma ligação entre duas pessoas), seu programa deve imprimir uma linha com S para caso exista a ligação ou N, caso contrário.

Exemplos de testes

Entrada	**Saída**
2 3 5 1 1 2 2 1 2 2 1 3 1 2 3 2 1 3 5 8 1 1 2 1 3 4 2 1 4 1 4 5 1 3 5 2 1 4 1 2 5 2 1 4	S N S N N S

Restrições

$10 \leq N \leq 100$

$10 \leq V, Q \leq 10^5$

$1 \leq A \leq 2$

$1 \leq B, C \leq V$

$B \neq C$

PROBLEMA C: FIGURINHAS

Arquivo: `figurinhas.[cpp/c/java]`

Cor: azul claro

Limite de tempo: 1s

Descrição do problema

Seu Severino estava um pouco nostagico. Lembrava muito de seu pai Zacarias que não está mais entre nós. Em sua época não tinha computadores pessoais. Ademais, que era isso? Somente brincava com as pedras que lhe ensinava muita coisa.

Com essa nostalgia seu Severino resolveu ver seu quartinho da bagunça e alí encontrou sua caixinha de figurinhas, um presente de seu pai. No entanto, nunca conseguiu completar sua coleção e nem lembrava mais de quais estavam faltando.

Agora seu Severino, já mais experiente, quer colecionar coisas como quando era criança. Assim, decidiu completar sua coleção de figurinhas e gostaria de saber quais figurinhas ele já possui para poder colocar em seu blog e poder procurar as faltantes na internet. Você, um grande amigo de seu Severino, decidiu ajudar-lo com a tarefa de criar um índice de figurinhas para seu blog. Como as figurinhas estão todas embaralhados, sua primeira tarefa é ordenar as figurinhas que seu Severino já possui.

Entrada

A entrada possui várias linhas. A primeira linha contém um inteiro Q que representa a quantidade de casos de teste. Cada caso de teste é constituído por duas linhas. A primeira linha contém um inteiro N que representa a quantidade de figurinhas. A segunda linha de um caso de teste contém N inteiros. Cada inteiro M desses N reprenseta uma figurinha que seu Severino já possui.

Saída

Para cada caso de teste, você deve imprimir uma linha com todas as figurinhas que seu Severino possui em ordem crescente separadas por um espaço.

Exemplos de Testes

Entrada	**Saída**
3 4 4 7 8 6 5 4 6 7 3 10 5 2 3 8 1 10	4 6 7 8 3 4 6 7 10 1 2 3 8 10

Restrições

$0 < M, N \leq 10^7$

$0 < Q \leq 20$

PROBLEMA D: NEYMAR, NEYMAR, NEYMAR...

Arquivo: `neymar.[cpp/c/java]`

Cor: laranja

Limite de tempo: 1s

Descrição do problema

Depois do fracasso na copa do mundo da Rússia (2018) e ficar fora da lista dos melhores do mundo pela Fifa, Neymar precisava encontrar uma maneira de parar de pensar em futebol temporariamente. Conversando com seus amigos, o craque ficou sabendo que havia um novo jogo na cidade de Paris, um jogo conhecido como “Somou? Ganhou!”. Ao procurar as regras do seu possível novo hobby, Neymar se deparou com o seguinte tutorial:

Somou? Ganhou! – Como jogar:

Dado um conjunto de X elementos, inicialmente todos com valor zero, o desafiante deve ficar atento aos Y comandos que serão ditos pelo mestre. Os comandos seguem o seguinte formato:

• **0 l r v** – o desafiante tem que adicionar v para todos os números na faixa de l à r (inclusive), onde l e r representam os índices da posição dos elementos no conjunto.

• **1 l r** – o desafiante deve dizer em voz alta a soma do valor dos elementos do conjunto entre os índices l e r (inclusive), se a resposta for correta o participante ganha 1000 euros, se for errada paga 100 euros ao mestre.

Apesar de parecer fácil, nesse jogo o desafiante não pode utilizar nada além de seu cérebro, nem mesmo papel e caneta, todas as respostas devem ser calculadas de cabeça. Com isso em mente, Neymar decidiu praticar bastante de sua primeira tentativa, então contratou você, um programador experiente, para desenvolver um software de treino para essa competição.

Entrada

A primeira linha contém um número inteiro T, que indica o número de casos de teste. Cada caso de teste começará com X e Y. Então, você deverá ler Y operações no formato descrito acima.

Saída

Para cada comando 1 l r você deve imprimir uma linha com o único inteiro contendo o valor que Neymar deveria dizer para ganhar os 1000 euros.

Exemplos de testes

Entrada	**Saída**
1 10 5 0 3 7 30 1 3 4 1 2 3 0 1 5 15 1 5 5	60 30 45

Restrições

$1 \leq X, Y \leq 10^5$

$1 \leq l, r \leq X$

$1 \leq v \leq 10^7$

PROBLEMA E: JOGUE A VELHA

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

Cor: rosa

Limite de tempo: 1s

Descrição do problema

O jogo da velha era um passatempo popular entre os estudantes entediados antes do surgimento do smartphone.

Ele é jogado por 2 pessoas e consiste de um tabuleiro 3 linhas e 3 colunas de espaços nos quais os jogadores podem colocar, de forma alternada, sua respectiva marca (normalmente a letra ‘x’ ou a letra ‘o’ maiúsculas).

O primeiro jogador que completar uma linha, coluna ou diagonal ganha o jogo. Se acabarem os espaços e nenhum jogador ganhar, significa que eles empataram.

A imagem abaixo ilustra todos os estados de uma sequência de jogadas.

Crie um programa que, dado o estado do jogado, exibe se algum jogador ganhou, se eles empataram ou se devem continuar jogando.

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro N que indica a quantidade de casos de testes. Cada caso de teste é composto por 5 linhas. A segunda e quarta linhas contém 5 traços ('-'). A primeira, terceira e última linhas contém os valores dos espaços do tabuleiro separados por uma barra vertical ('|'). Quando um espaço do tabuleiro está livre, ele é representado por um traço (‘-‘).

Não se preocupe, não há situações inválidas; por exemplo, dois jogadores em situação de vitória.

Saída

Para caso de teste, exiba uma das seguintes mensagens:

- 'Continue jogando!' se for possível continuar jogando;
- 'X ganhou!' se o jogador que usa o ‘X’ ganhou;
- 'O ganhou!' se o jogador que usa o ‘O’ ganhou;
- 'Empatou!' se eles empataram.

Exemplos de testes

Entrada	**Saída**
5 -|-|- ----- -|-|- ----- -|-|- X|O|X ----- O|X|O ----- X|-|- X|O|X ----- O|X|O ----- -|X|- X|O|X ----- O|O|X ----- X|O|- X|O|X ----- O|X|X ----- O|X|O	Continue jogando! X ganhou! Continue jogando! O ganhou! Empatou!

Restrições

1 ≤ N ≤ 10^6

PROBLEMA F: UM PASSEIO PELO BAIRRO

Arquivo: `casas.[cpp/c/java]`

Cor: azul escuro

Limite de tempo: 1s

Descrição do problema

Existem N casas em sequência. Elas estão numeradas de 1 a N da esquerda pra direita. Você começa na casa 1.

Você tem que fazer K movimentos para outras casas. Em um movimento você pode ir da casa que está para qualquer outra casa. Você não pode ficar na mesma casa, ou seja, em cada movimento você deve ir para uma casa diferente da sua. Se você for da casa X para a casa Y, a distância total percorrida incrementa em |X-Y| unidades de distância, onde |A| é o valor absoluto de A. É possível visitar a mesma casa quantas vezes quiser.

O seu desafio é andar exatamente S unidades de distância no total.

Se não for possível andar S unidades de distância no total, exiba "N". Caso contrário exiba "S".

Lembrando que você deve realizar exatamente K movimentos.

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro Q que representa a quantidade de casos de teste.

Cada caso de teste é composto por uma linha contendo os inteiros N, K e S.

Saída

Para caso de teste, imprima uma linha contendo 'S' se for possível realizar K movimentos que totalizam S unidades de distância ou 'N'; caso contrário.

Exemplos de testes

Entrada	Saída
4 10 2 15 10 9 45 10 9 81 10 9 82	S S S N

Restrições

1 ≤ N, K, S ≤ 10^3

1 ≤ Q ≤ 100

PROBLEMA G: PANQUECAS FELIZES

Arquivo: `panquecas.[cpp/c/java]`

Cor: verde claro

Limite de tempo: 1s

Descrição do problema

A Casa das Panquecas Infinitas acabou de criar um novo tipo de panqueca! Ela tem uma rosto sorrindo feito de chocolate de um lado (lado feliz) e nada do outro lado (lado vazio).

Você é o garçom chefe e a cozinha acabou de te entregar uma pilha de panquecas para servir aos clientes. Como qualquer outro bom garçom, você tem visão de raio-X e pode ver quais panquecas tem o lado feliz voltado pra cima. Você acha que os clientes ficarão mais felizes se todas as panquecas estiverem com o lado feliz virado pra cima.

Você consegue fazer a seguinte operação: cuidadosamente levantar um certo número de panquencas (até mesmo todas) a partir do topo, virar o grupo inteiro e então colocar novamente em cima da pilha que não foi levantada. Ao virar um grupo de panquecas, você vira o grupo em um movimento único; você não as vira individualmente. Formalmente: se numerarmos as panquecas 1, 2, ..., N a partir do topo, você escolhe as primeiras i panquecas para virar. Então, após virar, a pilha fica i, i-1, ..., 2, 1, i+1, i+2, ..., N. Panquecas 1, 2, ..., i agora têm o lado oposto virado para cima, enquanto as panquecas i+1, i+2, ..., N têm o mesmo lado virado pra cima que tinham antes da operação ser realizada.

Por exemplo, vamos representar o lado feliz como '+' e o lado vazio como '-'. Imagine que a pilha, a partir do topo, seja --+-. Uma maneira válida de realizar a operação seria pegar as três primeiras panquecas, virá-las e coloca-las em cima da quarta panqueca restante (que não sofreria alteração). A pilha ficaria então -++-. Outras maneiras válidas seriam virar apenas a primeira, as duas primeiras ou todas. Não é válido apenas as duas dois meio ou a de baixo, por exemplo; você só pode pegar as do topo.

Você não vai servir o cliente até que todas as panquecas estejam com o lado feliz virado pra cima, porém você não quer que as panquecas esfriem, então você tem que ser rápido! Qual o menor número de vezes que você precisará realizar a operação?

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro N com a quantidade de casos de teste.

Cada caso de teste consiste de um linha com um texto S; cada caracter de S é um '+' ou um '-'. Quando lida da esquerda pra direita, representa a pilha vista de cima pra baixo. O tamanho de S será menor ou igual a 100.

Saída

Para cada caso de teste, exiba uma linha com a quantidade mínima de operações necessárias.

Exemplos de testes

Entrada	**Saída**
5 - -+ +- +++ --+-	1 1 2 0 3

No primeiro caso só é necessário virar a primeira (e única) panqueca.

No segundo caso só é necessário virar a primeira panqueca.

No terceiro caso é necessário realizar a operação duas vezes. Uma maneira seria virar a primeira panqueca, mudando a pilha para -- e então realizar a operação novamente com todas as panquecas.

No quarto caso não é necessário realizar a operação.

No quinto caso você pode virar a pilha inteira, tornado-a +-++. Depois virar a primeira panqueca, ficando --++ e então virar as duas primeiras panquecas.

Restrições

$1 \leq N \leq 100$

$1 \leq |S| \leq 100$

PROBLEMA H: MARIAZINHA APRENDE A CONTAR

Arquivo: `proximo.[cpp/c/java]`

Cor: preto

Limite de tempo: 1s

Descrição do problema

Sua prima, Mariazinha, está aprendendo a contar.

Ela já sabe usar o computador e pediu a você para criar um programa que a ajude a saber se ela está acertando os exercícios que a professora passou.

O seu programa deve receber a quantidade de exercícios e, para cada exercício, receber um valor inteiro e exibir o próximo.

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro N que indica a quantidade de casos de teste. Cada caso de teste é composto por uma linha com um inteiro X.

Saída

Para cada caso de teste, seu programa deve exibir um linha com o número seguinte a X no conjunto dos naturais.

Exemplos de testes

Entrada	Saída
5 100 2 35 47 58	101 3 36 48 59

Restrições

$1 \leq N, X \leq 100$

PROBLEMA I: BLITZ DAS COMANDAS

Arquivo: `comandas.[cpp/c/java]`

Cor: marrom

Limite de tempo: 1s

Descrição do problema

Você já deve ter ido a uma padaria moderna. Dessas que utilizam comandas para controlar suas compras. Essas comandas muitas vezes são cartões plasticos com números normalmente com 3 ou mais dígitos. Quando você pega algum produto fresco como pães, frios ou doces, caso a leitura não seja automática, o atendente digita os números da sua comanda em um sistema para registrar seus pedidos.

O ser humano não é a prova de falhas, e, eventualmente o atendente pode errar na digitação. Se houver 2 comandas com números semelhantes, onde apenas 1 dígito é diferente, o seu pedido pode ser lançado na comanda de outro cliente. Dependendo da sua idoneidade, você pode até gostar dessa ideia. Mas pense que pode acontecer o inverso. Algum cliente pede 1 kg de queijo roquefort e acidentalmente o atendente lança na sua comanda. Aí complica.

Por esse motivo, o seu desafio é checar as comandas de uma padaria para tentar checar se existe essa possibilidade. Vamos levar em consideração que um raio não cai 2 vezes no mesmo lugar (o que já está provado que pode acontecer, mas vamos ignorar isso no momento) e que o atendente nunca erra 2 dígitos na digitação de uma mesma comanda. Dada uma sequência de números de comandas, seu programa deve buscar números que podem ser confundidos, isto é, que tenham menos que 2 dígitos diferentes. Também é importante que cada número de comanda não possua dígitos repetidos.

Entrada

A entrada é composta por várias linhas. A primeira linha contém um inteiro Q que representa a quantidade de casos de testes. A primeira linha de um caso de teste contém 2 números inteiros separados por espaço, M e N. M indica o número de dígitos das comandas e N, o número de comandas que serão informadas. Na sequência, haverá N linhas, cada uma com um inteiro de M dígitos (nunca iniciando com zero) que representa uma das comandas.

Saída

Para cada caso de teste exiba "OK" caso não haja um par de comandas com menos de 2 dígitos em comum na mesma posição e nenhuma delas tenha dígitos repetidos. Caso contrário, exiba "FALHA".

Exemplos de testes

Entrada	**Saída**
3 4 5 1234 1324 1243 1423 2345 6 3 135790 134790 862973 3 4 123 456 789 899	OK FALHA FALHA

Restrições

0 < Q < 100

2 < N < 1000

2 < M < 11