regionals | latin america region
icpc international collegiate programming contest
icpc.foundation

Maratona de Programação da SBC 2025

Sub-Regional Brasil do ICPC

13 de Setembro de 2025

Caderno de Problemas

Informações Gerais

Este caderno contém 13 problemas; as páginas estão numeradas de 1 a 20, não contando esta página de rosto. Verifique se o caderno está completo.

Este conjunto de problemas também está sendo utilizado simultaneamente nas seguintes competições: The 2025 ICPC Gran Premio de Centroamerica, The 2025 ICPC Gran Premio de Mexico e Competencia Boliviana de Programación.

Leia atentamente as instruções a seguir. Informações adicionais, incluindo os limites de tempo dos problemas estarão em um documento disponível no sistema utilizado para a competição, o BOCA.

A) Sobre os nomes dos programas
1) Para soluções em C/C++ e Python, o nome do arquivo-fonte não é significativo, pode ser qualquer nome.
2) Se sua solução é em Java, ela deve ser chamada *codigo*`.java` onde *codigo* é a letra maiúscula que identifica o problema. Lembre que em Java o nome da classe principal deve ser igual ao nome do arquivo.
3) Se sua solução é em Kotlin, ela deve ser chamada *codigo*`.kt` onde *codigo* é a letra maiúscula que identifica o problema. Lembre que em Kotlin 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 é composta de um único caso 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.
5) O final da entrada coincide com o final do arquivo.

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.

Promoção:

SBC Sociedade Brasileira de Computação

Problema A

Alimentação saudável

No Instituto de Competição em Programação Criativa (ICPC), todos os alunos adoram frutas! Sabendo disso, a direção decidiu realizar uma grande pesquisa sobre preferências alimentares para ajudar na elaboração do cardápio anual. Para tornar o levantamento mais profissional, contrataram a empresa **SBC Research Solutions™**, sigla para "Saladas Bem Científicas", embora alguns digam que o nome seja homenagem a uma conhecida sociedade de computação...

A SBC recebeu a seguinte missão: a escola tem M turmas e oferece N tipos diferentes de frutas. Em cada turma, para cada fruta, foi informado o número de alunos que gostam daquela fruta.

Contudo, como a SBC não teve acesso aos dados individuais dos alunos, nem sabe quantos alunos há em cada turma, precisa agora de sua ajuda! Com base apenas nos resultados da pesquisa (quantos alunos gostam de cada fruta em cada turma), determine o menor número possível de alunos que a escola pode ter, sabendo que as seguintes restrições são obedecidas:

- cada turma tem pelo menos um aluno;
- cada aluno pertence a uma única turma;
- cada aluno gosta de pelo menos uma fruta;
- um mesmo aluno pode gostar de várias frutas.

Entrada

A primeira linha contém dois inteiros N e M ($1 \leq N, M \leq 1000$), respectivamente o número de frutas e o número de turmas. Cada uma das N linhas seguintes contém M inteiros $G_{i,j}$, indicando quantos alunos da turma j gostam da fruta i ($0 \leq G_{i,j} \leq 10^6$ para $1 \leq i \leq N$ e $1 \leq j \leq M$).

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o menor número possível de alunos na escola, considerando as restrições dadas.

Exemplo de entrada 1	**Exemplo de saída 1**
3 3 20 15 14 12 20 12 18 5 10	54

Exemplo de entrada 2	**Exemplo de saída 2**
2 3 5 2 4 4 3 6	14

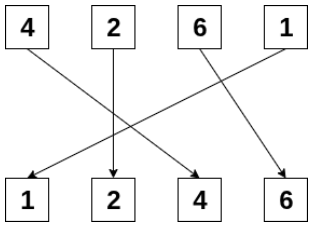
Problema B

Baralho Alho

Pesquisadora Isadora adora jogar baralho com seus amigos. Mais especificamente, ela joga uma versão chamada Baralho Alho, em que há N cartas (podendo haver cartas iguais). Inicialmente, as N cartas estão em uma ordem específica: a i-ésima carta tem valor A_i. Duas cartas são consideradas iguais se possuem o mesmo valor.

Antes do jogo começar, Isadora declarou: "eu sempre embaralho Baralho Alho". Ingenuamente, seus amigos toparam e deixaram que ela comandasse o embaralhamento. Mal sabem eles que Pesquisadora Isadora adora trapacear. Seu objetivo é embaralhar de tal forma que, ao final do processo, a i-ésima carta tenha valor B_i.

No entanto, ela conhece apenas um tipo de embaralhamento: aquele que faz a carta originalmente na posição i ir para a posição P_i. Por exemplo, se $P = [3, 2, 4, 1]$, então a primeira carta vai para a terceira posição, a segunda permanece na mesma, a terceira vai para a quarta e a quarta vai para a primeira. Assim, se o baralho inicial é $[4, 2, 6, 1]$, após aplicar o embaralhamento Isadora obtém $[1, 2, 4, 6]$.



Mesmo com essa limitação, Isadora é bastante inteligente e pensa em repetir o embaralhamento várias vezes para tentar alcançar novas configurações do baralho.

Escreva um programa que, dados A_i, B_i e P_i, determine a menor quantidade de vezes que Isadora precisa aplicar o embaralhamento para que o baralho fique na ordem desejada. Caso isso seja impossível, imprima "`IMPOSSIVEL`" (sem aspas). Se o número mínimo de embaralhamentos for maior que 10^9, imprima "`DEMAIS`" (sem aspas).

Entrada

A primeira linha da entrada contém um inteiro N ($1 \leq N \leq 10^6$).

A segunda linha contém N inteiros A_i ($1 \leq A_i \leq 10^9$), representando a configuração inicial do baralho.

A terceira linha contém N inteiros B_i ($1 \leq B_i \leq 10^9$), representando a configuração final desejada do baralho.

A quarta linha contém N inteiros distintos P_i ($1 \leq P_i \leq N$), indicando que a carta na posição i vai para a posição P_i após uma aplicação do embaralhamento.

Saída

Imprima um único inteiro k: o menor número de vezes que o embaralhamento deve ser aplicado, a partir de A_i, até que a configuração obtida seja B_i.

Se isso for impossível, imprima "`IMPOSSIVEL`" (sem aspas).

Se o menor k for maior que 10^9, imprima "`DEMAIS`" (sem aspas).

Exemplo de entrada 1	**Exemplo de saída 1**
6 8 6 5 5 1 3 5 1 8 5 3 6 2 3 6 5 1 4	2

Explicação do exemplo 1:

Podemos ver a configuração do baralho após cada quantidade k de embaralhementos:

- $k = 0$: o baralho está na ordem $8, 6, 5, 5, 1, 3$;
- $k = 1$: o baralho está na ordem $1, 8, 6, 3, 5, 5$;
- $k = 2$: o baralho está na ordem $5, 1, 8, 5, 3, 6$.

Logo, a resposta é $k = 2$.

Exemplo de entrada 2	**Exemplo de saída 2**
2 3 3 3 3 1 2	0

Explicação do exemplo 2:

Neste caso, o baralho já se encontra na configuração desejada, portanto não é necessário nenhuma operação de embaralhamento.

Exemplo de entrada 3	**Exemplo de saída 3**
5 6 3 8 4 2 3 6 4 2 8 2 1 4 5 3	5

Exemplo de entrada 4	**Exemplo de saída 4**
4 1 2 1 2 1 2 2 1 2 1 4 3	IMPOSSIVEL

Exemplo de entrada 5	**Exemplo de saída 5**
3 1 2 3 2 1 4 1 2 3	IMPOSSIVEL

Problema C

Collatz polinomial

Todos conhecem (ou já ouviram falar) da famosa Conjectura de Collatz: pegue um número inteiro positivo. Se ele for ímpar, multiplique por 3 e some 1. Se for par, divida por 2. Repita o processo até chegar em 1. Apesar de sua simplicidade, ninguém sabe provar se a sequência realmente sempre alcança 1, qualquer que seja o número inicial.

Aline, fã desse tipo de curiosidade, decidiu criar uma variação usando polinômios em vez de números. Para não complicar, ela trabalha apenas com polinômios cujos coeficientes são 0 ou 1, ou seja, cada potência de x aparece no máximo uma vez.

A brincadeira funciona assim:

- Se o polinômio possui termo constante (um termo que não depende de x), Aline multiplica o polinômio por $(x+1)$ e depois soma 1. Caso algum coeficiente resultante seja igual a 2, o termo correspondente é descartado (observe que coeficientes maiores que 2 não podem surgir).
- Se o polinômio não possui termo constante, Aline divide o polinômio por x.

Esse processo se repete até que o polinômio se reduza a $P(x) = 1$.

Considere $P(x) = x^3 + 1$. No primeiro passo há termo constante, então calculamos:

$$(x^3 + 1) \cdot (x + 1) + 1 = x^4 + x^3 + x + 1 + 1.$$

Como o coeficiente do termo constante resulta em 2, esse termo é descartado, restando:

$$x^4 + x^3 + x.$$

Em seguida, como não há termo constante, dividimos por x:

$$x^3 + x^2 + 1.$$

Continuando:

- Passo 3: $x^4 + x^2 + x$
- Passo 4: $x^3 + x + 1$
- Passo 5: $x^4 + x^3 + x^2$
- Passo 6: $x^3 + x^2 + x$
- Passo 7: $x^2 + x + 1$
- Passo 8: x^3
- Passo 9: x^2
- Passo 10: x
- Passo 11: 1

No total, foram necessárias 11 operações para chegar ao polinômio $P(x) = 1$.

Aline precisa de ajuda para estudar essa variação da Conjectura de Collatz. Como fazer essas contas manualmente é suscetível a erros, escreva um programa que determine o número de operações necessárias até o polinômio se tornar $P(x) = 1$.

Entrada

A primeira linha contém um inteiro N ($0 \leq N \leq 20$), indicando o grau do polinômio.

A segunda linha contém $N + 1$ inteiros $a_N,\ a_{N-1},\ \ldots,\ a_0$ (cada um igual a 0 ou 1), onde $a_i = 1$ indica que o termo x^i está presente no polinômio, e $a_i = 0$ indica que não está. Note que $a_N = 1$, já que o grau do polinômio é N.

Saída

Seu programa deve produzir uma única linha com o número de operações necessárias até o polinômio se tornar $P(x) = 1$. representando

Exemplo de entrada 1	**Exemplo de saída 1**
3 1 0 0 1	11

Exemplo de entrada 2	**Exemplo de saída 2**
2 1 0 1	6

Exemplo de entrada 3	**Exemplo de saída 3**
2 1 0 0	2

Exemplo de entrada 4	**Exemplo de saída 4**
0 1	0

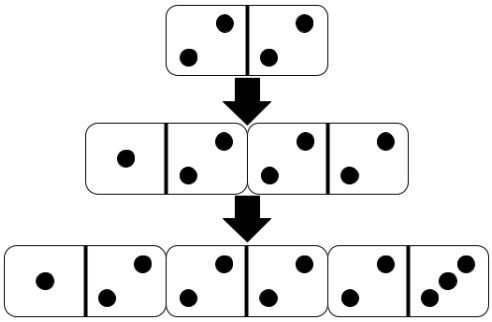
Problema D

Dominós

Alice, cansada de sair com seus amigos, inventou sua própria variação do famoso jogo de dominó. Sua versão usa o mesmo conjunto de peças de dominó que o jogo tradicional, em que cada peça é uma ficha que mostra dois números entre um e seis, com um número em cada extremidade.

Para começar o jogo, Alice embaralha as peças de dominó em uma pilha e então pega uma peça de cada vez do topo. Quando Alice pega a primeira peça da pilha, ela simplesmente a coloca na mesa. Da segunda peça em diante, ela precisa colocá-la no lado esquerdo ou no direito da corrente formada pelas peças já na mesa.

Para colocar uma peça em uma extremidade da corrente, um de seus números deve corresponder ao número naquela extremidade. O outro número da peça recém-jogada se torna a nova extremidade livre da corrente. Se Alice não conseguir colocar uma peça, ela perde o jogo; caso contrário, se conseguir colocar todas as peças com sucesso, ela vence.



Alice não está interessada em jogos que não pode vencer, então ela gostaria de saber se é possível vencer com um determinado conjunto de dominós. Além disso, ela pode não ter um conjunto completo, pois pode ter perdido algumas peças.

Como Alice pode escolher jogar com um subconjunto de suas peças (efetivamente descartando o restante para tornar o jogo vencível), você deve calcular o número total de subconjuntos de seus dominós para os quais existe uma chance não nula de vitória.

Entrada

A entrada contém múltiplos casos de teste. A primeira linha contém um inteiro T ($1 \leq T \leq 10^4$), o número de casos de teste. Cada caso de teste é descrito da seguinte forma:

A primeira linha do caso contém um inteiro N ($1 \leq N \leq 21$), o número de peças que Alice possui.

As N linhas seguintes contêm dois inteiros a_i e b_i ($1 \leq a_i \leq b_i \leq 6$), indicando que a i-ésima peça de dominó que Alice possui contém os números a_i e b_i.

É garantido que Alice não possui peças duplicadas; em outras palavras, $(a_i, b_i) \neq (a_j, b_j)$ se $i \neq j$.

Saída

Para cada caso de teste, imprima uma única linha contendo um inteiro: o número de subconjuntos de peças com os quais Alice tem uma chance não nula de vencer.

Exemplo de entrada 1	**Exemplo de saída 1**
3 3 1 2 2 3 3 4 4 1 1 1 2 2 3 3 4 3 1 2 2 2 2 3	6 10 7

Explicação do exemplo 1:

Para o terceiro caso de entrada, temos as peças 1-2, 2-2 e 2-3. Considerando o subconjunto em que todas as peças estão presentes, uma ordem possível para as peças retiradas da pilha após o embaralhamento é a mostrada na imagem: 2-2, seguida de 1-2 e, por fim, 2-3. Nessa ordem, Alice pode colocar a peça 1-2 à esquerda da 2-2 e, em seguida, a 2-3 à direita, como ilustrado na imagem. Portanto, para este subconjunto, a chance de Alice vencer não é nula.

Problema E

Expansão rodoviária

Diz a lenda que, há muito tempo, o Serviço de Bandeirantes Conectados (SBC) administrava uma rede de estradas bidirecionais que ligava diversas cidades. Naquela época, o traçado era extremamente simples: entre quaisquer duas cidades existia exatamente um único caminho.

Com o crescimento da população e o aumento no transporte de mercadorias, o Instituto de Cidades Planejadas e Conectadas (ICPC) assumiu o controle e decidiu modernizar a rede. Para evitar o trânsito interno nas cidades intermediárias e agilizar as viagens, foram construídas novas estradas diretas entre certos pares de cidades. Uma nova estrada era criada entre duas cidades A e B sempre que, no traçado original, o caminho entre elas passava por exatamente uma única cidade intermediária.

Hoje, só conhecemos o mapa atual da rede, e o ICPC quer descobrir se ele pode mesmo ter surgido seguindo esse processo.

Sua tarefa é analisar o mapa atual e determinar se a lenda pode ser verdadeira. Caso seja possível, também deve reconstruir e imprimir um possível traçado original da rede.

Entrada

A primeira linha contém dois inteiros N e M ($3 \leq N \leq 10^5$, $2 \leq M \leq 4 \times 10^5$), representando, respectivamente, o número de cidades e o número de estradas no mapa atual.

Cada uma das M linhas seguintes contém dois inteiros u_i e v_i ($1 \leq u_i, v_i \leq N$, $u_i \neq v_i$), indicando que há uma estrada bidirecional entre as cidades u_i e v_i. No mapa atual, é garantido que existe um caminho entre qualquer par de cidades e não há mais de uma estrada entre o mesmo par de cidades.

Saída

Caso o mapa atual possa ter surgido seguindo o processo descrito na lenda, a saída deve conter $N - 1$ linhas. Cada linha deve conter dois inteiros a_i e b_i ($1 \leq a_i, b_i \leq N$, $a_i \neq b_i$), representando que havia uma estrada direta entre as cidades a_i e b_i no traçado original.

Caso contrário, a saída deve conter apenas uma linha com um único caractere "`*`" (asterisco).

Caso haja mais de um possível traçado original, você pode imprimir qualquer um deles.

Exemplo de entrada 1	**Exemplo de saída 1**
3 3 1 2 3 2 1 3	1 2 1 3

Explicação do exemplo 1:

Um possível traçado original consiste em $1 - 2 - 3$. Neste caso, uma única estrada foi adicionada pelo processo descrito na lenda, a estrada $1 - 3$, que passa pela cidade intermediária 2.

Outros possíveis traçados originais são $1 - 3 - 2$ e $2 - 1 - 3$.

Exemplo de entrada 2	**Exemplo de saída 2**
3 2 1 2 2 3	*

Problema F

Frangolino ali na mesa

Washington, um chef entusiasta da inteligência artificial e amante da culinária, decidiu construir um robô-garçom para o seu novo restaurante, o Frangolino, especializado em frangos empanados. Washington abrirá o restaurante para uma noite especial entre amigos e decidiu testar o robô-garçom nessa ocasião.

O restaurante atenderá N mesas, numeradas de 1 a N, e servirá somente um prato: frango à milanesa. Washington gosta muito de brincar com palavras e decidiu então definir dois comandos para o robô-garçom: o comando "ali na mesa X" e o comando "a milanesa X".

O comando "ali na mesa X" significa que o robô-garçom deve se dirigir até a mesa X e aguardar a próxima instrução. Já o comando "a milanesa X" significa que o robô-garçom deve registrar no sistema um pedido de X frangos à milanesa para a mesa em que ele se encontra. No início da noite, o robô-garçom se encontra na mesa 1.

Infelizmente, o robô-garçom tem um defeito e não consegue lidar bem com anagramas. Para cada comando recebido, o robô tem 50% de chance de executá-lo corretamente e 50% de chance de executar o outro comando. Sua tarefa é, dado o histórico de comandos recebidos pelo robô, determinar, para cada mesa do restaurante, o valor esperado do número de frangos à milanesa que serão servidos.

Entrada

A primeira linha da entrada contém dois inteiros N e Q ($1 \leq N, Q \leq 10^5$), o número de mesas e o número de comandos do robô-garçom, respectivamente.

A segunda linha contém Q inteiros $X_1, \ldots, X_Q$ ($1 \leq X_i \leq N$) separados por espaço. Cada X_i descreve o valor de um dos comandos que o robô-garçom recebeu, na ordem em que ocorreram.

Note que o comando em si não é fornecido, já que o robô-garçom executará com 50% de chance um ou outro.

Saída

A saída deve conter N linhas. A i-ésima linha deverá conter o valor esperado do número de frangos à milanesa que serão servidos na mesa i, conforme descrito a seguir.

O valor esperado pode não ser um inteiro, mas sempre será um número racional e, portanto, pode ser representado por uma fração irredutível $\frac{p}{q}$. Como p e q podem ser muito grandes, você deve imprimir $p \times q^{-1} \mod (10^9 + 7)$, onde q^{-1} é o inverso multiplicativo de q módulo $10^9 + 7$.

Exemplo de entrada 1	**Exemplo de saída 1**
2 3 1 2 1	750000007 250000002

Exemplo de entrada 2	**Exemplo de saída 2**
4 4 1 2 3 4	750000008 250000003 1 0

Problema G

Gerador universal

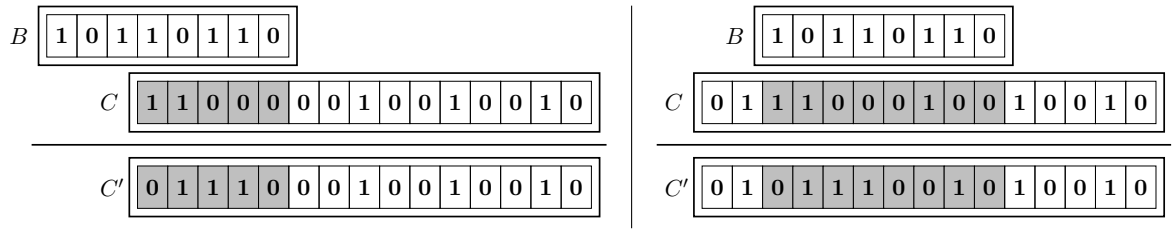
Sandy está desenvolvendo um novo computador como parte do ambicioso projeto Sistema Binário Compactador (SBC). Esse projeto faz parte de um grande desafio tecnológico conhecido como Interface de Codificação por Padrões Compactos (ICPC), cuja meta é atingir a máxima eficiência na escrita de grandes volumes de dados.

A proposta do SBC é ousada: escolher um padrão-base B, formado por 8 bits $b_0, \ldots, b_7$ e, a partir dele, gerar qualquer outro padrão aplicando apenas algumas operações simples e rápidas.

Sandy deseja escrever na memória uma sequência de N bits, com $N \geq 8$, denotada por $C = c_0, \ldots, c_{N-1}$. Inicialmente, a memória está preenchida apenas com zeros. Em seguida, ela pode repetir a seguinte operação quantas vezes quiser:

- Escolha um número inteiro i entre -7 e $N-1$, a posição onde B será aplicado;
- Para cada posição de B alinhada com a mensagem, isto é, para todo j de 0 a 7 tal que $0 \leq i+j \leq N-1$, substitua o conteúdo de c_{i+j} pelo resultado de $b_j \oplus c_{i+j}$, onde $\oplus$ representa a operação XOR (ou "ou-exclusivo").

O exemplo a seguir ilustra duas aplicações do procedimento acima. Aplicando o padrão B a um conteúdo C, obtém-se o resultado final C'.



Como os dados que desejamos escrever na memória geralmente não são aleatórios, Sandy acredita que, com uma boa escolha do padrão-base B, será possível produzir o conteúdo desejado com poucas operações.

Para testar essa hipótese, ela precisa da sua ajuda: dado o conteúdo C que deve ser escrito na memória, determine o padrão-base B que minimiza o número de operações necessárias para gerar C conforme descrito, e também a quantidade Q de operações necessárias.

Pode-se provar que é sempre possível escrever qualquer conteúdo usando esse procedimento. Porém, para que o projeto SBC seja bem-sucedido e conquiste o selo de excelência do ICPC, a sua solução precisa ser rápida e eficiente!

Entrada

A primeira linha contém um inteiro N ($8 \leq N \leq 4096$), o comprimento de C.

A segunda linha contém uma sequência de N bits C, o conteúdo que deve ser escrito na memória.

Saída

Seu programa deve produzir uma única linha, contendo uma sequência de 8 bits B, representando o padrão-base que minimiza o número de operações necessárias, e um inteiro Q, representando o número mínimo de operações.

Se houver mais de um padrão B que minimize o número de operações, imprima o que corresponder ao menor valor inteiro em base 2, considerando b_0 como o bit mais significativo e b_7 como o menos significativo.

Exemplo de entrada 1	**Exemplo de saída 1**
9 101001111	00111101 2

Exemplo de entrada 2	**Exemplo de saída 2**
12 111111001010	00010101 3

Exemplo de entrada 3	**Exemplo de saída 3**
10 0101001111	01000011 2

Problema H

Habilidades especiais

A Maratona de Programação WEB é uma competição organizada pela Sociedade Brasileira de CSS (SBC). Nela, as equipes são formadas por exatos três integrantes e o objetivo principal é desenvolver um projeto utilizando CSS e JavaScript.

Cada competidor possui um subconjunto das K habilidades mais importantes de um desenvolvedor frontend. Alguns exemplos dessas habilidades são:

1. Centralizar uma `div` (o clássico ritual do front);
2. Dominar o CSS sem perder a sanidade;
3. Lembrar a diferença entre `==` e `===`;
4. Invocar o poder místico de um `console.log()`.

Uma universidade possui N alunos, e cada aluno domina um subconjunto das K habilidades. A habilidade total de uma equipe é definida pela **união** dos subconjuntos de seus integrantes.

Por exemplo, considere as seguintes habilidades de três alunos:

$$\text{membro}_1 = \{1, 2\}, \quad \text{membro}_2 = \{2\}, \quad \text{membro}_3 = \{1, 4\}$$

Assim, a equipe formada por esses três alunos possui como conjunto de habilidades $\{1, 2, 4\}$.

O professor Joãozinho, utilizando uma LLM muito avançada, descobriu M **subconjuntos de habilidades especiais**. Se o conjunto de habilidades de uma equipe for *exatamente igual* a um desses subconjuntos especiais, então ela tem grande chance de se tornar campeã.

Agora, o professor deseja saber, para cada subconjunto especial, quantas equipes distintas, formadas por três alunos, podem ser montadas de modo que o conjunto resultante de habilidades seja exatamente aquele subconjunto.

Entrada

A primeira linha contém dois inteiros N e K ($1 \leq N \leq 10^5$, $1 \leq K \leq 20$), representando, respectivamente, o número de alunos e o total de habilidades possíveis.

As próximas N linhas contêm uma string binária H_i de tamanho K, que representa o conjunto de habilidades do aluno i. Se o caractere na posição j ($1 \leq j \leq K$) for 1, significa que o aluno possui a habilidade j; caso contrário, ele não a possui.

A próxima linha contém um inteiro M ($1 \leq M \leq 5 \cdot 10^4$), representando o número de subconjuntos especiais.

As próximas M linhas contêm uma string binária E_i de tamanho K, que representa um subconjunto especial. Se o caractere na posição j ($1 \leq j \leq K$) for 1, significa que o subconjunto especial inclui a habilidade j; caso contrário, não inclui.

Saída

A saída deve conter M linhas. A i-ésima linha deve conter um único inteiro representando a quantidade de equipes distintas, formadas por três alunos, cujo conjunto de habilidades seja exatamente igual a E_i.

Exemplo de entrada 1	**Exemplo de saída 1**
5 3 010 100 010 110 010 3 010 011 110	1 0 9

Exemplo de entrada 2	**Exemplo de saída 2**
3 2 10 01 11 1 10	0

Problema I

Investigação Cósmica

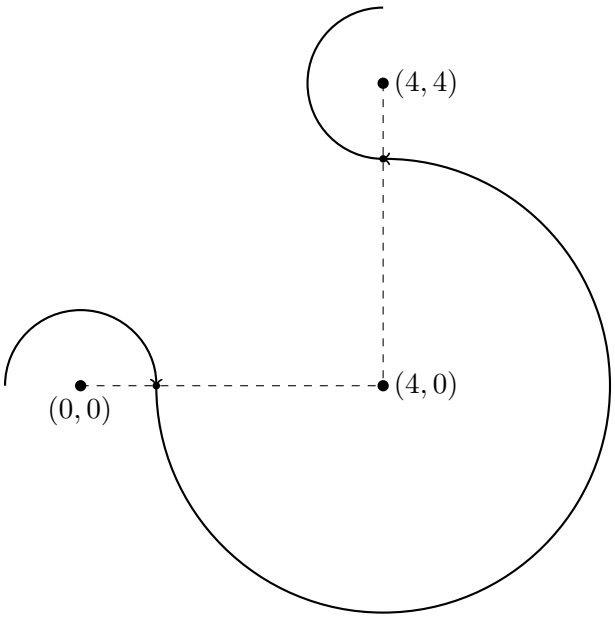
A Sociedade Brasileira de Cosmonáutica (SBC) está treinando suas equipes para a próxima edição da Investigação Cósmica de Planetas Celestes (ICPC).

A SBC fará uma exploração simulada de uma galáxia distante chamada Quadradômeda. Para esta missão, foram selecionadas N estrelas, por sua localização geometricamente estratégica, e foi determinada a ordem de visitação, identificada pela numeração de 1 a N. Para preparar as equipes, modelos simplificados são utilizados, nos quais cada estrela é representada por um ponto no plano com coordenadas inteiras (x_i, y_i).

As estrelas estão dispostas de forma que, para cada $1 \leq i < N$, a estrela i está alinhada com a estrela $i + 1$, ou seja, compartilham a mesma coordenada x ou a mesma coordenada y.

O objetivo da missão é orbitar cada estrela i em uma circunferência de raio inteiro constante $R_i \geq 1$. Durante a simulação, a nave orbita a estrela atual e, ao atingir o ponto da órbita mais próximo da próxima estrela, abandona essa órbita e passa a orbitar a estrela seguinte imediatamente. Para que essa manobra seja possível, para cada $1 \leq i < N$, o raio R_i deve ser estritamente menor que a distância euclidiana entre as estrelas i e $i + 1$.

No exemplo abaixo, é ilustrada uma configuração válida de órbitas, em que $N = 3$ e as estrelas estão nos pontos $(0, 0)$, $(4, 0)$ e $(4, 4)$. Na configuração exibida, temos $R_1 = 1$, $R_2 = 3$ e $R_3 = 1$.



Sua tarefa é determinar o maior valor inteiro possível de R_1 tal que seja possível escolher valores $R_1, R_2, \ldots, R_N$ que satisfaçam todas as condições descritas. Caso não exista nenhuma configuração válida de órbitas, informe que a missão é impossível.

Entrada

A primeira linha contém um inteiro N ($2 \leq N \leq 10^5$), o número de estrelas.

Cada uma das próximas N linhas contém dois inteiros x_i e y_i ($|x_i|, |y_i| \leq 10^9$), as coordenadas da estrela i. Para cada $1 \leq i < N$, as estrelas i e $i + 1$ estão alinhadas horizontal ou verticalmente.

Saída

Seu programa deve produzir uma única linha contendo o maior valor inteiro possível para R_1 tal que exista uma configuração válida de órbitas, ou `-1` caso a missão seja impossível.

Exemplo de entrada 1	**Exemplo de saída 1**
3 0 0 4 0 4 4	3

Exemplo de entrada 2	**Exemplo de saída 2**
5 0 0 4 0 4 2 4 6 6 6	-1

Exemplo de entrada 3	**Exemplo de saída 3**
4 0 0 4 0 4 4 4 7	2

Problema J

João João

Em 2025, o responsável por coordenar a equipe criadora da prova da OBI – Olimpíada Brasileira de Influenciadores – é o professor João João, que, além de professor, é um famoso e influente influenciador.

Até o momento, a equipe criou 10 tarefas, e cada uma dessas tarefas foi categorizada em um de quatro níveis de dificuldade: 1, 2, 3 e 4.

O professor João João deseja saber quantas tarefas ainda faltam ser criadas para que seja possível montar uma prova composta de exatamente quatro tarefas, sendo cada tarefa de um nível de dificuldade distinto.

Dada a lista dos níveis de dificuldade das tarefas já criadas, você poderia ajudar o professor João João a determinar quantas tarefas faltam ser criadas?

Entrada

A entrada é composta por uma única linha, contendo 10 inteiros $D_1, \ldots, D_{10}$, o nível de dificuldade da tarefa i ($1 \leq D_i \leq 4$, para $1 \leq i \leq 10$).

Saída

Seu programa deve produzir uma única linha na saída, contendo um único número inteiro: a menor quantidade de tarefas que precisam ser criadas para que uma prova com quatro tarefas de níveis distintos possa ser produzida.

Exemplo de entrada 1	**Exemplo de saída 1**
`1 3 4 1 3 4 1 3 4 1`	`1`

Explicação do exemplo 1:

Já foram criadas tarefas de níveis de dificuldade 1, 3 e 4. Falta apenas uma tarefa de nível 2, então a reposta é 1.

Exemplo de entrada 2	**Exemplo de saída 2**
`4 1 1 4 3 1 2 1 2 2`	`0`

Explicação do exemplo 2:

Já foram criadas tarefas de níveis de dificuldade 1, 2, 3 e 4. Como não falta nenhum nível de dificuldade, a resposta é 0.

Problema K

Knockout, suíço e outros formatos de torneio

Torneios de jogos e esportes são cada vez mais frequentes, servindo como uma forma de testar as habilidades dos participantes. A escolha do formato ideal depende da modalidade e do número de participantes. Por exemplo, um formato "todos contra todos" pode ser inviável quando o número de participantes é grande, enquanto um torneio de *knockout* (conhecido popularmente como "mata-mata") pode ser frustrante quando dois fortes participantes se enfrentam precocemente. Já o formato "suíço" é um bom intermediário e utilizado em várias modalidades. Neste formato, se há N participantes no torneio, ocorrerão cerca de $\lceil \log_2 N \rceil$ rodadas, nas quais os jogadores sempre enfrentam oponentes com pontuações similares.

Um formato similar vem sendo usado em alguns jogos, como alternativa ao formato suíço: chamaremos este formato de "(A, B)-eliminação". Neste formato, toda partida tem um vencedor e um perdedor (ou seja, não há empates) e jogadores sempre enfrentam oponentes com a mesma pontuação que eles. Cada participante joga partidas até alcançar A vitórias ou B derrotas, o que ocorrer primeiro. Este formato tem uma característica muito conveniente: o número de rodadas não depende do número de participantes. Além da escalabilidade, essa característica facilita a definição da estrutura de premiação, uma vez que é possível prever o número de participantes com cada pontuação possível.

Apesar das vantagens, este formato tem uma complicação: nem sempre é possível realizá-lo, pois pode não haver oponentes suficientes para completar uma rodada com as restrições estabelecidas. Por exemplo, se $A = 2$ e $B = 2$, e temos $N = 6$ participantes, após a primeira rodada teríamos 3 jogadores com 1 vitória e 0 derrotas. Então, na segunda rodada, como esse número é ímpar, não será possível parear todos os jogadores com oponentes de mesma pontuação. Por outro lado, se $N = 8$, é possível realizar o pareamento.

Sua ajuda foi solicitada para determinar, dados os valores de A e B, qual é o menor número de jogadores tal que todas as rodadas do torneio possam ser realizadas.

Entrada

A entrada consiste de uma única linha contendo dois inteiros A e B ($1 \leq A, B \leq 10^{18}$) separados por espaço, definindo o formato do torneio conforme descrito acima.

Saída

Seu programa deve escrever uma única linha, contendo o menor número de jogadores possível no torneio. Como a resposta pode ser muito grande, imprima a resposta módulo $10^9 + 7$.

Exemplo de entrada 1	**Exemplo de saída 1**
3 3	16

Exemplo de entrada 2	**Exemplo de saída 2**
3 2	16

Problema L
LLMs

Bruno recentemente começou a se interessar por grandes modelos de linguagem (os chamados LLMs, sigla do termo em inglês, *large language models*). Uma das primeiras coisas que ele aprendeu foi que um dos componentes mais fundamentais das LLMs é o sistema de predição de *tokens* (que, por simplicidade, consideraremos como palavras). A ideia deste tipo de sistema é relativamente simples de explicar: dada uma frase "incompleta", o sistema deve sugerir a próxima palavra da frase. Tais sistemas se baseiam fortemente em álgebra linear e redes neurais, necessitam de grandes bases de dados de treinamento e possuem no mínimo milhões (muitas vezes bilhões ou até trilhões) de parâmetros, o que torna inviável, para um indivíduo, treinar seu próprio modelo.

Por conta destas limitações (e também pelo fato de não precisar de um LLM em todo seu potencial), Bruno resolveu testar novas ideias, potencialmente mais baratas em termos computacionais, que precisem de menos dados de entrada e que até eliminem a fase de treinamento. Ele não espera obter algo tão bom quanto os modelos comerciais, mas acha que pode encontrar um modelo muito mais barato e suficientemente bom. Ele precisa de ajuda para testar o sistema de predição de palavras que acabou de conceber.

Seu sistema, batizado de Sistema Bruno de Chat ou, simplesmente, SBC, começa de forma similar aos LLMs tradicionais: ele recebe como entrada um mapeamento de um conjunto de palavras, chamado de "dicionário", em um espaço euclidiano, que de alguma forma tende a codificar palavras similares em pontos próximos. Este dicionário vem ordenado da palavra mais comum para a menos comum, no conjunto de dados utilizado para construí-lo. Por simplicidade, Bruno fez uma projeção deste espaço em um plano e arredondou as coordenadas, de forma que cada palavra p seja representada por um ponto (ou vetor) $v(p)$ com coordenadas inteiras no $\mathbb{R}^2$. Além deste mapeamento, Bruno utilizará um texto de entrada como sua "base de conhecimento", ou seja, a fonte a partir da qual suas perguntas serão respondidas.

Para cada consulta, ele tentará prever a próxima palavra. Para isso, ele olhará as últimas palavras da consulta e, utilizando o mapeamento e o texto, fará a previsão de qual palavra deve ser utilizada.

O sistema de Bruno funciona da seguinte forma. Inicialmente, é escolhido um inteiro K, que será o tamanho da "janela de contexto". Para as K últimas palavras da consulta, buscaremos, na base de conhecimento, onde estas palavras ocorrem no texto, exatamente na mesma ordem e de forma consecutiva. Em cada ocorrência, registramos a palavra c que ocorre na base de conhecimento logo após as K últimas palavras. Após repetir este processo por todo o texto, obtemos uma sequência $c_1, \ldots, c_r$ de palavras, que chamaremos de "candidatas".

Em seguida, para cada palavra d do dicionário, é calculada a similaridade de d com as candidatas. Como as palavras estão associadas a vetores, e o produto interno de dois vetores pode ser interpretado como uma medida de similaridade entre eles, Bruno decidiu utilizá-lo como métrica de similaridade. Por abuso de notação, identificamos cada palavra com seu vetor: $d \equiv v(d)$ e $c_i \equiv v(c_i)$. O produto interno de dois vetores $v = (v_x, v_y)$ e $w = (w_x, w_y)$ é denotado por $v \cdot w$ e é definido como $v \cdot w = v_x w_x + v_y w_y$. A similaridade de uma palavra d do dicionário com as candidatas é dada por

$$S(d) = \sum_{i=1}^{r} d \cdot c_i.$$

O SBC então escolhe a palavra com maior valor de $S(d)$ e esta será a próxima palavra do texto. Em caso de empate, o SBC escolherá a palavra mais comum, ou seja, a que aparece primeiro no dicionário. Caso não haja nenhuma palavra candidata, o valor de K é diminuído de uma unidade e o processo é feito novamente. Caso nem mesmo para $K = 1$ haja palavras candidatas, o processo é abortado.

Muito embora na intuição de Bruno todo este processo faça sentido, ele tem muita dificuldade para implementá-lo e precisa de sua ajuda.

Entrada

A entrada contém 3 partes: a primeira descrevendo o dicionário, a segunda descrevendo a base de conhecimento e a terceira contendo as consultas.

A primeira linha da entrada contém um inteiro N ($2 \leq N \leq 10^3$), indicando o número de palavras do dicionário. A i-ésima das N linhas seguintes conterá uma palavra P_i composta apenas por letras minúsculas do alfabeto, com no máximo 12 caracteres, seguida de dois números inteiros X_i e Y_i ($-10^3 \leq X_i, Y_i \leq 10^3$), que indicam o vetor (X_i, Y_i) associado à i-ésima palavra do dicionário.

A linha seguinte contém um inteiro M ($2 \leq M \leq 10^3$), o número de palavras no texto correspondente à base de conhecimento. As linhas seguintes conterão a base de conhecimento, com 8 palavras por linha, separadas por espaço (exceto, possivelmente, na última linha).

Em seguida, dois inteiros Q ($1 \leq Q \leq 10$) e K ($1 \leq K \leq 5$), indicando o número de consultas e o tamanho da janela de contexto. Cada uma das linhas seguintes descreverá uma consulta. Cada consulta conterá um inteiro F ($K \leq F \leq 8$), seguido de F palavras.

Saída

Para cada consulta, deverá ser impressa uma linha contendo as palavras da consulta seguidas da próxima palavra prevista pelo SBC, se existir, ou seguidas de "*" (asterisco), caso contrário.

Exemplo de entrada 1	**Exemplo de saída 1**
6 the -15 0 world 7 6 star -4 2 wars 10 12 peace 10 -11 trek 13 1 12 the peace the war the trek the star star wars star peace 4 3 3 i love star 3 this is the 4 star trek is very 3 star star star	i love star trek this is the peace star trek is very * star star star wars

Problema M

Muralhas reforçadas

Devido à sua localização privilegiada, com um relevo bem favorável, geralmente apenas muralhas frontais eram necessárias para a proteção de cidades medievais em Minas Gerais. Ainda hoje, é possível encontrar vestígios dessas construções ao admirar o belo horizonte da região.

Cada uma dessas muralhas era composta por um número de segmentos consecutivos, cada um com uma altura inicial medida em unidades, correspondendo ao número de blocos usados para construí-lo.

De tempos em tempos, os construtores reforçavam as muralhas escolhendo um segmento e empilhando blocos extra em um padrão de escada: o segmento escolhido recebia K blocos adicionais, o anterior $K-1$, e assim sucessivamente até que apenas 1 bloco fosse adicionado ou que não houvesse mais segmentos à esquerda.

A solidez da defesa era determinada pela altura do segmento mais baixo da muralha.

Dada a descrição inicial de uma muralha, seu objetivo é determinar qual é a *maior altura mínima possível* após a aplicação de um único reforço.

Entrada

A primeira linha contém dois inteiros N ($1 \leq N \leq 10^5$), o número de segmentos da muralha, e K ($1 \leq K \leq N$), o número de blocos adicionados ao segmento escolhido.

A segunda linha contém N inteiros $x_1, x_2, \ldots, x_N$ ($1 \leq x_i \leq 10^9$), representando as alturas iniciais dos segmentos.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro: a maior altura mínima possível da muralha após um único reforço.

Exemplo de entrada 1	**Exemplo de saída 1**
5 5 5 4 3 2 1	6

Exemplo de entrada 2	**Exemplo de saída 2**
6 1 3 3 1 3 3 3	2

Exemplo de entrada 3	**Exemplo de saída 3**
5 5 3 4 7 8 7	7

Explicação do exemplo 3:

Se reforçarmos o primeiro segmento, a muralha passa a ter alturas $[8, 4, 7, 8, 7]$. O menor segmento nesse caso tem altura 4.

Reforçando o último segmento, temos $[4, 6, 10, 12, 12]$, e o menor valor é 4.

Entre todas as opções, a melhor escolha é reforçar o segundo segmento, obtendo $[7, 9, 7, 8, 7]$, cujo valor mínimo é 7.

Portanto, a maior altura mínima possível é 7.