

Ogro

Nome do arquivo: `ogro.c`, `ogro.cpp`, `ogro.java`, `ogro.js` ou `ogro.py`

Ogro e Bicho-Papão têm fama de malvados, mas na verdade são amáveis, honestos e trabalhadores, além de vizinhos e amigos. O Bicho-Papão tem dificuldades em aprender aritmética e por isso o Ogro inventou uma brincadeira simples para auxiliar seu amigo: o Ogro inicia mostrando um certo número de dedos na sua mão esquerda (vamos chamar esse valor de E) e um número de dedos diferente na mão direita (vamos chamar esse valor de D). Então, Bicho-Papão deve falar o *resultado* da brincadeira, definido assim:

- se o número de dedos na mão esquerda é maior do que o número de dedos na mão direita (ou seja $E > D$) então o resultado é a soma dos dois números (ou seja $E + D$);
- caso contrário, o resultado é o dobro da diferença entre o número de dedos na mão direita e o número de dedos na mão esquerda (ou seja, $2 \times (D - E)$).

O problema é que o Ogro também não é lá muito bom em aritmética, e pediu sua ajuda para conferir se o Bicho-Papão falou a resposta correta.

Dados o número de dedos mostrados na mão esquerda (E) e o número de dedos mostrados na mão direita (D), escreva um programa para determinar a resposta da brincadeira.

Entrada

A entrada é composta por duas linhas. A primeira linha contém um inteiro E , o número de dedos mostrados na mão esquerda. A segunda linha contém um inteiro D , o número de dedos mostrados na mão direita.

Saída

Seu programa deve produzir uma única linha na saída, contendo um único número inteiro, o resultado da brincadeira.

Restrições

- $0 \leq E \leq 5$
- $0 \leq D \leq 5$
- $E \neq D$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima.

- **Subtarefa 1 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (30 pontos):** $E > D$.
- **Subtarefa 3 (70 pontos):** Sem restrições adicionais.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima (*elas não precisam ser resolvidas em ordem*). Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1 1 0	Exemplo de saída 1 1
Exemplo de entrada 2 2 5	Exemplo de saída 2 6

Concurso

Nome do arquivo: `concurso.c`, `concurso.cpp`, `concurso.java`, `concurso.js` ou `concurso.py`

Cláudia trabalha na OBI (Organização dos Bons Informáticos), que recentemente realizou um concurso para contratar novos funcionários. Agora, Cláudia tem a tarefa de determinar a *nota de corte* para o concurso. Chamamos de nota de corte a nota mínima necessária para ser aprovado no concurso. Ou seja, se a nota de corte do concurso for C , então todos os participantes com uma nota maior ou igual a C serão aprovados no concurso e todos com nota menor que C serão reprovados.

Seu chefe pediu para que Cláudia aprove no mínimo K candidatos do concurso para a próxima fase, mas ela também não quer que a nota de corte seja muito baixa. Por isso, Cláudia decidiu que a nota de corte deverá ser a maior nota C que faz com que no mínimo K candidatos sejam aprovados.

Sua tarefa é: dados o número N de candidatos, as notas A_1, A_2, \dots, A_N dos candidatos e a quantidade mínima de aprovados K , diga qual deve ser a maior nota de corte C para que pelo menos K candidatos sejam aprovados.

Entrada

A primeira linha da entrada contém dois inteiros, N e K , representando, respectivamente, o número de participantes e o número mínimo de candidatos que devem ser aprovados.

A segunda linha da entrada contém N inteiros A_i , representando as notas dos participantes.

Saída

Seu programa deve imprimir uma linha contendo um único inteiro C , a nota de corte que deve ser escolhida por Cláudia.

Restrições

- $1 \leq K \leq N \leq 500$
- $1 \leq A_i \leq 100$ para todo $1 \leq i \leq N$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima.

- **Subtarefa 1 (0 pontos):** Esta sub tarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (20 pontos):** $K = 1$.
- **Subtarefa 3 (20 pontos):** $K = 3$.
- **Subtarefa 4 (20 pontos):** $A_i \leq 2$.
- **Subtarefa 5 (40 pontos):** Sem restrições adicionais.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima (*elas não precisam ser resolvidas em ordem*). Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1 3 1 92 83 98	Exemplo de saída 1 98
Exemplo de entrada 2 4 2 1 2 3 4	Exemplo de saída 2 3
Exemplo de entrada 3 5 3 20 20 10 20 30	Exemplo de saída 3 20
Exemplo de entrada 4 10 5 1 2 2 1 2 2 1 1 1 1	Exemplo de saída 4 1

Placas de Carro

Nome do arquivo: `placas.c`, `placas.cpp`, `placas.java`, `placas.js` ou `placas.py`

As placas usadas nos carros em circulação no Brasil possuem dois padrões com formatos diferentes: algumas placas estão no antigo padrão Brasileiro, enquanto outras estão no novo padrão Mercosul.

O antigo padrão Brasileiro é sempre formado por 8 caracteres:

- os três primeiros caracteres são letras maiúsculas (de A a Z);
- o quarto caractere é um hífen (-);
- os últimos quatro caracteres são dígitos (de 0 a 9).

Por exemplo, `OBI-2024` é uma placa válida no antigo padrão Brasileiro.

O novo padrão Mercosul, por sua vez, é sempre formado por 7 caracteres:

- os três primeiros caracteres são letras maiúsculas;
- o quarto caractere é um dígito;
- o quinto caractere é uma letra maiúscula;
- os últimos dois caracteres são dígitos.

Assim, `OBI2P24` é uma placa válida no novo padrão Mercosul.

Há também um grande contingente de carros em situação irregular – carros com placas falsificadas que não estão nem no antigo padrão Brasileiro, nem no novo padrão Mercosul. Por exemplo, um carro com a placa `OBI-24` está em situação irregular, pois a placa não é válida em nenhum dos dois padrões.

O Departamento Nacional de Trânsito identificou que seus funcionários gastam muito tempo verificando manualmente quais placas estão em qual padrão e quais são falsificadas. Por isso, eles pediram sua ajuda para automatizar o processo: dada uma placa formada por uma sequência de letras maiúsculas, dígitos e hífens, determine se a placa está no antigo padrão Brasileiro, está no novo padrão Mercosul, ou é uma placa falsificada.

Entrada

A entrada é composta de uma única linha, contendo uma sequência de caracteres representando a placa a ser analisada.

Saída

Seu programa deverá imprimir uma linha contendo um único número inteiro:

- 1, se a placa está no antigo padrão Brasileiro;
- 2, se a placa está no novo padrão Mercosul;
- 0, se a placa é falsificada.

Restrições

- A placa possui entre 6 e 10 caracteres.
- Cada caractere da placa é uma letra maiúscula (de A a Z), um dígito (de 0 a 9) ou um hífen (o caractere -).

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima.

- **Subtarefa 1 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (40 pontos):** É garantido que a placa está ou no antigo padrão Brasileiro, ou no novo padrão Mercosul (ou seja, ela não é falsificada).
- **Subtarefa 3 (60 pontos):** Sem restrições adicionais.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima (*elas não precisam ser resolvidas em ordem*). Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1 OBI-2024	Exemplo de saída 1 1
Exemplo de entrada 2 OBI2P24	Exemplo de saída 2 2
Exemplo de entrada 3 OBI-24	Exemplo de saída 3 0
Exemplo de entrada 4 XYZ-1234	Exemplo de saída 4 1
Exemplo de entrada 5 A1B2C3D4E5	Exemplo de saída 5 0

Jogo da Vida

Nome do arquivo: `jogo.c`, `jogo.cpp`, `jogo.java`, `jogo.js` ou `jogo.py`

O Jogo da Vida de Conway é um processo de simulação (conhecido como *autômato celular*) criado pelo matemático britânico John Conway para reproduzir, por meio de uma matriz, processos de mudança em grupos de seres vivos. As regras do jogo indicam como a matriz é modificada a cada passo. Os valores da matriz em um determinado passo são coletivamente chamados de *estado* do jogo.

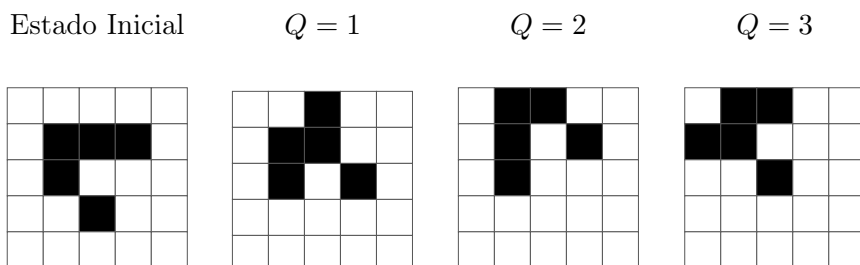
Mais especificamente, o jogo acontece em uma matriz quadrada $N \times N$ (ou seja, com N linhas e N colunas) no qual cada célula está viva (representada pelo número 1) ou morta (representada pelo número 0). Para simular o próximo estado do autômato, para cada célula calculamos o seu número de vizinhos vivos (duas células são consideradas vizinhas se elas são adjacentes diagonalmente, horizontalmente ou verticalmente – ou seja, uma célula pode ter até 8 vizinhas), e decidimos se a célula estará viva ou morta no próximo estado de acordo com as seguintes regras:

- se uma célula morta possui exatamente três vizinhas vivas, ela vira uma célula viva;
- se uma célula morta possui uma quantidade de vizinhas vivas diferente de três, ela continua morta;
- se uma célula viva possui duas ou três vizinhas vivas, ela continua viva;
- se uma célula viva possui menos que duas vizinhas vivas, ela morre;
- se uma célula viva possui mais que três vizinhas vivas, ela morre.

Toda célula fora da matriz é considerada morta, ou seja, células fora da matriz nunca afetam a quantidade de vizinhos vivos de alguma célula. Observe que as regras são aplicadas em todas as células simultaneamente, uma vez a cada passo.

Dada uma matriz que representa o estado inicial do jogo e um inteiro positivo Q , sua tarefa é determinar o Q -ésimo estado do jogo de acordo com as regras descritas acima, ou seja, o valor de cada célula da matriz após Q passos do jogo.

A figura abaixo mostra um exemplo de jogo em uma matriz 5×5 e seus estados para diferentes valores de Q . Células vivas são representadas com a cor preta e células mortas são representadas com a cor branca.



Entrada

A primeira linha contém dois números inteiros, N e Q , representando, respectivamente, o número de linhas/colunas da matriz e o número de passos a serem simulados.

As próximas N linhas contêm N caracteres cada. O j -ésimo caractere da i -ésima linha representa o estado inicial da célula na linha i e coluna j . Caso o caractere seja '0', a célula naquela posição inicia o jogo morta; caso o caractere seja '1', a célula inicia o jogo viva.

Saída

O seu programa deverá imprimir N linhas, cada uma contendo N caracteres. Na i -ésima linha, o j -ésimo caractere deve representar o Q -ésimo estado da célula na linha i e coluna j . Caso a célula esteja morta, o caractere deve ser '0'; se ela estiver viva, o caractere deve ser '1'.

Restrições

- $1 \leq N \leq 50$
- $1 \leq Q \leq 100$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima.

- **Subtarefa 1 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (30 pontos):** $Q = 1$.
- **Subtarefa 3 (70 pontos):** Sem restrições adicionais.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima (*elas não precisam ser resolvidas em ordem*). Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1	Exemplo de saída 1
5 3	01100
00000	11000
01110	00100
01000	00000
00100	00000
00000	

<p>Exemplo de entrada 2</p> <pre> 15 1 000010000010000 000010000010000 000011000110000 000000000000000 111001101100111 001010101010100 000011000110000 000000000000000 000011000110000 001010101010100 111001101100111 000000000000000 000011000110000 000010000010000 000010000010000 </pre>	<p>Exemplo de saída 2</p> <pre> 000000000000000 000110000011000 000011000110000 010010101010010 011101101101110 001010101010100 000111000111000 000000000000000 000111000111000 001010101010100 011101101101110 010010101010010 000011000110000 000110000011000 000000000000000 </pre>
<p>Exemplo de entrada 3</p> <pre> 15 3 000010000010000 000010000010000 000011000110000 000000000000000 111001101100111 001010101010100 000011000110000 000000000000000 000011000110000 001010101010100 111001101100111 000000000000000 000011000110000 000010000010000 000010000010000 </pre>	<p>Exemplo de saída 3</p> <pre> 000010000010000 000010000010000 000011000110000 000000000000000 111001101100111 001010101010100 000011000110000 000000000000000 000011000110000 001010101010100 111001101100111 000000000000000 000011000110000 000010000010000 000010000010000 </pre>