Maratona de Programação FACENS 2019

CADERNO DE PROBLEMAS

Informações gerais

Este caderno contém 8 problemas; as páginas estão numeradas de 1 a 22.

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: POTÊNCIA

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

Cor: branco

Limite de tempo: 1s

Descrição do problema

Faltou tempo para a história então vamos começar com um simples: calcular e exibir a potência A elevado a B.

Entrada

A entrada será composta por vários casos de testes.

A primeira linha da entrada contém um inteiro N; a quantidade de casos de testes.

Cada caso de teste é composto por uma linha com os valores de A e B separados por um espaço.

Saída

Para cada caso de teste, exiba uma linha, o resultado de A elevado a B módulo (10^100 + 13).

Exemplos de testes

Entrada	**Saída**
3 2 3 3 4 3 2	8 81 9

Restrições

1 ≤ N ≤ 10

1 ≤ A, B ≤ 10^100

PROBLEMA B: IMPLEMENTAÇÃO DO RAID 5

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

Cor: amarelo

Limite de tempo: 1s

Descrição do problema

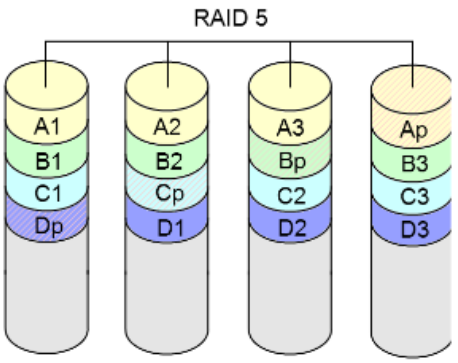
RAID (Redundant Array of Inexpensive Disks - Conjunto Redundante de Discos Econômicos) é um meio de se criar um sub-sistema de armazenamento composto por vários discos individuais com a finalidade de ganhar segurança e desempenho. RAID seriam compostos por dois ou mais discos trabalhando simultaneamente para um mesmo fim.

Há várias implementações possíveis de RAID dividida em alguns modos:

- **RAID 0** - É uma simples concatenação de partições para criar uma grande partição virtual.
- **RAID 1** - é o nível de RAID que implementa o espelhamento de disco, também conhecido como *mirror*.
- **RAID 2** – trabalha com controle de erro ECC
- **RAID 3** – é uma versão simplificada do RAID nível 2.
- **RAID 4** – funciona com dois ou mais discos iguais. Um dos discos guarda a paridade (uma forma de soma de segurança) da informação contida nos discos.
- **RAID 5** – funciona similarmente ao RAID 4, mas supera alguns dos problemas mais comuns sofridos por esse tipo. As informações sobre paridade para os dados do conjunto são distribuídas ao longo de todos os discos do conjunto, ao invés de serem armazenadas num disco dedicado, oferecendo assim mais desempenho que o RAID 4, e, simultaneamente, tolerância a falhas.

Uma empresa precisa elaborar o *software* que vai operar em uma controladora de discos com suporte a RAID 5. Para isso, a princípio você deve elaborar um algoritmo simplificado que será utilizado em simulações e testes.

O funcionamento do RAID 5 é o seguinte. Para cada faixa de dados, um disco é usado para armazenar a paridade calculada para aquela faixa, como mostra a figura.



Na primeira faixa de dados (A), o quarto disco é usado para armazenar a paridade .
Na segunda faixa (B), o terceiro disco é usado para paridade, e assim sucessivamente

Para a simulação, cada bloco de dados será composta de 4 bits, e o conjunto será composto de 5 discos.

Para calcular o bloco de paridade, segue o exemplo:

Dados a serem gravados em hexadecimal: A2B7 (cada algarismo corresponde a um bloco de um disco)

Bloco 1: $A_{(16)} = 1010_{(2)}$
Bloco 2: $2_{(16)} = 0010_{(2)}$
Bloco 3: $B_{(16)} = 1011_{(2)}$
Bloco 4: $7_{(16)} = 0111_{(2)}$

Para cada conjunto de bits, a paridade é obtida da seguinte forma. Somam-se os bits. Se o resultado é PAR, a paridade é 0. Se o resultado é ÍMPAR, a paridade é 1.

```
1010
0010
1011
0111
————
0100 ← Bloco de paridade para a faixa específica.
```

$0100_{(2)} = 4_{(16)}$

Portanto, para o conjunto de blocos **A2B7**, o bloco de paridade é **4**.

Bloco p: $4_{(16)} = 0100_{(2)}$

No simulador, serão considerados 5 discos e 16 faixas de gravação. A tabela a seguir demostra a posição dos blocos de paridade (P) para cada faixa:

Discos	**0**	**1**	**2**	**3**	**4**
Faixa 0					P
Faixa 1				P	
Faixa 2			P		
Faixa 3		P			
Faixa 4	P				
Faixa 5					P
Faixa 6				P	
Faixa 7			P		
Faixa 8		P			
Faixa 9	P				
Faixa 10					P
Faixa 11				P	
Faixa 12			P		
Faixa 13		P			
Faixa 14	P				
Faixa 15					P

Tabela1: Posição dos Blocos de Paridade

Esse simulador aceitara vários comandos de entrada, tais como escrita de dados (W), leitura de dados (R), limpeza de bloco (C), saída (Q) e importação (I) que simula a existência de dados nos discos. Inicialmente, considere que todas as faixas estão livres, isto é, todos os discos estão vazios (preenchidos com 0).

Teremos as operações:

Operação de importação (I): recebe o valor dos 5 blocos e grava na próxima faixa livre, marcando a mesma como ocupada, sem se preocupar com paridade. Retorna a posição gravada e o valor de cada bloco na ordem ou 'CHEIO' no caso de todas as faixas estarem ocupadas:

Ex: `Entrada: IA2B27 Saída: 01A2B27`

Operação de gravação (W): recebe o valor dos 4 blocos de dados, calcula a paridade e grava os dados na próxima faixa livre, marcando a mesma como ocupada, nos discos corretos, de acordo com a posição da faixa. Retorna a posição gravada e o valor de cada bloco na ordem ou 'CHEIO' no caso de todas as faixas estarem ocupadas:

Ex: `Entrada: WA2B7 Saída: 02A24B7`

Operação de leitura (R): recebe o número da faixa a ser lida, faz a leitura e verifica se a paridade está correta. Retorna os blocos lidos. Caso a paridade esteja errada, apresenta a palavra 'FALHA' na frente dos dados.

Ex: `Entrada: R02 Saída: A2B7`

Operação de limpeza (C): recebe o número da faixa a ser limpa, zera todos os discos e marca a faixa como livre.

Ex: `Entrada: C01 Saída: 0100000`

Operação de saída (Q): encerra a execução do programa.

Ex: `Entrada: Q Saída:`

Entrada

A entrada é sempre composta de uma letra maiúscula que representa a operação, seguida do parâmetro esperado por essa operação. Para a operação de importação, o parâmetro é composto de 5 dígitos hexadecimais. Para a operação de escrita, o parâmetro é composto de 4 dígitos hexadecimais. Para as operações de leitura e limpeza, o parâmetro é composto de 2 dígitos decimais. Para a operação de saída, não há parãmetros.

Saída

Para as operações de importação (I) e escrita (W), a saída é composta de 2 dígitos decimais (faixa) e 5 hexadecimais (blocos) ou 'CHEIO' no caso do disco estar cheio. Para a operação de leitura (R) a saída é composta de 4 dígitos hexadecimais e, no caso de erro de paridade, seguido da palavra 'FALHA'. Para a operação de limpeza (C), a saída é composta de 2 dígitos decimais (faixa) e 5 dígitos zeros. Para a operação de saída (Q), não há saídas

Exemplos de testes

Entrada	**Saída**
I12344	0012344
I63207	0163207
IFA3CA	02FA3CA
IA3AAA	03A3AAA
I0BBBB	040BBBB
ICCCC0	05CCCC0
IDDD0D	06DDD0D
IEE2EE	07EE2EE
IF0FFF	08F0FFF
I0ABCD	090ABCD
I8777F	108777F
I98765	1198765
IAB6BC	12AB6BC
IC0DBA	13C0DBA
I11101	1411101
I20222	1520222
I00000	CHEIO
R00	1234
R03	AAAA-FALHA
R05	CCCC
R07	EEEE-FALHA
R09	ABCD
R11	9875-FALHA
C05	0500000
C02	0200000
C07	0700000
R05	0000
R07	0000
WCAC0	02CAAC0
W4359	054359B
WF123	07F1F23
W0201	CHEIO
C10	1000000
W0201	1002013
W1312	CHEIO
R02	CAC0
R05	4359
R07	F123
Q	

PROBLEMA C: CONSTRUINDO PALÍNDROMOS

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

Cor: rosa

Limite de tempo: 1s

Descrição do problema

Anna possui uma linha de N blocos, cada um com uma letra de A a Z escrita nele. Os blocos são numerados 1, 2, ..., N da esquerda para a direita.

Hoje, ela está aprendendo palíndromos. Um palíndromo é um texto que não muda se você o lê da esquerda para a direita ou da direita para a esquerda. Por exemplo, ANNA, ARARA, AAA e X são todos palíndromos, enquanto AB, SAPO e IOIO não são.

Bob quer testar o quão bem Anna conhece palíndromos e, para isso, vai realizar Q perguntas. A i-ésima pergunta é: é possível Anna criar um palíndromo usando todos os blocos no intervalo Li a Ri, podendo rearranjá-los? Após uma pergunta, Anna volta os blocos em suas posições originais.

Ajude Anna a descobrir quantas perguntas de Bob ela consegue responder "sim".

Entrada

A entrada será composta por vários casos de testes.

A primeira linha da entrada contém um inteiro T; a quantidade de casos de testes.

Cada caso de teste é composto por várias linhas. A primeira linha de um caso de teste contém dois inteiros N e Q, o número de blocos e a quantidade de perguntas, respectivamente. A segunda linha de um caso de teste contém uma string de tamanho N contendo letras de A-Z (todas maiúsculas) representando as letras dos blocos na ordem em que eles aparecem. As próximas Q linhas contém as perguntas de Bob. A i-ésima linha contém os valores Li e Ri descrevendo a i-ésima pergunta.

Saída

Para cada caso de teste, exiba uma linha, a quantidade de perguntas de Bob para as quais Anna responde "sim".

Exemplos de Testes

Entrada	**Saída**
2 7 5 ABAACCA 3 6 4 4 2 5 6 7 3 7 3 5 XYZ 1 3 1 3 1 3 1 3 1 3	3 0

Para o primeiro caso de teste:

Na primeira pergunta, Anna deve usar o bloco AACC. Ela pode rearranjar os blocos em um palíndromo ACCA ou CAAC.

Na segunda pergunta, Anna deve usar o bloco A que já é um palíndromo.

Na terceira pergunta, Anna deve usar o bloco BAAC; não é possível criar um palíndromo com esses blocos.

Na quarta pergunta, Anna deve usar o bloco CA; também não é possível formar um palíndromo.

Na quinta e última pergunta do primeiro caso, Anna deve usar o bloco AACCA. Ela pode obter os palíndromos ACACA e CAAAC.

Para o segundo caso de teste, ela não consegue criar um palíndromo com os blocos XYZ; note que todas as perguntas de Bob são iguais.

Restrições

1 ≤ T ≤ 100

1 ≤ N, Q, Li, Ri ≤ 10^5

PROBLEMA D: ALCANCE OS NÚMEROS

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

Cor: marrom

Limite de tempo: 1s

Descrição do problema

Seja f(x) uma função que soma 1 a x, então, enquanto houver um 0 a direita no número resultante, remove esse 0. Por exemplo:

f(599) -> 600 -> 60 -> 6

f(7) -> 8

f(9) -> 10 -> 1

f(10099) -> 10100 -> 1010 -> 101

Um número y é dito alcançável a partir de x se for possível aplicar f a x uma quantidade de vezes (possivelmente zero) até que y seja obtido.

Por exemplo, 102 é alcançável a partir de 10098 pois f(f(f(10098))) = f(f(10099)) = f(101) = 102; e qualquer número é alcançável a partir dele próprio.

Dado um número N, conte quantos números são alcançáveis a partir de N.

Entrada

A entrada será composta por vários casos de testes. A primeira linha da entrada contém um inteiro T; a quantidade de casos de testes.

Cada caso de teste é composto por uma linha com o valor de N.

Saída

Para cada caso de teste, exiba uma linha, a quantidade de números alcançáveis a partir de N.

Exemplos de testes

Entrada	**Saída**
2 1098 10	20 19

Números alcançáveis a partir de 1098:

1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1098, 1099.

Restrições

1 ≤ T ≤ 200

1 ≤ N ≤ 10^9

PROBLEMA E: CRESCENTE

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

Cor: verde

Limite de tempo: 1s

Descrição do problema

Dada uma lista de N inteiros, você deve realizar uma sequência de movimentos. A cada movimento, você deve pegar e remover o elemento mais à esquerda ou mais à direita da lista de inteiros.

A sua tarefa é obter o tamanho da maior sequência estritamente crescente possível realizando apenas esses dois tipos de movimentos.

Por exemplo, para a lista A = [1, 2, 4, 3, 2] a resposta é 4:

- pegar o 1 e a lista se torna [2, 4, 3, 2]
- pegar o 2 a direita resultando em [2, 4, 3]
- pegar o 3 sobrando [2, 4]
- pegar o 4 restando [2]

A sequência obtida [1, 2, 3, 4] é estritamente crescente e possui o maior tamanho possível dentre todas as sequências que poderiam ser obtidas realizando esses tipos de movimentos.

Entrada

A entrada será composta por vários casos de testes. A primeira linha da entrada contém um inteiro T; a quantidade de casos de testes.

Cada caso de teste é composto por duas linhas. A primeira linha de um caso de teste contém um inteiro N, a quantidade de elementos da lista. A segunda linha de um caso de teste contém N inteiros Ai separados por espaço; Ai representa o i-ésimo elemento da lista.

Saída

Para cada caso de teste, exiba uma linha com o tamanho da maior sequência que pode ser obtida.

Exemplos de testes

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

Restrições

1 ≤ N ≤ 500

1 ≤ T, Ai ≤ 20.000

PROBLEMA F: ENGRENAGENS

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

Cor: azul claro

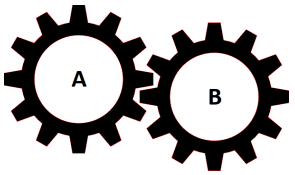
Limite de tempo: 1s

Descrição do problema

Elementos mecânicos que funcionam em pares, as engrenagens permitem a criação dos mais diversos tipos de mecanismos.

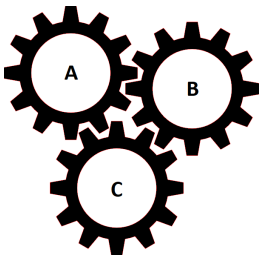
Elas podem variar em tamanho, quantidade de dentes, distância entre dentes e tipos. Existem diversos tipos: cônicas, helicoidais, hipoides, cremalheira, retas etc. Esse ano focaremos nas mais comuns, as retas.

A imagem abaixo mostra duas engrenagens engatadas. Quando a engrenagem A gira no sentido horário, a engrenagem B gira no sentido anti-horário, obrigatoriamente. Se o movimento de uma engrenagem é invertido, o movimento da outra também é.

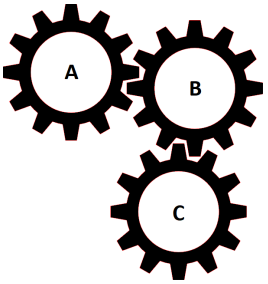


Ao adicionarmos uma terceira engrenagem, temos que tomar cuidado pois, se engatadas da maneira errada, pode impossibilitar o movimento do sistema inteiro.

A figura abaixo ilustra bem essa situação. Se tentarmos mover a engrenagem A no sentido horário, então B e C serão forçadas a girar no sentido anti-horário, porém isso é impossível, uma vez que B e C também estão engatadas e devem ter movimentos em sentidos opostos.



Para resolver esse problema, basta desengatar as engrenagens B e C que o sistema volta a funcionar; como mostra a figura abaixo.



Você resolveu criar um programa para, dado um conjunto de engrenagens e seus engates, avaliar se o sistema funciona (gira) ou não.

Entrada

A entrada será composta por vários casos de testes. A primeira linha da entrada contém um inteiro T; a quantidade de casos de testes.

Cada caso de teste é composto por diversas linhas. A primeira linha de um caso de teste contém a quantidade N de engrenagens; cada engrenagem é representada por um inteiro A entre 1 e N.

As próximas N linhas descrevem as engrenagens. A i-ésima linha descreve a engrenagem i. A descrição de uma engrenagem A começa com um inteiro Q que representa a quantidade de engrenagens com as quais A está engatada. Em seguida, são dados Q inteiros que representam as engrenagens com as quais A está engatada.

Saída

Para cada caso de teste, exiba uma linha com a palavra "sim" se for possível que o sistema funcione; imprima "nao" caso contrário.

Exemplos de testes

Entrada	Saída
3 2 1 2 1 1 3 2 2 3 2 1 3 2 1 2 3 1 2 2 1 3 1 2	sim nao sim

Os exemplos acima descrevem as imagens na mesma ordem em que aparecem.

Restrições

1 ≤ T, N, Q ≤ 200

PROBLEMA G: TIPO SANGUÍNEO

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

Cor: preto

Limite de tempo: 1s

Descrição do problema

Você foi contratado para criar um programa que, dado os tipos sanguíneos de um doador e um receptor, dizer se é possível realizar a doação sanguínea ou não.

A imagem abaixo mostra todas as relações de possibilidade ou não de doação.

Receptor	Doador							
	O−	O+	A−	A+	B−	B+	AB−	AB+
O−	✓	✗	✗	✗	✗	✗	✗	✗
O+	✓	✓	✗	✗	✗	✗	✗	✗
A−	✓	✗	✓	✗	✗	✗	✗	✗
A+	✓	✓	✓	✓	✗	✗	✗	✗
B−	✓	✗	✗	✗	✓	✗	✗	✗
B+	✓	✓	✗	✗	✓	✓	✗	✗
AB−	✓	✗	✓	✗	✓	✗	✓	✗
AB+	✓	✓	✓	✓	✓	✓	✓	✓

Note que o tipo O- pode doar para todos os outros tipos mas só pode receber dele mesmo. O oposto ocorre para o tipo AB+.

Entrada

A entrada será composta por vários casos de testes.

A primeira linha da entrada contém um inteiro T; a quantidade de casos de testes.

Cada caso de teste é composto por uma linha com dois textos D e R que representam os tipos sanguíneos do doador e receptor, respectivamente.

Cada tipo sanguíneo pode ser O+, O-, A+, A-, B+, B-, AB+ ou AB-.

Saída

Para cada caso de teste, exiba uma linha com o texto "sim" se D pode doar para R; caso contrário exiba "nao".

Exemplos de testes

Entrada	**Saída**
3 O- O+ A+ AB+ AB+ A+	sim sim nao

Restrições

1 ≤ T ≤ 64

PROBLEMA H: GOT

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

Cor: vermelho

Limite de tempo: 1s

Descrição do problema

ALERTA DE SPOILER: VÁ PARA A PRÓXIMA PÁGINA SE NÃO QUISER SPOILER DE GAME OF THRONES

"One dragon in recorded history has been brought down from the air by a projectile, and it was a one in a million shot. A mature dragon in the air is virtually invulnerable" -George R.R. Martin

Tradução livre: [Apenas] Um dragão na história foi abatido no ar por um projétil, e foi um tiro com uma chance de um em um milhão. Um dragão maduro no ar é virtualmente invulnerável.

Claramente, os tiros que Euron acertou em Rhaegal no episódio cinco são completamente contraditórias à descrição do autor dos livros que originaram a série. Infelizmente, esse não foi o único problema da série nessas últimas temporadas.

Como um tributo à série, a dificuldade do exercício será tão decepcionante quanto esses episódios foram para mim.

Vamos calcular, de uma maneira bem simples, a distância que as flechas percorreram até o dragão. Serão dados as distâncias vertical e horizontal entre Euron (E) e o dragão (D), você deve calcular a distância em linha reta até a fera.

Você deve criar um programa que recebe as distâncias horizontal e vertical entre E e D, calcular e exibir a distância em linha reta entre eles.

Entrada

A entrada será composta por vários casos de testes.

A primeira linha da entrada contém um inteiro T; a quantidade de casos de testes.

Cada caso de teste é composto por uma linha com dois inteiro H e V que representam as distâncias horizontal e vertical, respectivamente.

Saída

Para cada caso de teste, seu programa deve exibir um linha com um inteiro que representa a distância entre E e D. Será garantido que a distância entre E e D é um inteiro.

Exemplos de testes

Entrada	Saída
3 3 4 5 12 6 8	5 13 10

Restrições

1 ≤ T, H, V ≤ 100