2022
Caderno de Problemas

IX Maratona Mineira de Programação

Patrocínio

Algar Telecom — ALPHABOT — edgeuno — Meta

<ambevtech/> — BITKA ANALYTICS — dti digital crafters — netlex

Colaboração

CEFET-MG CAMPUS TIMÓTEO

Realização

SBC Sociedade Brasileira de Computação

Informações gerais

Este caderno de tarefas é composto por 25 páginas (não contando a folha de rosto), numeradas de 1 a 25. Verifique se o caderno está completo.

Nome do programa

Cada problema tem os possíveis nomes de arquivo fonte indicados abaixo do título. Soluções na linguagem C devem ser arquivos com sufixo *.c*; soluções na linguagem C++ devem ser arquivos com sufixo *.cc* ou *.cpp*; soluções na linguagem Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; e soluções na linguagem Python devem ser arquivos com sufixo *.py*.

Entrada

- A entrada deve ser lida da entrada padrão.
- A entrada consiste em exatamente um caso de teste, que é descrito usando uma quantidade de linhas que depende do problema. O formato da entrada é como descrito em cada problema. A entrada não contém nenhum conteúdo extra.
- Todas as linhas da entrada, incluindo a última, terminam com o caractere de fim de linha (\n).
- A entrada não contém linhas vazias.
- Quando a entrada contém múltiplos valores separados por espaços, existe exatamente um espaço em branco entre dois valores consecutivos na mesma linha.

Saída

- A saída deve ser escrita na saída padrão.
- A saída deve respeitar o formato especificado no enunciado. A saída não deve conter nenhum dado extra.
- Todas as linhas da saída, incluindo a última, devem terminar com o caractere de fim de linha (\n).
- Quando uma linha da saída apresentar múltiplos valores separados por espaços, deve haver exatamente um espaço em branco entre dois valores consecutivos.
- Quando a um valor da saída for um número real, use pelo menos o número de casas decimais correspondente à precisão requisitada no enunciado.

Problema A. Descriptografando

Nome do arquivo fonte: `itens.c`, `itens.cpp`, *ou* `itens.java`

Alice e Bob resolveram dar um tempo da criptografia. Enquanto estavam aproveitando as merecidas férias em uma das belas cachoeiras de Minas Gerais, eles conheceram um ex-competidor da Maratona Mineira de Programação, que explicou em detalhes como a competição funcionava. Infelizmente, Alice e Bob não são mais elegíveis para competir. Porém, eles queriam muito participar de alguma forma.

Bob pensou em propor problemas para a prova, mas Alice rapidamente percebeu que não seria uma boa ideia pois os competidores saberiam que se trataria de um problema de criptografia assim que vissem os nomes dos autores. Então, sugeriram uma forma de deixar a competição ainda mais interessante. Toda vez que algum time resolver um problema, o time vai receber uma notificação no computador e, alguns segundos depois, o balão do problema amarrado a um peso será solto do alto em algum ponto do teto do salão onde a prova estiver sendo realizada. Para poderem ficar com o balão, um dos membros do time tem que conseguir pegar o balão antes que ele caia no chão.

Os organizadores da prova acharam a ideia muito interessante, e estão pensando em implementá-la nas próximas edições. Antes disso, precisam avaliar cuidadosamente todos os riscos e implicações. Por exemplo, algum competidor correndo desesperado para pegar um balão poderia tropeçar em um cabo de rede. Então, talvez seja melhor usar uma rede sem fio. Outra questão importante é o impacto na felicidade dos competidores. Os organizadores acreditam que felicidade dos competidores está ligada ao número de balões que eles levam pra casa. Por isso, resolveram fazer uma análise para determinar a probabilidade de um competidor conseguir pegar um balão.

O salão da prova foi modelado como um tabuleiro $N \times M$. Cada célula pode estar vazia ou conter algum obstáculo como, por exemplo, uma mesa. O balão é solto diretamente acima de uma célula vazia escolhida aleatoriamente com probabilidade uniforme, e leva T segundos para cair. O competidor se encontra na célula da linha c_i e coluna c_j e consegue se deslocar verticalmente ou horizontalmente para células vizinhas vazias. O competidor leva 1 segundo para se mover de uma célula para outra. Se o competidor chegar em uma célula ao mesmo tempo que o balão cair, ele consegue pegar o balão. Como os competidores são muito espertos, se existir mais de um caminho que permite que o competidor pegue o balão, ele vai escolher o menor deles. A organização quer saber qual é probabilidade de que o competidor consiga pegar o balão antes que ele caia. Como a organização estava muito ocupada crimpando cabos de rede, enchendo balões e comprando salgados, resolveram pedir para que você calcule essa probabilidade, o que deixou Bob muito feliz pois agora ele ajudou a colocar um problema na prova.

Entrada

A primeira linha da entrada contém cinco inteiros separados por espaço N, M, T, c_i e c_j, onde N e M representam respectivamente o número de linhas e colunas do grid, T representa o tempo em segundos que o balão leva para cair, e c_i e c_j representam respectivamente a linha e a coluna do grid da célula onde o competidor se encontra. Cada uma das N linhas seguintes contém M caracteres representando a i-ésima linha do grid. O j-ésimo caractere da i-ésima dessas linhas representa a célula do grid na linha i e coluna j. O caractere '.' é usado para representar uma célula vazia e o caractere '#' para representar uma célula ocupada.

Saída

A saída deve conter um única linha com dois inteiros A e B separados por espaço, de forma que $\frac{A}{B}$ represente a probabilidade desejada, $B > 0$ e o maior fator comum entre A e B seja 1.

Restrições

- $1 \leq N \leq 1000$.
- $1 \leq M \leq 1000$.
- $1 \leq T \leq 10^6$.
- $1 \leq c_i \leq N$.
- $1 \leq c_j \leq M$.

Exemplos

Entrada	**Saída**

`4 5 7 1 1`
`.#.#.`
`...#.`
`##...`
`...#.`
	`13 14`

Entrada	**Saída**

`10 5 100 3 2`
`#####`
`#.#.#`
`#.#.#`
`#.#.#`
`#.#.#`
`#.#.#`
`#.#.#`
`#.#.#`
`#.#.#`
`#####`
	`1 2`

Problema B. Festa Laser

Nome do arquivo fonte: `laser.c`, `laser.cpp`, *ou* `laser.java`

Maria Joaquina, ou pela alcunha Majô, é uma jovem muito criativa e interessada em ciência. Desde criança, ela aprendeu que o conhecimento racional que a humanidade adquiriu ao longo da história permitiu com que a qualidade de vida melhorasse, e que descobertas ainda maiores fossem feitas. "Coitado dos negacionistas!", sempre diz ela.

Um dos feitos mais inspiradores para Majô foi a expedição à Lua de 1969, resultado da cooperação de matemáticos, engenheiros e cientistas de várias áreas do conhecimento. Após muito trabalho, o foguete Apollo 11 da NASA levou astronautas de uma base na Terra por uma viagem de 8 dias, com partes muito críticas, como a decolagem, a aproximação e pousos lunares, a operação do módulo lunar, e a reentrada na atmosfera terrestre.

Nessa viagem, os astronautas Michael Collins, Buzz Aldrin e Neil Armstrong deixaram um arranjo de espelhos no solo lunar, capaz de refletir raios lasers emitidos a partir do planeta Terra. Quando soube disso, Majô imediatamente começou a montar o equipamento apropriado em seu quintal e chamou todos os seus vizinhos para acompanharem o experimento durante a festa que ela titulou "AFE – Avoid Flat Earthers".

Além de conversar sobre astronomia, Majô, amante do estoicismo, também conversará com os convidados sobre outros avanços da humanidade, como o pensamento filosófico, o método científico e as vacinas. Entretanto, para sua surpresa, ela descobriu que vários vizinhos não acreditam na chegada da humanidade à Lua ou em algum avanço da ciência, e vão só causar transtornos em sua festa laser.

Para evitar a fadiga, a jovem Joaquina pediu que você escreva um programa que indica se uma pessoa deve ser cortada do evento, por ser negacionista ("Óbvio, né gente!", como diz Majô). Neste programa, o vizinho deve informar se ele é negacionista ou não. Se for um pessoa sensata, com certeza será convidada.

Entrada

A entrada possui uma linha contendo um valor inteiro V, que vale 1, quando o vizinho analisado é negacionista, e 0 caso contrário.

Saída

A saída possui uma linha com um valor inteiro A, que vale 1 quando o vizinho deve ser convidado por Majô para a festa AFE, de acordo com o critério que ela adota, e 0 caso contrário.

Restrições

- $0 \leq V \leq 1$.

Exemplos

Entrada	**Saída**
1	0

Entrada	**Saída**
0	1

Problema C. Caindo na estrada

Nome do arquivo fonte: `artista.c`, `artista.cpp`, *ou* `artista.java`

Pedro é um artista pop que anda fazendo muito sucesso no Brasil. Suas músicas estão em todas as rádios e os cachês de seus shows estão cada vez mais altos! Ele está realmente no topo de sua carreira.

Pedro está planejando uma grande turnê que irá rodar o mundo todo, porém está com dificuldade em definir os dias dos seus shows. O valor do cachê do Pedro varia diariamente (depende do dia da semana, mês, previsão do tempo do dia, etc...). Ele quer organizar os shows de modo que seu cachê seja o mais alto possível, porém ele não quer fazer shows em dias consecutivos, pois isso o deixaria muito cansado e ele não conseguiria dar o seu melhor. Você pode ajudá-lo com isso?

Dada a quantidade de dias que durará a turnê e o valor do cachê de cada dia, ajude Pedro a encontrar o maior valor possível que ele pode receber somados todos os shows, lembrando que Pedro não pode realizar shows em dias consecutivos.

Entrada

A primeira linha da entrada contem um inteiro N, a quantidade de dias que durará a turnê. Cada uma das próximas N linhas seguintes contém um inteiro $K[i]$, representando o valor do cachê de Pedro no dia i.

Saída

A saída deve conter um único inteiro X, o valor máximo do cachê que Pedro pode receber na turnê.

Restrições

- $1 \leq N \leq 10^5$.
- $1 \leq K[i] \leq 10^9$.

Exemplos

Entrada	**Saída**
`10` `5 2 3 10 1 8 9 15 3 4`	`42`

Entrada	**Saída**
`5` `1 8 15 11 2`	`19`

Problema D. Crise da Chuva

Nome do arquivo fonte: `colapso.c`, `colapso.cpp`, *ou* `colapso.java`

Bibika trabalha no centro nacional de meteorologia onde sua função é estudar dados passados referentes a períodos chuvosos em diversas cidades a fim de entender como cidades em estados de emergência afetam para o crescimento do país.

Para fins didáticos, iremos considerar que um país é uma grade retangular de tamanho NxM onde cada posição desse retângulo representa uma cidade. Após anos de estudos, Bibika está quase certa de que um país entra em colapso, por conta dos problemas em longos períodos de chuva, sempre que todas as cidades contidas em um quadrado de lado K declaram estado de emergência.

Querendo confirmar sua teoria, Bibika conta com sua ajuda para analisar mais alguns períodos chuvosos da história e descobrir qual foi o primeiro momento em que o país entrou em colapso, dada a definição de colapso descrita anteriormente.

É válido destacar que uma cidade nunca entra em estado de emergência mais de uma vez e, uma vez em estado de emergência, ela permanece nesse estado para sempre.

Entrada

A primeira linha da entrada contém quatro inteiros N, M, K, Q, representando, respectivamente, a quantidade de linhas e colunas do país, o tamanho do lado do quadrado e a quantidade de dados de cidades que declararam estado de emergência. Cada uma das próximas Q linhas contém três inteiros A, B, D significando que no momento D no tempo a cidade da posição (A, B) declarou estado de emergência.

Saída

A saída deve conter um único inteiro: o primeiro instante em que seria possível dizer que o país entrou em colapso. Caso ele não tenha entrado em colapso em nenhum momento, exiba -1.

Restrições

- $1 \leq N, M \leq 400$.
- $1 \leq K \leq min(N, M)$.
- $1 \leq Q \leq N \times M$.
- $1 \leq A \leq N$.
- $1 \leq B \leq M$.
- $1 \leq D \leq 10^9$.

Exemplos

Entrada	**Saída**
4 3 2 9 1 1 6 1 2 5 4 2 4 3 1 1 1 3 4 2 2 3 3 2 8 2 3 7 4 3 1	7

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

Problema E. Drawdown

Nome do arquivo fonte: `drawdown.c`, `drawdown.cpp`, *ou* `drawdown.java`

Investir na bolsa de valores pode ser uma grande oportunidade para obter bons rendimentos. Um tipo importante de investidor que vem ganhando cada vez mais destaque nos últimos anos são os chamados fundos quantitativos ou sistemáticos, como a Alphabot. Nesses fundos, os investimentos são realizados de forma 100% automatizada, fazendo uso de grandes bases de dados, modelagens matemáticas, simulações estatísticas, algoritmos de aprendizado de máquina, otimização e implementações computacionais de alta performance para identificar, validar, desenvolver e executar estratégias de investimento automatizadas que não seriam possíveis de serem criadas e operacionalizadas manualmente por humanos.

Como qualquer investimento na bolsa, os riscos, assim como os retornos, podem ser altos. Isso torna essencial um rigoroso gerenciamento de risco, em especial nos fundos quantitativos, devido ao alto volume financeiro e frequência de suas negociações. Pensando nisso, a Alphabot utiliza dezenas de métricas de risco para analisar e gerenciar seu portfólio de estratégias quantitativas, dentre as quais destacam-se o *drawdown* e o *drawdown percentual*.

Seja $P_1, P_2, \ldots, P_N$ a sequência de preços de um ativo em N instantes de tempo. O *drawdown* entre os instantes i ($1 \leq i \leq N$) e j ($i \leq j \leq N$) é definido como a queda de preço do instante i para o instante j e pode ser calculado pela seguinte fórmula:

$$\text{drawdown}\,(i,j) = \begin{cases} P_i - P_j, \text{ se } P_j < P_i \\ 0, \text{ caso contrário.} \end{cases}$$

O *drawdown percentual* entre dois instantes de tempo i ($1 \leq i \leq N$) e j ($i \leq j \leq N$) é definido como a razão entre o *drawdown* entre i e j e o preço do ativo no instante i. Isto é

$$\text{drawdown\_percentual}\,(i,j) = \frac{\text{drawdown}\,(i,j)}{P_i}.$$

Por exemplo, considere um ativo com o seguinte histórico de preços:

$$P_1 = 100, P_2 = 120, P_3 = 80, P_4 = 60, P_5 = 70.$$

O *drawdown* e o *drawdown percentual* entre os instantes 2 e 4 são 60 e 50%, respectivamente.

O *drawndown percentual* entre os instantes i e j representa qual porcentagem do capital investido teria sido perdido caso o ativo tivesse sido comprado no instante i e vendido no instante j. No exemplo acima, um investidor que tivesse comprado 10 unidades do ativo no instante 2 por um preço total de 1200 e vendido no instante 4 por um preço total de 600, teria perdido 600. Ou seja, teria perdido 50% dos 1200 investidos inicialmente.

Uma forma de avaliar o risco de um ativo em um intervalo de tempo entre os instantes a ($1 \leq a \leq N$) e b ($a \leq b \leq N$) é calcular os maiores valores de *drawdown* e *drawdown percentual* nesse intervalo de tempo. Mais precisamente, temos que:

$$\text{max\_drawdown}\,(a,b) = \max\{\text{drawdown}\,(i,j) : a \leq i \leq j \leq b\}$$

$$\text{max\_drawdown\_percentual}\,(a,b) = \max\{\text{drawdown\_percentual}\,(i,j) : a \leq i \leq j \leq b\}$$

Dada a sequência de preços de um ativo investido pela Alphabot e vários intervalos de tempo, calcule o *drawdown máximo* e o *drawdown percentual máximo* para cada intervalo.

Entrada

A primeira linha da entrada contém um inteiro N, que representa o número de instantes de tempo na sequência de preços. A segunda linha da entrada contém N inteiros positivos $P_1, P_2, \ldots, P_N$, onde P_i representa o preço do ativo no i-ésimo instante de tempo. A terceira linha da entrada contém um inteiro Q, representando o número de intervalos de tempo para os quais o *drawdown máximo* e o *drawdown percentual máximo* devem ser calculados. A $j-$ésima das Q linhas seguintes contém dois inteiros a_j e b_j separados por espaço, representado o intervalo de tempo entre os instantes a_j e b_j, inclusive.

Saída

A saída deve conter exatamente Q linhas. A $j-$ésima linha da saída deve conter três inteiros não-negativos separados por espaço D, R e S, onde D e $\frac{R}{S}$ representam o *drawdown máximo* e o *drawdown percentual máximo* entre a_j e b_j, respectivamente. Também é preciso garantir que o maior divisor comum entre R e S seja 1.

Restrições

- $1 \leq N \leq 5 \times 10^5$.
- $1 \leq P_i \leq 10^9$ para todo $1 \leq i \leq N$.
- $1 \leq Q \leq 5 \times 10^5$.
- $1 \leq a_j \leq b_j \leq N$ para todo $1 \leq j \leq Q$.

Exemplos

Entrada	**Saída**
6 100 40 200 150 100 10 1 1 5	100 3 5

Entrada	**Saída**
2 100 100 2 1 2 1 1	0 0 1 0 0 1

Problema F. Caindo de Paraquedas

Nome do arquivo fonte: `mania.c`, `mania.cpp`, *ou* `mania.java`

Ernesto é um grande paraquedista que já acumula mais de 5000 saltos em sua carreira. Ele já se jogou de diferentes aviões, helicópteros, balões, já pousou no deserto, na praia, na floresta e em uma variedade de outros locais.

Ultimamente ele está com uma mania um tanto quanto diferente, antes de se jogar do avião ele decide que irá fazer apenas curvas de G graus, seja para esquerda ou para direita, enquanto estiver navegando seu paraquedas.

Dado o valor G e as direções das curvas (`'E'` = esquerda, `'D'` = direita) feitas por Ernesto, deve-se assumir que seu paraquedas não muda de direção (a não ser que ele faça propositalmente uma curva) e ele está sempre olhando fixamente para frente.

É possível dizer que Ernesto teve uma visão completa ao seu redor (360 graus) durante sua navegação?

Entrada

A primeira linha contém um inteiro G e a segunda linha uma string de tamanho S (composta apenas pelos caracteres `'D'` ou `'E'`) representando a sequência de curvas feitas por Ernesto.

Saída

Exiba o caractere `'S'` caso Ernesto tenha tido uma visão completa ao seu redor durante sua navegação ou `'N'` caso contrário.

Restrições

- $1 \leq G \leq 360$.
- $1 \leq |S| \leq 10^5$.

Exemplos

Entrada	**Saída**
90 DDEDDDEE	S

Entrada	**Saída**
150 EEDED	N

Problema G. Bário

Nome do arquivo fonte: `bario.c`, `bario.cpp`, *ou* `bario.java`

Tudo começou com uma grande explosão cósmica. Surgiram os quarks, os planetas, as nebulosas, as galáxias, e a Maratona Mineira de Programação. Para a edição de 2022, os organizadores pensaram bastante em como fazer uma super edição do evento para comemorar o seu retorno. Uma das ideias mais ousadas era o desenvolvimento de um jogo de plataforma de um dos personagens mais icônicos do mundo: o Bário.

Nesse novo lançamento, Bário se moveria em fases compostas por N blocos. Cada bloco pode ser sólido ou um buraco, e o objetivo do jogo é chegar do início da fase (o bloco 1) até o final (o bloco N) sem cair em nenhum buraco. Para evitar os buracos, Bário pode pulá-los.

Além disso, ele tem o poder de carregar suas botas de energia para pular buracos muito grandes. Cada bloco sobre o qual Bário corre adiciona uma unidade de carga a seu equipamento. Um pulo de carga v faz com que Bário vá do bloco b para o primeiro bloco sólido entre as posições $b+1$ e $b+v+1$; ao aterrissar, Bário perde toda a energia de suas botinhas, retornando ao valor inicial de 1.

O bloco onde Bário aterrissa e o primeiro bloco da fase não adicionam carga às suas botas.

Como vocês devem ter reparado, não há um novo jogo para vocês se divertirem, só esse problema, que é quase tão legal. O comitê de prova se rebelou e não quis implementar as fases, pois achou muito difícil determinar se uma fase era completável. Resolvam esse problema e talvez, quem sabe, podemos ter uma edição especial de Bário para Mineira de 2023.

Entrada

A primeira linha da entrada contem um único inteiro N, que representa o número de blocos na fase. A segunda linha contém N caracteres. Caracteres x representam blocos sólidos, enquanto caracteres . representam buracos na fase. É garantido que o último bloco da fase é sólido.

Saída

A saída contem um único inteiro P, o menor número de pulos que Bário precisa dar para completar a fase, ou -1 caso seja impossível completar a fase.

Restrições

- $2 \leq N \leq 5 \times 10^5$.

Exemplos

Entrada	**Saída**
`4` `x..x`	`-1`

Entrada	**Saída**
`5` `xx.xx`	`1`

Problema H. Bagunça da Madalena

Nome do arquivo fonte: `bagunca.c`, `bagunca.cpp`, *ou* `bagunca.java`

Madalena é uma cachorrinha que adora fazer bagunça e andar em círculos. Sempre que seu dono, Dâmiko, chega em casa, encontra vários brinquedos espalhados pela casa e a cachorrinha Madalena girando e girando pelos cômodos. Dâmiko adora encontrar padrões e descobriu que os brinquedos não ficam espalhados de forma aleatória:

- Um brinquedo fica sempre em cima de um azulejo e Madalena brinca apenas com os brinquedos do azulejo que ela está no momento.
- Quando Madalena começa a brincar, ela pega o máximo possível de brinquedos do azulejo em que está e distribui de forma igualitária para os azulejos adjacentes. Por exemplo, se existem 20 brinquedos no azulejo atual e 8 azulejos adjacentes, 2 brinquedos irão para cada um deles e 4 continuarão no azulejo atual.
- Após distribuir os brinquedos, Madalena se move para um azulejo adjacente. Ela escolhe aquele que tem mais brinquedos (porque assim é muito mais divertido) e recomeça o processo de distribuí-los. Se existe mais de um possível próximo azulejo com o mesmo número de brinquedos, ela escolhe o primeiro do sentido horário que encontrar (iniciando pelo azulejo à sua esquerda).

A brincadeira acaba quando Madalena passa por $L+C$ azulejos.

Dado o azulejo inicial onde Madalena se encontra, Dâmiko deseja saber a maior quantidade de brinquedos em um único azulejo ao final da brincadeira.

Entrada

A primeira linha da entrada contém dois valores inteiros L e C que representam as quantidade de linhas e colunas de azulejos. Em seguida, L linhas conterão os elementos de uma matriz M de dimensão $L \times C$ com elementos inteiros representando a quantidade de brinquedos em cada um dos azulejos. Por fim, será dada uma última linha contendo dois valores inteiros A e B que representam a linha e a coluna, respectivamente, onde Madalena inicia sua brincadeira.

Saída

A saída deve conter o maior número de brinquedos em um único azulejo ao final da brincadeira.

Restrições

- $1 \leq L, C \leq 500$.
- $0 \leq M_{ij} \leq 10^6$.
- $1 \leq A \leq L$.
- $1 \leq B \leq C$.

Exemplos

Entrada	**Saída**
4 5 0 3 8 0 0 3 1 0 8 4 5 0 0 0 12 3 7 9 4 7 2 3	9

Entrada	**Saída**
3 3 0 0 0 0 9 0 0 0 0 2 2	1

Problema I. Bloomhaven

Nome do arquivo fonte: `bloomhaven.c`, `bloomhaven.cpp`, *ou* `bloomhaven.java`

Cirila adora jogos de tabuleiro, tanto pelas noites de diversão que eles proporcionam para seu grupo de amigos quanto pelos quebra cabeças que ela tem de resolver para jogar de forma mais eficiente possível e vencer.

Sua paixão por eles é tão grande que ela resolveu criar o próprio jogo: Bloomhaven. Rapidamente, ela descobriu que garantir o equilíbrio entre os diferentes componentes é muito difícil, e que resolver vários casos de teste na mão é cansativo e demanda muito tempo.

A última mecânica de jogo que ela precisa implementar é o sistema de feitiços, e Cirila decidiu fazer ele bem diferente de todos os outros que já viu. Ele funciona da seguinte forma:

- Cada personagem possui um disco com N símbolos, e entre dois símbolos consecutivos está um encantamento com um determinado poder.
- Para realizar uma magia, é preciso escolher um subconjunto não vazio de encantamentos, cada encantamento escolhido ativa os dois símbolos adjacentes a ele.
- Para que a magia seja considerada válida, é preciso que:
  1. Cada símbolo seja ativado por no máximo um dos encantamos escolhidos.
  2. Os símbolos ativos formem uma sequência contígua no disco.
- O poder de uma magia é soma dos poderes dos encantamentos escolhidos para realizá-la.

Cirila gostaria de saber qual a magia mais poderosa que pode ser usada pelo personagem, mas não quer gastar mais tempo que o necessário fazendo essas contas.

Escreva um programa que ajude Cirila a resolver esse problema e, quem sabe, terá seu nome no livro de regras com um agradecimento especial!

Entrada

A primeira linha da entrada contém um único inteiro N o número de símbolos no disco do personagem. A segunda linha contém N inteiros, o i-ésimo deles, X_i, representa o poder do encantamento entre os símbolos i e $(i \mod N) + 1$.

Saída

A saída contem um único inteiro M, o poder máximo do feitiço que pode ser utilizado pelo personagem.

Restrições

- $3 \leq N \leq 10^6$.
- $-10^5 \leq X_i \leq 10^5$.

Exemplos

Entrada	**Saída**
4 10 -2 20 7	30

Entrada	**Saída**
5 1 -8 1 10 -1	11

Problema J. Um dia no museu

Nome do arquivo fonte: `museu.c`, `museu.cpp`, *ou* `museu.java`

Foi construído um museu de arte moderna na cidade de Timóteo. Com o intuito de garantir a segurança das obras de arte ali presentes, câmeras com visão 360° foram instaladas pelo museu. Devido ao corte de custos, apenas uma câmera foi instalada por sala.

A empresa que realizou a instalação não certificou se todas as paredes eram completamente visíveis pela câmera e agora ela precisa de sua ajuda.

Dado a planta baixa de uma sala do museu e a posição da câmera, você deve indicar se todas as paredes são visíveis por ela ou não.

Uma parede é considerada vigiada pela câmera se todos os pontos sobre ela podem ser vistos pela câmera, ou seja, não são bloqueados por outra parede. Uma parede colinear com a câmera também não é considerada como vigiada.

Entrada

A entrada é composta pela descrição de uma sala do museu e da posição da câmera, sendo que a sala é representada por um polígono simples. A primeira linha contêm um inteiro N, que representa a quantidade de paredes da sala. Em seguida, temos N linhas. A i-ésima linha contêm 2 inteiros X_i, Y_i que são as coordenadas do i-ésimo ponto. Uma parede é formada pela ligação do ponto i com o ponto $i + 1$. O N-ésimo ponto da entrada com o primeiro ponto formam a última parede. Por fim, temos mais uma linha com dois inteiros C_x, C_y, que representam as coordenadas da câmera.

Saída

A saída deve conter uma linha com o caractere `S` caso todas as paredes sejam completamente vigiadas pela câmera e `N` caso contrário.

Restrições

- $3 \leq N \leq 10^5$.
- $-10^5 \leq X_i, Y_i, C_x, C_y \leq 10^5$.

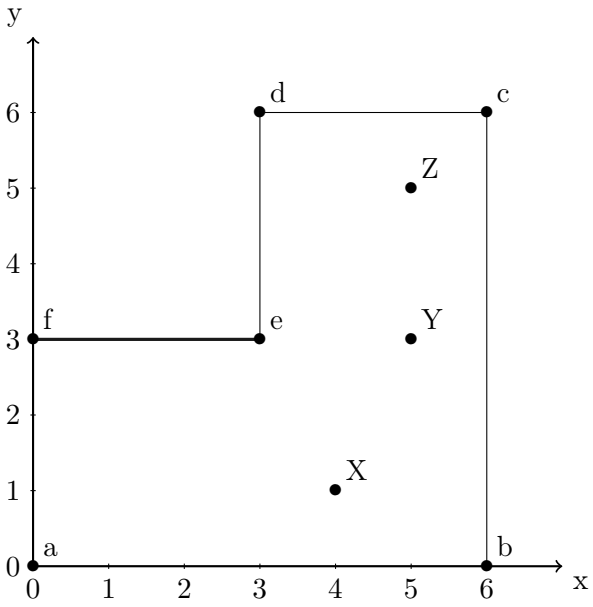
É garantido que as coordenadas geram um polígono simples e que as coordenadas da câmera se encontram dentro desse polígono.

Exemplos

Entrada	**Saída**
6 0 0 6 0 6 6 3 6 3 3 0 3 4 1	S

Entrada	**Saída**
6 0 0 6 0 6 6 3 6 3 3 0 3 5 3	N

Entrada	**Saída**
6 0 0 6 0 6 6 3 6 3 3 0 3 5 5	N



Os pontos X, Y e Z representam a posição da câmera no primeiro, segundo e terceiro exemplo, respectivamente. No segundo exemplo, a câmera Y não consegue monitorar completamente a parede (e, f) e no terceiro exemplo a câmera Z não monitora (e, f) e (f, a)

Problema K. Bacon e o tesouro

Nome do arquivo fonte: `capivara.c`, `capivara.cpp`, *ou* `capivara.java`

Sim, isso está acontecendo de novo.

Pela terceira maratona seguida teremos um problema envolvendo a dinastia fundada por Bacon – o Grande. Desta vez, o protagonista de nossa história é o neto mais novo de Bacon – o Bravo – filho de Bacon – a Ambiciosa – e o rei mais improvável já registrado como monarca de todas as capivaras.

Esta passagem das *Crônicas das Capivaras* narra os primeiros momentos do reinado de Bacon – o Grafo – que, ao invés de se tornar um monarca absolutista como seus antecessores, repartiu o poder entre vários ministros para poder se dedicar a sua verdadeira paixão: a combinatória. Ele voltou suas atenções para a abertura de um antigo baú adquirido por sua mãe após uma dura campanha de inverno contra insurgentes e que tem frustrado os mais renomados pensadores do reino.

A fechadura do baú é composta por um tabuleiro quadriculado de dimensões N por M em que cada umas das $N \times M$ posições tem uma joia encrustada que pode mudar para uma de 3 possíveis cores. Uma joia na posição (a, b) do tabuleiro é adjacente a todas as joias (x, y) que satisfazem $|a - x| + |b - y| = 1$. As joias tem propriedades mágicas, de tal forma que duas joias em posições adjacentes no tabuleiro entram em ressonância se e somente se elas tem a mesma cor. Esse efeito é transitivo, ou seja, se (a, b) está em ressonância com (c, d) e (c, d) está em ressonância com (x, y), então (a, b) também está em ressonância com (x, y). As joias, porém, não são muito resistentes e se uma joia está em ressonância com pelo menos outras K joias, ela se desintegra instantaneamente. O tabuleiro do baú também é especial, e permite que seu usuário troque a cor de uma única joia por vez.

O mecanismo de segurança é simples: o tesouro é destrancado se e somente se todas as joias no tabuleiro se desintegram no mesmo instante de tempo.

Exausto após tantas tentativas mal sucedidas, nosso herói caminhou para uma necessária noite de sono; ele não sabia nem se era possível abrir o tesouro e as dúvidas o deixaram cada vez mais desesperançoso. Mas por quê necessária? Bom, está escrito nas *Crônicas* que Bacon – o Grafo – recebeu, durante o sonho, a resposta de três entidades místicas que se auto intitulavam "maratonistas".

Claramente, vocês são essas entidades. Salvem nosso monarca favorito lhe dizendo se o tesouro pode ou não ser aberto e, caso possa, qual o menor número de joias que devem ter a cor trocadas para tal.

Entrada

A primeira linha da entrada contém os inteiros N, M e K, separados por espaço, que representam, respectivamente, o número de linhas e colunas do tabuleiro e o limite de resistência das joias. Em seguida, existem N linhas, cada uma com M inteiros, separados por espaço, representando as cores das joias na respectiva linha. É garantido que nenhuma joia está em ressonância com K outras joias.

Saída

A saída contem um único inteiro M, o menor número de recolorações que devem ser feitas para abrir o baú. Caso isso seja impossível, imprima -1.

Restrições

- $1 \leq N, M \leq 5 \times 10^6$.
- $1 \leq N \times M \leq 5 \times 10^6$.
- $1 \leq K \leq 5 \times 10^6$.

Exemplos

Entrada	**Saída**

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

Entrada	**Saída**
`1 6 3` `1 1 2 2 3 3`	`4`

Problema L. Se liga na cifra

Nome do arquivo fonte: `cipher.c`, `cipher.cpp`, *ou* `cipher.java`

A cifra de transposição Rail Fence é um clássico algoritmo de criptografia. A ideia é cifrar uma mensagem escrevendo cada caractere diagonalmente para baixo e para cima em sucessivos trilhos de uma cerca imaginária. Um verdadeiro zig-zag! Depois, basta obter a sequência horizontal de caracteres, da esquerda para a direita, de cima para baixo, para formar a mensagem cifrada, sempre ignorando espaços e pontuação. Por exemplo, para criptografar a mensagem: ‘‘`QUEM TE CONHECE NAO ESQUECE JAMAIS`’’ em 3 trilhos, escreva o texto como:

```
Q . . . T . . . N . . . E . . . E . . . E . . . A . . . S
. U . M . E . O . H . C . N . O . S . U . C . J . M . I .
. . E . . . C . . . E . . . A . . . Q . . . E . . . A . .
```

Então, leia o texto horizontalmente, da esquerda para direita, de cima para baixo, ignorando espaços e pontuação. O resultado é o seguinte texto cifrado:

```
QTNEEEASUMEOHCNOSUCJMIECEAQEA
```

Sua missão é escrever uma solução para obter a mensagem original, ignorando espaços e pontuação, a partir da mensagem cifrada C e do número de trilhos N utilizados para codificá-la.

Entrada

A primeira linha da entrada contém uma única string, C, que representa a mensagem que deve ser decodificada. A segunda linha da entrada contém um único inteiro, N, que é o número de trilhos que devem ser usados para decodificar a mensagem.

Saída

A saída deve conter uma única string M, representando a mensagem decodificada.

Restrições

- $1 \leq |C| \leq 2000$.
- $1 \leq N \leq |C|$.

É garantido que C contém apenas letras maiúsculas.

Exemplos

Entrada	**Saída**
`QTNEEEASUMEOHCNOSUCJMIECEAQEA` `3`	`QUEMTECONHECENAOESQUECEJAMAIS`

Entrada	**Saída**
`PIAEJOUODQE` `5`	`PAODEQUEIJO`

Problema M. Jogo do Gato

Nome do arquivo fonte: `gato.c`, `gato.cpp`, *ou* `gato.java`

Um dos principais passatempos nas escolas da Bacônia moderna é o estudo da teoria dos números. O que poucos sabem é que toda essa popularidade se deve aos esforços de Bacon – o Grafo – de promover o conhecimento e o avanço científico durante seu reinado, principalmente na matemática e sua paixão pessoal, a combinatória.

A mais nova febre entre as jovens capivaras é o Jogo do Gato, que rapidamente tem ocupado espaço na Bacônia devido a sua natureza competitiva. Seu crescimento tem sido tão rápido que já há uma maratona planejada para o próximo ano, que promete desafiar até as capivaras mais bem preparadas do reino. O jogo é disputado entre duas capivaras, e funciona em três fases. Na primeira fase, uma capivara é designada como a numeróloga e a outra como o gato. Na segunda fase, a numeróloga escolhe um número N enquanto, ao mesmo tempo, o gato constrói uma lista P de números primos. No início da terceira e última fase, as capivaras revelam suas escolhas e começa a disputa. Mas como ela funciona? O objetivo de cada capivara é encontrar uma forma de dividir N em uma sequência de partes de tal forma que cada parte é divisível por pelo menos um elemento de P. Uma parte é uma sequência contígua de dígitos de N. Alguns minutos após o início da terceira fase, o juiz apita e as competidoras devem mostrar suas sequências, ou declarar que é impossível formar uma sequência. É declarada a vencedora do desafio a capivara que tiver a menor sequência em ordem lexicográfica; em caso de empate, inicia-se uma nova rodada a partir da primeira fase.

Para tudo ficar mais claro, vamos a um exemplo com $N = 227$ e $P = \{2, 3, 7\}$. As três formas de partir N, ordenadas lexicograficamente, são $\{\{2, 2, 7\}, \{2, 27\}, \{22, 7\}\}$. Dessa forma, se uma capivara escolheu $\{2, 2, 7\}$ e a outra escolheu $\{2, 27\}$, a primeira é declarada a vencedora.

Os juízes da competição estão cansados de ter de resolver os casos esdrúxulos dos competidores manualmente a cada rodada, e pediram para você escrever um programa que, dados N e P, retorne a primeira sequência da ordem lexicográfica. No nosso exemplo, o seu programa deve retornar $\{2, 2, 7\}$.

Entrada

A linha da entrada contem dois inteiros, N e K, onde K é o comprimento do conjunto de primos P. A segunda linha da entrada contém K inteiros $p_1, \ldots, p_K$, separados por espaços, que correspondem aos elementos de P.

Saída

Caso seja impossível construir uma sequência, a única linha da saída deve conter o inteiro -1. Caso contrário, a primeira linha da saída deve conter um único inteiro M: o tamanho da primeira sequência da ordem lexicográfica. A segunda linha deve conter M inteiros, separados por espaços, que correspondem às partes da sequência.

Restrições

- $1 \leq N \leq 10^{1000} - 1$.
- $1 \leq K \leq 50$.
- $2 \leq p_i \leq 10^6$.

Exemplos

Entrada	**Saída**
227 3 2 3 7	3 2 2 7

Entrada	**Saída**
222 2 5 7	-1

Problema N. Maratonando

Nome do arquivo fonte: `maratonando.c`, `maratonando.cpp`, *ou* `maratonando.java`

Pedro é professor de uma instituição que nunca participou de competições de algoritmos e programação. Ao tomar conhecimento deste tipo de competição, ele percebeu uma oportunidade incrível para incentivar o aprendizado de seus estudantes. Assim, ele começou as atividades de divulgação e recrutamento para atrair estudantes para a próxima Maratona Mineira.

Ele notou que em várias destas competições as equipes são formadas por equipes de três competidores. Como sua instituição ainda não tem uma tradição na competição, o número de alunos interessados em participar dos treinamentos foi pequeno. Por conta disso, ele resolveu criar equipes de dois competidores, para que o número de equipes fosse maior e, com isso, pudesse desenvolver uma competitividade maior entre seus estudantes. Cada equipe deve ser formada por um aluno que já cursou todas as disciplinas de algoritmos e por um aluno novato, que ainda as está cursando.

Pedro notou que alguns de seus estudantes não se sentiam confortáveis em formar duplas com quaisquer outros estudantes. Por conta disso, ele pediu para cada estudante novato elaborar uma lista de potenciais parceiros, isto é, uma lista de estudantes experientes com os quais eles gostariam de formar duplas. Naturalmente, nem sempre é possível formar duplas com todos os alunos satisfazendo tais listas, então Pedro decidiu analisá-las de modo a determinar qual seria o maior número de duplas que poderia ser formado. Infelizmente, ele sabia que os alunos que estivessem em duplas que não gostassem ou que não pertencessem a alguma dupla acabariam desmotivados e desistiriam dos treinamentos, assim, ele formou apenas duplas entre potenciais parceiros.

Uma outra observação de Pedro é que alunos que possuem potenciais parceiros em outras duplas acabam interagindo com eles após as competições. Isto é bem interessante, porque, além de trocar conhecimento com seu colega de equipe, estudantes de diferentes duplas trocam experiências entre si. Assim, o conhecimento acaba se disseminando entre as diversas duplas e todos evoluem juntos. Mas, por conta da natureza das relações de potenciais parceiros, Pedro notou que nem sempre o conhecimento alcançava todos os estudantes.

Pedro deseja selecionar um conjunto de duplas que ele entenda como sendo o mais promissor. Para isso, ele deseja selecionar o maior número possível de duplas de forma que (a) se dois estudantes formam uma dupla, eles se consideram potenciais parceiros e que (b) seja possível que o conhecimento possa ser passado entre quaisquer duas duplas, direta ou indiretamente. Como Pedro ainda não está familiarizado com todos os algoritmos de programação competitiva, ele pediu sua ajuda para determinar quais duplas formar.

Entrada

A primeira linha da entrada dois inteiros N e E, correspondente ao número de estudantes novatos e estudantes experientes, respectivamente. Em seguida, serão fornecidas N linhas, onde a i-ésima destas linhas conterá um inteiro N_i, o número de potenciais parceiros do i-ésimo estudante novato, seguido de N_i inteiros, $p_{i1}, \ldots, p_{iN_i}$, separados por espaço, contendo os potenciais parceiros do i-ésimo novato. Os identificadores dos estudantes são organizados de tal forma que alunos novatos são numerados de 1 a N e os experientes de $N+1$ a $N+E$.

Saída

A primeira linha da saída deverá conter um inteiro M, correspondente ao número máximo de duplas que podem ser formadas satisfazendo as restrições impostas por Pedro. Cada uma das M linhas seguintes conterá um par de inteiros A e B, indicando que os alunos A e B formarão uma dupla.

Caso haja mais de uma solução, qualquer uma delas pode ser impressa. As duplas podem ser impressas em qualquer ordem.

Restrições

- $1 \leq N, E \leq 10^3$.
- $0 \leq N_i \leq E$.
- $N + 1 \leq p_{ij} \leq N + E$.

Exemplos

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

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