Maratona de Programação FACENS 2023

CADERNO DE PROBLEMAS

Informações gerais

Este caderno contém 11 problemas; as páginas estão numeradas de 1 a 24, 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: Aprendendo a Somar

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

Miguel está aprendendo a somar! Seu pai pensou em um jogo simples para ensiná-lo a somar utilizando um dado com 4 lados (um dado que possui os valores 1, 2, 3 e 4). Nesse jogo, os jogadores compartilham um valor total que começa em zero. A vez de um jogador consiste em rolar o dado e somar o valor sorteado ao total compartilhado. O jogo continua até que um dos jogadores erre (ou se distraia com alguma outra coisa).

Exemplo:
Total: 0
Miguel joga o dado e sorteia o 3. Novo total = 0 + 3 = 3
Seu pai sorteia o 2. Novo total = 3 + 2 = 5
Miguel sorteia o 1. Novo total = 5 + 1 = 6
Seu pai sorteia o 4. Novo total = 6 + 4 = 10
Miguel se distrai. Fim de jogo.

Uma vez finalizado o jogo, seu pai, um tanto nerd, resolve calcular de quantas maneiras possíveis eles poderiam ter chegado àquele resultado. Por exemplo, se o resultado final foi 3, então eles poderiam ter obtido esse resultado de 4 maneiras diferentes:

- 1 1 1
- 2 1
- 1 2
- 3

Eles poderiam ter obtido 4 de 8 maneiras diferentes:

- 1 1 1 1
- 2 1 1
- 1 2 1
- 3 1
- 1 1 2
- 2 2
- 1 3
- 4

Como a quantidade de maneiras diferentes que eles podem chegar a um resultado cresce muito rápido, o pai de Miguel pediu para você criar um programa para verificar se ele está correto. Novamente, esse valor cresce muito rápido, então mostrar essa quantidade mod 1.000.000.013 é suficiente. Um exemplo simples é que existem 2.456.659.116 maneiras de chegar a 100. Nesse caso, a resposta esperada será 456.659.090 (2.456.659.116 % 1.000.000.013).

Continua no verso >

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 total obtido ao final do jogo.

Saída

Para cada caso de teste, exiba uma linha contendo a quantidade de maneiras (mod 1.000.000.013) de chegar a T.

Restrições

$1 \leq N \leq 100$
$1 \leq T \leq 1.000.000.000$

Exemplo

Entrada	Saída
4 1 3 4 100	1 4 8 456659090

B: Eu sou Groot

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

Groot conseguiu um trabalho nas horas vagas como manobrista de naves em Luganenhum enfileirando as naves espaciais e caças estelares dos mercadores que ali chegam.

A árvore curiosa que é, Groot quer saber de quantas maneiras poderia enfileirar esses veículos. O Grande Colecionador monopoliza a venda das naves e caças, então todos são do mesmo modelo. As naves têm 10 metros de comprimento e os caças 5 metros, cada um vendido em diversas cores.

Dado o tamanho da fila em metros, a quantidade de cores de naves e a quantidade de cores dos caças, crie um programa para ajudar Groot a calcular de quantas maneiras ele pode formar uma fila desses veículos.
Por exemplo, se as naves são vendidas em 3 cores (A, B e C) e os caças em 2 cores (D e E), e Groot deseja formar uma fila 15 metros, então ele pode fazer isso de 20 maneiras diferentes:

- A D
- A E
- B D
- B E
- C D
- C E
- D A
- E A
- D B
- E B
- D C
- E C
- D D D
- D D E
- D E D
- E D D
- E E E
- E E D
- E D E
- D E E

Continua no verso >

Entrada

A entrada contém um inteiro T que representa a quantidade de casos de teste. Cada caso de teste é composto por apenas uma linha, contendo três inteiros N, K e L, separados por espaço. O inteiro N representa o comprimento total, em metros, da fila que Groot está considerando. K e L representam o número de cores distintas disponíveis para caças estelares e naves espaciais, respectivamente. Note que, como os inteiros N, K e L podem ser muito grandes, recomenda-se o uso de inteiros de 64 bits.

Saída

Para cada caso de teste, exiba uma linha contendo o resultado. Como o número de formas diferentes de se formar a fila pode ser muito grande, Groot está interessado nos últimos 6 dígitos da quantidade. Assim, para cada caso de teste, seu programa deve produzir exatamente 6 dígitos, correspondentes aos últimos dígitos da solução.

Restrições

$1 \leq T \leq 40$
$5 \leq N \leq 10^{15}$ e N é múltiplo de 5
$1 \leq K \leq 10^{15}$
$1 \leq L \leq 10^{15}$

Exemplo

Entrada	Saída
4 25 5 5 5 1000 1000 20 17 31 15 9 2	006000 001000 111359 000765

C: Vetores Contrastantes

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

Para um vetor de números inteiros $[a_1, a_2, ..., a_n]$, vamos chamar o valor $|a_1 - a_2| + |a_2 - a_3| + ... + |a_{n-1} - a_n|$ de contraste do vetor. Note que o contraste de um vetor de tamanho 1 é igual a 0.

Você receberá um conjunto de números inteiros a. Sua tarefa é construir um vetor b de forma de que todas as condições a seguir sejam satisfeitas:

- b não está vazio, ou seja, existe pelo menos um elemento;
- b é uma subsequência de a, ou seja, b pode ser produzido eliminando alguns elementos de a (ou zero elementos);
- O contraste de b é igual ao contraste de a.

Entrada

A primeira linha contém um único número inteiro T - o número de casos de teste.
A primeira linha de cada caso de ensaio contém um único número inteiro N - a dimensão do vetor a.
A segunda linha de cada caso de teste contém N inteiros $[a_1, a_2, ..., a_n]$ - elementos do vetor.

Saída

Para cada caso de teste, exiba um único inteiro - o tamanho mínimo possível do vetor b.

Restrições

$1 \le T \le 10^4$
$1 \le N \le 300.000$
$0 \le a_i \le 10^9$

Continua no verso >

Exemplo

Entrada	Saída
4 5 1 3 3 3 7 2 4 2 4 1 1 1 1 7 5 4 2 1 0 0 4	2 2 1 3

No primeiro caso de teste, o contraste do vetor é:
$|1 - 3| + |3 - 3| + |3 - 3| + |3 - 7| = 2 + 0 + 0 + 4 = 6$

Se eliminarmos todos os 3 do vetor resultando no vetor $[1, 7]$, então o contraste será:
$|1 - 7| = 6$

Esse é o menor vetor que podemos obter e manter o contraste, logo a resposta é 2 (o tamanho do menor vetor possível).

D: Leitura Otimizada

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

Lex quer se tornar um leitor otimizado, ou seja, ganhar o máximo de habilidades ao ler livros no menor tempo possível. Por algum motivo, ele acha que isso é uma boa ideia.
Lex possui n livros, ler o livro i requer m_i minutos e pode dá-lo alguma (talvez nenhuma) das duas habilidades que ele deseja. As habilidades são representadas por uma string binária de tamanho 2.
Qual é o menor tempo necessário para que Lex adquira as duas habilidades?

Entrada

A primeira linha consiste de um inteiro T, o número de casos de teste. Cada caso de teste é composto por diversas linhas. A primeira linha contém um inteiro N, o número de livros disponíveis. Em seguida, N linhas seguem contendo dois valores inteiros (m_i e s_i). m_i é o número de minutos necessários para ler o livro i. s_i é a string binária de habilidades que ele ganha ao ler o livro i, por exemplo a string s_i "11" dá a Lex as duas habilidades que ele deseja. A string "10" dá a primeira habilidade.

Saída

Para cada caso de teste, exiba um único inteiro, o menor tempo necessário para Lex adquirir as duas habilidades. Caso não seja possível obtê-las, imprima -1.

Restrições

$1 \leq T \leq 10^4$
$1 \leq N \leq 200.000$
$1 \leq m_i \leq 10^9$
A soma de N em todos os casos de teste não excede 10^6.

Exemplo

Entrada	Saída
1 4 2 00 3 10 4 01 4 00	7

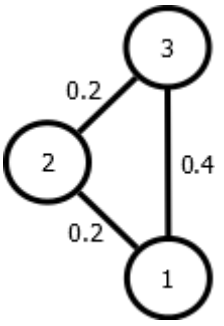
E: Old MacDonald Had a Farm

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

"Old MacDonald Had a Farm" é uma canção infantil sobre um fazendeiro chamado MacDonald que tinha muitos animais em sua fazenda. Algo que poucos sabem é que essa canção foi baseada em uma pessoa. Essa canção é antiga e, infelizmente, o senhor MacDonald não está mais vivo. Sua fazenda foi passada de geração em geração e, atualmente, seu bisneto tenta mantê-la funcionando. Ele está tendo dificuldades financeiras e quer diminuir seus gastos. Um grande problema que ele tem é o transporte dos ovos produzidos até as cidades onde os vende. Ao transportar os ovos, um certo percentual é perdido ou quebrado devido às condições precárias das estradas por onde passa.

Ele quer minimizar essas perdas para poder economizar e continuar a trabalhar na fazenda de seu bisavô. Ele anotou o percentual de ovos que perde ao passar por cada estrada mas agora está com dificuldades para escolher qual caminho deve tomar para diminuir suas perdas.

Você, comovido com a situação, resolveu ajudá-lo. Você decidiu criar um programa que, dada a quantidade de cidades pelas quais ele pode passar e a lista de estradas e suas respectivas perdas percentuais, calcula a menor perda percentual possível. Para facilitar o problema você decidiu enumerar as cidades de 1 a N e considerar que todas as estradas são de mão dupla. Também tomou o cuidado para que a cidade de origem fosse sempre 1 e a destino fosse N. Um exemplo do problema é ilustrado na figura abaixo.



Nela, o bisneto de MacDonald quer ir da cidade 1 para a cidade 3; logo ele pode escolher um dos seguintes caminhos:

- Ir até a cidade 2 e então ir para 3;
- Ir diretamente para a cidade 3.

No primeiro caso a sua perda percentual seria 36.00% já que no trecho entre as cidades 1 e 2 ele perderia 20% do total e no segundo trecho perderia 20% do restante. No segundo caso a perda seria de 40.00%. Então, mesmo sendo um caminho maior, é preferível passar antes pela cidade 2.

Continua no verso >

Entrada

A primeira linha da entrada contém um inteiro T, o número de casos de teste. Cada caso de teste é composto por várias linhas. A primeira linha de um caso de teste contém dois valores inteiros N e M que representam a
quantidade de cidades e a quantidade de estradas, respectivamente. As M linhas seguintes contém dois valores inteiros A e B que representam as cidades que a estrada liga e um valor decimal P que representa o percentual de perda ao transitar por aquela estrada. A entrada termina quando N = M = 0.

Saída

Para cada caso de teste, exiba uma linha contendo o menor percentual de perda possível com arredondamento na segunda casa decimal.

Restrições

$1 \leq T \leq 100$
$1 \leq N \leq 1000$
$1 \leq M \leq N*(N-1)/2$
$1 \leq A, B \leq N$
$0,00 \leq P \leq 1,00$

Exemplo

Entrada	Saída
1 3 3 1 2 0.20 2 3 0.20 1 3 0.40	36.00%

F: Caverna do Dragão

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

Nesse ano de 2023 houve o lançamento do filme Dungeons and Dragons honra entre rebeldes, que é uma adaptação do jogo de RPG de mesa Dungeons & Dragons (D&D). Nesse jogo, diversos amigos se reúnem e cada um interpreta um personagens da era medieval, com total controle de suas ações. Um dos jogadores é o mestre, ou seja, o responsável por narrar a história e decidir se as ações dos jogadores deram certo ou não.
Para auxiliar o mestre nessas decisões é utilizado um dado de 20 lados (d20), onde o mestre define um valor (de acordo com a dificuldade da ação ou algum outro critério) e o jogador precisa rolar o dado e somá-lo ao modificador que seu personagem tem para aquela situação. Caso o valor final seja maior que o valor definido pelo mestre a ação é realizada com sucesso.

Por exemplo, Heitor está jogando com um Arqueiro Elfo chamado Legolas e possui +5 de acerto com o seu arco. Ele está enfrentando um humano guerreiro que tem um valor 15 de armadura e, ao rolar 1d20, ele deve somar 5 e verificar se o valor final foi maior do que 15.

Dependendo da situação, o mestre pode dar vantagem ou desvantagem no ataque. Se, no exemplo acima, Legolas está escondido em cima de uma árvore para atacar o humano, o mestre pode definir que Heitor tem vantagem nesse ataque. Em caso de vantagem, o jogador deve rolar dois d20 e utilizar o maior valor no cálculo. Já em situação de desvantagem, por exemplo, se Legolas tentasse atingir o Humano de uma distância muito maior do que ele está acostumado, o jogador deve rolar dois d20 e utilizar o menor valor para o cálculo.

Um detalhe fundamental é que se o valor utilizado do dado é 20, então o ataque é considerado um sucesso automaticamente.
Crie um programa para ajudar Heitor a saber quais são as chances de seu ataque acertar de acordo com cada situação do combate.

Continua no verso >

Entrada

A entrada contém um inteiro T que representa a quantidade de casos de teste. Cada caso de teste é composto por apenas uma linha, contendo a situação do ataque S, o inteiro M, que representa o modificador do ataque e o inteiro A, que representa a armadura da criatura.

Saída

Para cada caso de teste, seu programa deve mostrar a chance do ataque acertar, com duas casas decimais (não esquecer o caractere “%” após o valor).

Restrições

$1 \leq T \leq 40$
$S \in$ [“Normal”, ”Vantagem”, “Desvantagem”]
$0 \leq M \leq 10$
$0 \leq A \leq 30$

Exemplo

Entrada	Saída
4 Normal 3 15 Desvantagem 2 10 Vantagem 0 9 Normal 2 10	40.00% 36.00% 79.75% 60.00%

G: Quero meu Balão

Arquivo: balao.[cpp/c/java/py]
Cor: rosa

Exiba “quero meu balao”.

Entrada

Não há.

Saída

Exiba uma única linha com o texto “quero meu balao”. Não coloque acentos ou letras maiúsculas. Quem estiver utilizando C ou C++, por favor, não esqueça do “\n”.
“Presentation error” significa que sobrou/faltou caracteres invisíveis como espaços ou quebras de linha.

Restrições

Não há.

Exemplo

Entrada	Saída
	quero meu balao

H: Samuel, O Cafeicultor

Arquivo: cafe.[cpp/c/java/py]
Cor: lilás

“Você gosta de café? Pois é, a maioria das pessoas gostam de café. Já pensou em comprar um terreno e fazer o seu próprio café? Um sonho, não é mesmo? Hoje é seu dia de sorte. Venha negociar um terreno com a gente!!!”

Samuel adora café. Vendo esse anúncio, logo correu para a imobiliária com seu rico dinheirinho para começar a plantar seu próprio café. Na imobiliária, foram oferecidos dois terrenos de mesmo preço para Samuel.

Como os terrenos foram adicionados ao sistema da imobiliária há apenas alguns dias, o gerente sabe apenas que cada terreno é retangular e conhece as localizações dos vértices superior esquerdo e inferior direito.

Samuel quer sua ajuda para comprar o terreno de maior área. Quem sabe ele não te dê uma xícara de café?

Portanto, dado os dois vértices do terreno A e os dois vértices do terreno B, informe para Samuel qual deles tem a maior área.

Continua no verso >

Entrada

A primeira linha da entrada é um inteiro T que indica a quantidade de casos de teste.
Cada caso de teste é composto por uma linha com oito inteiros Xsa, Ysa, Xia, Yia, Xsb, Ysb, Xib, Yib.
Xsa e Ysa são as coordenadas do vértice superior esquerdo do terreno A.
Xsb e Ysb são as coordenadas do vértice superior esquerdo do terreno B.
Xia e Yia são as coordenadas do vértice inferior direito do terreno A.
Xib e Yib são as coordenadas do vértice inferior direito do terreno B.

Saída

Para cada caso de teste, exiba uma linha contendo o terreno que Samuel deve comprar (terreno A ou terreno B). Em caso de empate, deve ser escolhido o terreno B.

Restrições

$1 \leq T \leq 100$
$1 \leq Xsa, Ysa, Xia, Yia, Xsb, Ysb, Xib, Yib \leq 100$
$Xsa < Xia, Xsb < Xib$
$Ysa > Yia, Ysb > Yib$
Pode haver intersecção entre os terrenos e isso não vai impedir Samuel de comprar um terreno.

Exemplo

Entrada	Saída
3 41 86 84 35 24 57 54 28 23 52 83 25 38 92 80 36 48 71 72 31 8 83 80 6	terreno A terreno B terreno B

I: Cycles

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

João é o mais novo programador na empresa. Como todo novo programador, ele tem uma tarefa especial: alterar e compilar um código legado. Nem tudo são flores, ele encontrou seu primeiro problema, ninguém na empresa sabe como compilar o código legado. O código foi feito em uma linguagem muito antiga e, para piorar, parece possuir algumas dependências de bibliotecas e do próprio código que não tinham sido resolvidas.
Você, um programador mais experiente, sentiu pena de João e decidiu ajudá-lo.
Depois de algumas horas, vocês descobriram que o código tinha importações cíclicas. Isso quer dizer que o arquivo A faz a importação do arquivo B e o arquivo B faz a importação do arquivo A. Na linguagem de programação do projeto, ciclos de importação não são permitidos.
Para ajudar a encontrar os ciclos você decidiu fazer um programa que, dadas as importações de arquivos, indica se existem ciclos de importação entre eles.

Continua no verso >

Entrada

A primeira linha da entrada consiste de um inteiro T indicando o número de casos de testes. Cada caso de teste é composto por diversas linhas. A primeira linha contém dois inteiros N e M indicando o número de arquivos e quantas importações existem em todos os arquivos, respectivamente. Seguido M par de inteiros (A e B) indicando que o arquivo A importa o arquivo B.

Saída

Para cada caso de teste, exiba uma linha contendo "Imports with cycle are not allowed" quando existirem ciclos de importação ou “Good to go”, caso contrário.

Restrições

N ≤ 200
M ≤ 40000
1 ≤ A ≤ N
1 ≤ B ≤ N

Exemplo

Entrada	Saída
3 2 2 1 2 2 1 3 3 1 2 2 3 3 1 3 3 1 2 1 3 2 3	Imports with cycles are not allowed Imports with cycles are not allowed Good to go

J: Soma de Verificação

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

Soma de verificação (do inglês Checksum) é um código usado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo.

Isto é feito calculando a soma de verificação dos dados antes do envio ou do armazenamento deles, e recalculá-los ao recebê-los ou recuperá-los do armazenamento. Se o valor obtido é o mesmo, as informações não sofreram alterações e portanto não estão corrompidas.

Há várias formas de se calcular estas somas para diferentes aplicações, como por exemplo o MD5 e o CRC.

A verificação cíclica de redundância (do inglês, CRC - Cyclic Redundancy Check) é um método de detecção de erros normalmente usado em redes digitais e dispositivos de armazenamento para detectar mudança acidental em cadeias de dados. Mensagens de dados entrando nesses sistemas recebem um pequeno anexo com um valor de verificação baseado no resto de divisão polinomial do seu conteúdo. No ato da recuperação do dado o cálculo é refeito e comparado com o valor gerado anteriormente. Se os valores não se mostrarem semelhantes podem ser aplicadas ações para correção de dados, evitando assim a corrupção de dados.

Um CRC é chamado de n-bit CRC quando seu código de verificação possui tamanho de n bits. Para um dado n, múltiplos códigos de verificação são possíveis de serem construídos, cada um com um polinômio diferente. Tal polinômio possui o maior grau de valor n, além de n + 1 termos. Ou seja, ele possui tamanho n + 1 bits.

Para calcular um CRC de n bits, coloque os bits representando o dado em uma linha, e abaixo do bit mais significativo dessa linha posicione o padrão (n + 1)-bit que representa o divisor CRC, também chamado de "polinômio".

Neste exemplo será utilizado uma mensagem de 14 bits, com um 3-bit CRC, e o polinômio gerador. O polinômio precisa ser escrito em binário, um polinômio de 3ª ordem possui 4 coeficientes. Nesse caso, os coeficientes são: 1, 0, 1, 1. O resultado do cálculo tem 3 bits de tamanho.

Continua no verso >

A mensagem a ser codificada será:

```
11010011101100
```

Primeiro, são adicionados zeros no fim da palavra correspondentes ao tamanho n do CRC. Então é feito o primeiro cálculo para se obter o 3-bit CRC:

```
11010011101100 000 <--- entrada deslocada 3 bits a esquerda
1011               <--- divisor (4 bits) = x³ + x + 1
------------------
01100011101100 000 <--- resultado
```

O algoritmo age sobre os bits diretamente acima do divisor em cada passo. O resultado para a iteração acima é a operação OU-exclusivo bit-a-bit para os bits acima do divisor polinomial. Os bits restantes são diretamente copiados para o resultado daquele passo. O Divisor é então deslocado para a direita em 1 bit e o processo é repetido até que o divisor alcance o último bit da mensagem. O cálculo completo pode ser visto a seguir:

```
11010011101100 000 <--- entrada deslocada 3 bits a esquerda
1011               <--- divisor
01100011101100 000 <--- resultado (note que os primeiros 4 bits são um XOR com os
 1011                   bits acima do divisor, o resto é copiado).
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000 <--- o divisor irá se mover mais de 1 bit para se alinhar com o
       1011            próximo bit 1 do dividendo (pois o quociente para aquele
00000000110100 000     passo seria 0).(Ou seja, ele não se move apenas 1 bit
        1011           necessariamente.)
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
------------------
00000000000000 100 <--- resto (3 bits). O algoritmo de divisão para neste ponto
                       pois o dividendo é zero.
```

Como o bit divisor mais à esquerda zerou todos os bits que interagiu, ao final desse processo apenas os n bits mais à direita da mensagem não serão zerados. Esses n bits correspondem ao resto da divisão polinomial, e ao valor da função CRC.

Continua no verso >

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 2 valores hexadecimais separados por espaço que são, respectivamente, a mensagem M e o polinômio P.

Saída

Para cada caso de teste, exiba uma linha contendo o valor C da função CRC para a entrada fornecida.

Restrições

$1 \leq N \leq 9999$
$0 \leq M \leq FFFFFFFFFFFFFFFFFFFF$
$3 \leq P \leq 1FFFFFFFF$

Exemplo

Entrada	Saída
5 34EC B 6D 113 E100CAFE 131 4C D 0 3	4 D 23 1 0

K: HameKameKa

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

O Hamekameka foi inventado por Mestre Hame praticado por cinquenta anos antes de conhecer Kogu. Chamando sua energia latente nas palmas de suas mãos, Hame consegue lançar um raio explosivo de energia. Kogu aprende após ver Mestre Hame usando-o para apagar as chamas na casa de um Rei. Para a surpresa de Hame, Kogu consegue performar a técnica de primeira, embora seja apenas forte o suficiente para destruir o carro que Chamya deu para Mulba. Kogu descobriu que há um padrão na pronúncia correta deste ataque, de modo que, se não for pronunciado corretamente, o mesmo não acontece.

Escreva um programa que, dada a parte inicial de um Hamekameka, faça a finalização ideal para que o ataque seja realizado com sucesso.

Entrada

A primeira linha da entrada contém um inteiro C, indicando a quantidade de casos de teste. Cada caso de teste é composto por uma única linha contendo uma string S, o início de um ataque.

Saída

Para cada caso de teste, imprima a finalização adequada, para que o ataque se concretize.

Restrições

$1 \leq N \leq 100$
$1 \leq |S| \leq 200$

Exemplo

Entrada	Saída
4 hamekame haamekaame haaamekaame haaaamekaaame	ka kaaaa kaaaaaa kaaaaaaaaaaa