

Competidor(a): _____

Número de inscrição: _____-_____ (opcional)

Este Caderno de Tarefas não pode ser levado para casa após a prova. Após a prova entregue este Caderno de Tarefas para seu professor guardar. Os professores poderão devolver os Cadernos de Tarefas aos competidores após o término do período de aplicação das provas (3 de outubro de 2026).



Olimpíada Brasileira de Informática

OBI2026

Caderno de Tarefas

Modalidade Programação • Nível 2 • Fase 3

3 de outubro de 2026

A PROVA TEM DURAÇÃO DE 5 HORAS

Promoção:



Sociedade Brasileira de Computação

Apoio:



Coordenação:



Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 19 páginas (não contando a folha de rosto), numeradas de 1 a 19. Verifique se o caderno está completo.
- A prova deve ser feita individualmente.
- É proibido consultar a Internet, livros, anotações ou qualquer outro material durante a prova. É permitida a consulta ao *help* do ambiente de programação se este estiver disponível.
- As tarefas têm o mesmo valor na correção.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa; em particular, seu programa não deve escrever frases como “Digite o dado de entrada.” ou similares.
- Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
- As tarefas **não** estão necessariamente ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
- Preste muita atenção no nome dos arquivos fonte indicados nas tarefas. 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 Pascal devem ser arquivos com sufixo *.pas*; soluções na linguagem Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; soluções na linguagem Python 3 devem ser arquivos com sufixo *.py*; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*.
- Na linguagem Java, **não** use o comando *package*, e note que o nome de sua classe principal deve usar somente letras minúsculas (o mesmo nome do arquivo indicado nas tarefas).
- Para tarefas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada tarefa.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou pen-drive, conforme especificado pelo seu professor.
- Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (normalmente é o teclado) e escritos na saída padrão (normalmente é a tela). Utilize as funções padrão para entrada e saída de dados:
 - em Pascal: *readln*, *read*, *writeln*, *write*;
 - em C: *scanf*, *getchar*, *printf*, *putchar*;
 - em C++: as mesmas de C ou os objetos *cout* e *cin*.
 - em Java: qualquer classe ou função padrão, como por exemplo *Scanner*, *BufferedReader*, *BufferedWriter* e *System.out.println*
 - em Python: *read*, *readline*, *readlines*, *input*, *print*, *write*
 - em Javascript: *scanf*, *printf*
- Procure resolver a tarefa de maneira eficiente. Na correção, eficiência também será levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo nas tarefas.

Six Seven

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

Augusto adora ver *rows* (vídeos curtos em forma horizontal) no *TocToc*. Um dos seus *rows* favoritos é um vídeo onde um personagem comicamente diz “*Six Seven*” (ou seja, os números 6 e 7 em inglês).

Ao mesmo tempo, Augusto precisava comprar dois tipos de peças para um projeto da faculdade: engrenagens e botões, que por coincidência, custam 6 e 7 reais, respectivamente. Ele tem um orçamento de N reais, e quer gastar o orçamento inteiramente, ou seja, a soma de todos os gastos deve ser N . No entanto, engrenagens são muito mais importantes que botões, e por isso, Augusto quer maximizar o número de engrenagens compradas, independente do número de botões comprados.

Infelizmente, Augusto passou o dia todo vendo *rows*, e agora está atrasado para comprar as peças do seu projeto. Por isso, pediu sua ajuda. Sua missão é escrever um programa que determina o número X de engrenagens e Y de botões que Augusto deve comprar de forma que no total, sejam gastos exatamente N reais, e que o número de engrenagens compradas seja o máximo possível. Caso seja impossível gastar todo o orçamento, reporte isso.

Entrada

A entrada contém um único número inteiro, o orçamento N , em reais.

Saída

A saída deve conter, caso possível, dois números inteiros, X e Y , o número de engrenagens e de botões respectivamente. Caso seja impossível, deve conter um único número, -1 .

Restrições

- $1 \leq N \leq 10^{18}$.

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 (16 pontos):** $N \leq 20$.
- **Subtarefa 3 (19 pontos):** $N \leq 10^3$.
- **Subtarefa 4 (28 pontos):** $N \leq 10^7$.
- **Subtarefa 5 (37 pontos):** Sem restrições adicionais.

Para competidores que utilizam C++ ou Java: Observe que o valor de N na entrada pode ser muito grande para caber em um inteiro de 32 bits. É recomendado o uso de inteiros de 64 bits (`long long` em C++; `long` em Java). (*Competidores usando Python podem ignorar este aviso.*)

Dica de pontuação: se o seu programa funcionar corretamente apenas para as restrições de uma subtarefa, você receberá os pontos daquela subtarefa. Pode valer a pena entregar uma solução parcial!

Exemplos

Exemplo de entrada 1	Exemplo de saída 1
14	0 2

Explicação do exemplo 1: É possível gastar todos os 14 reais de uma única forma, comprando 0 engrenagens e 2 botões.

Exemplo de entrada 2	Exemplo de saída 2
15	-1

Explicação do exemplo 2: É impossível gastar todos os 15 reais.

Exemplo de entrada 3	Exemplo de saída 3
67	10 1

Explicação do exemplo 3: Note que há mais de um modo de gastar todo o orçamento: comprando 3 engrenagens e 7 botões (totalizando $3 \times 6 + 7 \times 7 = 67$) ou comprando 10 engrenagens e 1 botão (totalizando $6 \times 10 + 7 \times 1 = 67$). Como queremos maximizar o número de engrenagens compradas, a solução é o segundo modo.

Picos

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

Você trabalha em uma empresa de desenvolvimento de jogos mobile e está ajudando a criar um jogo de trilhas. Cada fase do jogo é representada por uma sequência de N marcos numerados de 1 a N , onde cada marco possui uma altitude única — ou seja, a sequência de altitudes forma uma **permutação** de 1 a N .

Vale lembrar que uma **permutação** de 1 a N é uma sequência de N inteiros distintos de 1 a N , arranjados em alguma ordem. Por exemplo, as sequências 1, 2, 3 e 2, 1, 4, 3 são **permutações**, porém, as sequências 1, 2, 4 (falta o elemento 3) e 1, 1, 2, 3 (o elemento 1 está repetido) não são **permutações**.

Durante os testes internos, a equipe de design percebeu que fases com **picos** fazem os jogadores desistirem. Um pico ocorre quando existe um marco j com altitude A_j tal que:

- existe algum marco anterior a j com altitude menor, ou seja, existe $i < j$ com $A_i < A_j$.
- e também existe algum marco posterior a j com altitude menor, ou seja, existe $k > j$ com $A_k < A_j$.

Por exemplo, a seguinte permutação 1, 4, 3, 5, 2, possui diversos picos:

- $i = 1, j = 2, k = 3$ é um pico, pois $A_1 = 1 < A_2 = 4$ e $A_3 = 3 < A_2 = 4$.
- $i = 1, j = 3, k = 5$ é outro pico, pois $A_1 = 1 < A_3 = 3$ e $A_5 = 2 < A_3 = 3$. Perceba que i, j, k não precisam ser consecutivos.

Já a permutação 5, 3, 2, 1, 4, não possui nenhum pico.

Seu chefe percebeu que as fases sem picos engajam muito mais os jogadores, e lhe passou a tarefa de determinar a K -ésima delas, em ordem lexicográfica.

Vale lembrar que, sejam P e Q duas permutações de 1 a N sem picos, dizemos que P vem antes de Q lexicograficamente se existe um índice i ($1 \leq i \leq N$) tal que:

- $P_j = Q_j$ para todo índice $j < i$.
- $P_i < Q_i$.

Por exemplo suponha que P seja a permutação 5, 3, 1, 2, 4 (perceba que não possui nenhum pico), e Q seja a permutação 5, 3, 2, 1, 4. Nesse caso P vem antes que Q lexicograficamente, pois para $i = 3$ temos:

- $P_1 = Q_1$.
- $P_2 = Q_2$.
- $P_3 < Q_3$.

Portanto, dados N e K , determine a K -ésima (em ordem lexicográfica) permutação de 1 a N que não possui nenhum pico.

Entrada

A entrada contém uma única linha com dois inteiros N e K , representando o número de marcos de uma fase e o índice da permutação desejada.

Saída

A saída deve conter uma única linha com N inteiros separados por espaços, representando a K -ésima permutação de 1 a N sem picos em ordem lexicográfica.

Restrições

- $3 \leq N \leq 10^5$.
- $1 \leq K \leq 10^{18}$.
- É garantido que existam pelo menos K permutações de N elementos sem picos.

Para competidores que utilizam C++ ou Java: Observe que o valor de K na entrada pode ser muito grande para caber em um inteiro de 32 bits. É recomendado o uso de inteiros de 64 bits (`long long` em C++; `long` em Java). (*Competidores usando Python podem ignorar este aviso.*)

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 (13 pontos):** $N \leq 10$.
- **Subtarefa 3 (18 pontos):** $N \leq 16$.
- **Subtarefa 4 (21 pontos):** $K = 2^{N-2} + 1$.
- **Subtarefa 5 (30 pontos):** $N \leq 30$.
- **Subtarefa 6 (18 pontos):** Sem restrições adicionais.

Exemplos

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

Explicação do exemplo 1: Todas as permutações de 3 elementos, em ordem lexicográfica, são listadas abaixo:

Permutação	Tem pico?	Justificativa
[1, 2, 3]	Não	
[1, 3, 2]	Sim	$i = 1, j = 2, k = 3$ pois $A_1 = 1 < A_2 = 3$ e $A_3 = 2 < A_2 = 3$
[2, 1, 3]	Não	
[2, 3, 1]	Sim	$i = 1, j = 2, k = 3$ pois $A_1 = 2 < A_2 = 3$ e $A_3 = 1 < A_2 = 3$
[3, 1, 2]	Não	
[3, 2, 1]	Não	

As permutações sem pico, em ordem lexicográfica, são: [1, 2, 3], [2, 1, 3], [3, 1, 2], [3, 2, 1]. A quarta delas é [3, 2, 1].

Exemplo de entrada 2 3 3	Exemplo de saída 2 3 1 2
------------------------------------	------------------------------------

Explicação do exemplo 2: Este exemplo satisfaz as restrições da subtarefa 4.

Exemplo de entrada 3 5 12	Exemplo de saída 3 5 3 2 1 4
-------------------------------------	--

Exemplo de entrada 4 10 500	Exemplo de saída 4 10 9 8 7 6 3 2 1 4 5
---------------------------------------	---

O Analista

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

Beto é um analista júnior em uma renomada casa de análises financeiras. Ele foi encarregado de preparar o slide principal da apresentação de fim de ano para os investidores, que deve exibir o histórico do valor da Carteira Recomendada da empresa ao longo de N dias consecutivos.

Para convencer os investidores de que a casa entrega resultados sólidos, Beto teve uma ideia: ele vai agrupar os N dias do histórico em vários períodos contínuos e não vazios (os períodos não precisam ter o mesmo tamanho, pois a casa libera relatórios em diversos momentos). Para que a narrativa de crescimento seja impecável, o valor máximo alcançado pela carteira em cada período deve ser estritamente crescente.

No entanto, o mercado financeiro é volátil. Um único dia com um pico fora de hora ou uma queda abrupta pode arruinar completamente a separação dos períodos. Portanto, Beto decidiu que pode limpar os dados apagando alguns dias do histórico antes de formar os grupos. Ele justificará essas exclusões como falhas no sistema de cotação, e sabe que se apagar no máximo K dias, seu chefe não desconfiará.

Dada a sequência diária do valor da carteira e o limite K de exclusões permitidas, sua tarefa é descobrir qual é o número máximo de períodos que Beto consegue formar seguindo todas as regras.

Entrada

A primeira linha contém dois inteiros N e K , representando, respectivamente, o número de dias anotados por Beto e a quantidade máxima de dias que ele pode deletar.

A segunda linha contém N inteiros $V_1, V_2, \dots, V_N$, onde V_i representa o valor da carteira no i -ésimo dia.

Saída

Seu programa deve imprimir um único inteiro, o número máximo de períodos contínuos e não vazios que Beto pode obter após deletar no máximo K dias, garantindo que o valor máximo de cada período seja estritamente crescente.

Restrições

- $1 \leq N \leq 5000$
- $0 \leq K \leq N$
- $1 \leq V_i \leq 10^5$, 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 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 (9 pontos): $K = 0$.
- Subtarefa 3 (8 pontos): $N \leq 20$.
- Subtarefa 4 (8 pontos): $K = N$.
- Subtarefa 5 (22 pontos): $N \leq 100$.
- Subtarefa 6 (19 pontos): $N \leq 500$.
- Subtarefa 7 (34 pontos): Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	Exemplo de saída 1
6 0 3 2 5 3 4 6	3

Explicação do exemplo 1: Uma possível forma de separar os períodos é a seguinte: [3] [2 5 3] [4 6]. Note que os valores máximos de cada período são estritamente crescentes: 3, 5 e 6. É possível provar que, neste caso, Beto não consegue separar a lista em 4 períodos ou mais. Esse exemplo satisfaz a restrição da subtarefa 2.

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

Explicação do exemplo 2: Beto pode apagar os dias 1 e 3 (cujos valores são 3 e 5), assim a lista restante fica: 2, 3, 4, 6. Uma forma de separar os períodos é: [2] [3] [4] [6]. Note que os máximos são estritamente crescentes. É possível provar que, neste caso, Beto não consegue separar em 5 períodos ou mais.

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

Explicação do exemplo 3: Perceba que, apesar de poder deletar até 3 dias, nesse caso o ótimo é Beto não deletar nenhum dia.

Exemplo de entrada 4	Exemplo de saída 4
12 3 4 2 8 3 10 1 5 6 9 7 11 12	7

Selva

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

Lucy é a líder de uma expedição científica explorando uma vasta selva, mas infelizmente sua equipe perdeu a rota e acabou se desorientando. Para piorar a situação, a região abriga diversas espécies de animais perigosos e já está anoitecendo.

Felizmente, ela possui um radar portátil de última geração. O radar estabeleceu um sistema de coordenadas cartesiano para uma grande área, representando-a em um plano de tamanho $N \times N$. Lucy e sua equipe se encontram atualmente na origem, o ponto $(0, 0)$, e o radar indica que um acampamento seguro está no ponto $(N - 1, N - 1)$. Além disso, o aparelho também detectou a localização exata de M animais perigosos adormecidos pela floresta. A posição de cada animal é dada pelas coordenadas (x_i, y_i) .

Atualmente, o objetivo da equipe é sair do ponto $(0, 0)$ e chegar no acampamento no ponto $(N - 1, N - 1)$. Para não andar sobre territórios desconhecidos, Lucy decidiu que eles deveriam caminhar somente por pontos cujas coordenadas satisfazem $0 \leq x, y < N$. A cada passo, eles podem se deslocar do ponto atual para qualquer um dos até 8 pontos adjacentes (ou seja, movendo-se paralelamente aos eixos ou nas diagonais).

Como a segurança da sua equipe é a prioridade, Lucy deseja manter a maior distância possível de qualquer animal durante o trajeto. O nível de segurança de um caminho é determinado pela menor distância entre um ponto do caminho e um dos animais. A distância medida pelo radar é a distância de Manhattan, ou seja, a distância entre dois pontos (x_a, y_a) e (x_b, y_b) é calculada por:

$$\text{dist}(a, b) = |x_a - x_b| + |y_a - y_b|$$

Ou seja, é a soma da diferença entre as coordenadas x com a diferença entre as coordenadas y .

Assim, dadas as dimensões da área rastreada e as coordenadas de todos os animais, encontre um caminho do início até o acampamento que maximize a menor distância a algum animal e imprima essa distância.

Entrada

A primeira linha da entrada contém dois inteiros N e M , representando o tamanho da selva e a quantidade de animais, respectivamente.

As próximas M linhas contêm dois inteiros x_i e y_i cada, representando as coordenadas do i -ésimo animal.

Saída

Seu programa deve imprimir uma única linha contendo um inteiro: a maior menor distância de algum animal em um caminho válido de $(0, 0)$ a $(N - 1, N - 1)$.

Restrições

- $3 \leq N \leq 10^9$.
- $1 \leq M \leq 10^5$.
- $0 \leq x_i, y_i < N$, para todo $1 \leq i \leq M$.
- A origem $(0, 0)$ e o acampamento $(N - 1, N - 1)$ nunca contêm animais.
- Pode haver mais de um animal na mesma posição.

Informações sobre a pontuação

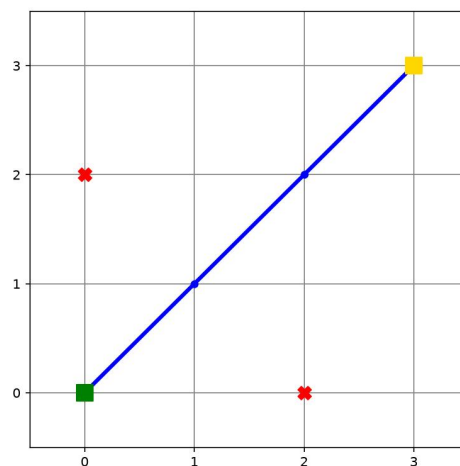
A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima.

- **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 (7 pontos):** $M = 1$.
- **Subtarefa 3 (15 pontos):** $N \leq 500$ e $M \leq 500$.
- **Subtarefa 4 (23 pontos):** $N \leq 2000$ e $M \leq 1000$.
- **Subtarefa 5 (24 pontos):** $M \leq 1000$.
- **Subtarefa 6 (31 pontos):** Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	Exemplo de saída 1
4 2 0 2 2 0	2

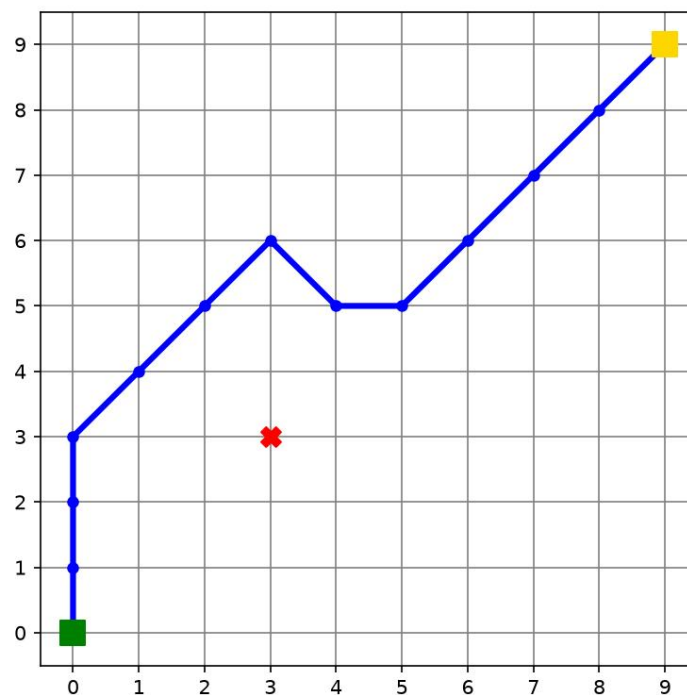
Explicação do exemplo 1: Um exemplo de caminho cuja maior menor distância a algum animal seja 2 é mostrado a seguir:



Perceba que a distância do ponto (1,1) para ambos os animais é exatamente 2 (por exemplo: $|1 - 0| + |1 - 2| = 2$). Assim, é possível "espremer-se" pela diagonal, garantindo uma distância de segurança mínima igual a 2.

Exemplo de entrada 2	Exemplo de saída 2
10 1 3 3	3

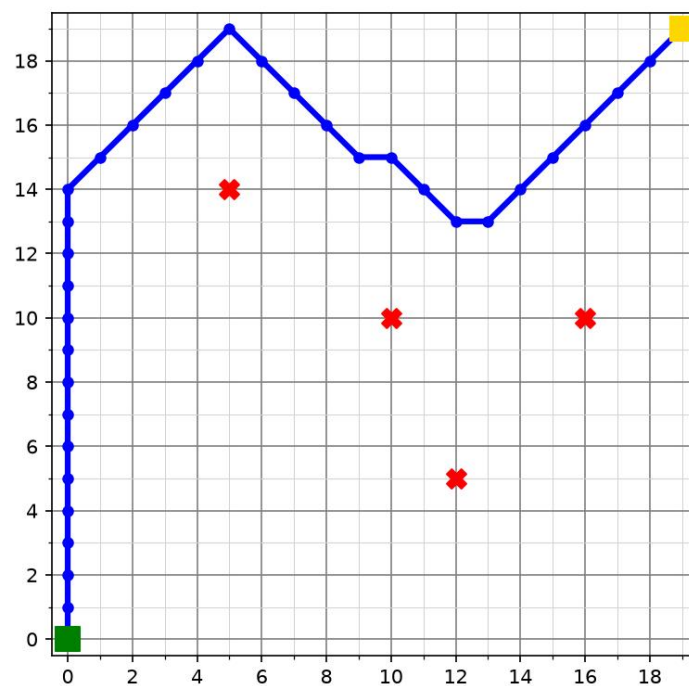
Explicação do exemplo 2: Um exemplo de caminho ótimo é mostrado a seguir:



Vale ressaltar que este exemplo satisfaz as restrições da subtarefa 2.

Exemplo de entrada 3	Exemplo de saída 3
20 4 5 14 10 10 12 5 16 10	5

Explicação do exemplo 3:



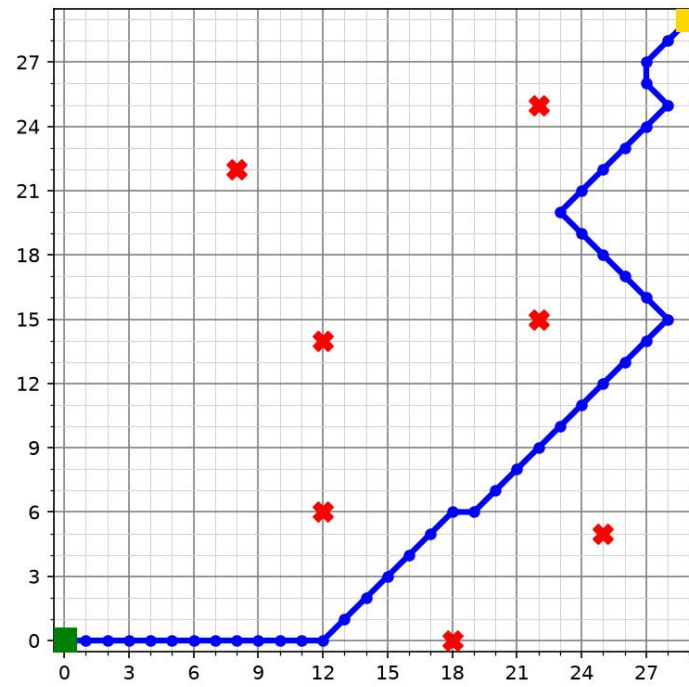
Exemplo de entrada 4

```
30 7
8 22
12 6
12 14
22 15
22 25
25 5
18 0
```

Exemplo de saída 4

```
6
```

Explicação do exemplo 4:



Divisão do Reino

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

O reino da OBI é governado pelo sábio e benevolente rei Rodrigo. Seu reino consiste em N cidades, numeradas de 1 a N . Essas estão interconectadas por $N - 1$ estradas, através das quais é possível viajar entre quaisquer duas cidades do reino.

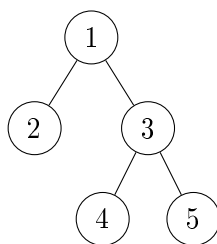
O rei Rodrigo deseja repartir o território entre a nobreza do reino e, para isso, pediu que cada um dos Q nobres enviasse uma proposta de quais territórios ele gostaria de governar. No entanto, um nobre não pode escolher cidades de uma forma qualquer. O nobre i deve escolher duas cidades distintas a_i e b_i , as quais significam que ele deseja governar todas as cidades no caminho de a_i até b_i (inclusive estas duas).

Rodrigo é bondoso e permitiu que cada um escolhesse as cidades que quisesse, mas também tem muita experiência com administração de territórios. Ele sabe que, para manter o equilíbrio de poder no reino, deve haver uma hierarquia clara entre todos os pares de nobres. Assim, para dois nobres x e y , suas propostas de territórios são **coerentes** se, caso aceitas, uma das seguintes situações seria verdadeira:

- x seria soberano de y , portanto, **todos** os territórios de y também seriam controlados por x .
- y seria soberano de x , portanto, **todos** os territórios de x também seriam controlados por y .
- x e y não seriam soberanos um do outro, portanto, eles não compartilhariam controle sobre **nenhum** território.

Em especial, vale notar que, mesmo que alguma cidade seja controlada por vários nobres, suas propostas ainda podem ser coerentes entre si, desde que eles satisfaçam as regras de hierarquia de Rodrigo.

Por exemplo, considere o seguinte mapa do reino:



Considere as seguintes propostas de quatro nobres:

- **Nobre 1:** $a_1 = 1$ e $b_1 = 2$. Ou seja, o nobre 1 deseja governar as cidades 1 e 2.
- **Nobre 2:** $a_2 = 2$ e $b_2 = 5$. Ou seja, o nobre 2 deseja governar as cidades 1, 2, 3 e 5.
- **Nobre 3:** $a_3 = 1$ e $b_3 = 5$. Ou seja, o nobre 3 deseja governar as cidades 1, 3 e 5.
- **Nobre 4:** $a_4 = 3$ e $b_4 = 4$. Ou seja, o nobre 4 deseja governar as cidades 3 e 4.

Nesse caso, há três pares de propostas coerentes:

- As propostas dos nobres 1 e 2 são coerentes, pois o nobre 2 controlaria todos os territórios do nobre 1.
- As propostas dos nobres 2 e 3 são coerentes, pois o nobre 2 controlaria todos os territórios do nobre 3.
- As propostas dos nobres 1 e 4 são coerentes, pois eles não compartilhariam controle de nenhum território.

Note que os outros pares de propostas não são coerentes.

Rodrigo deseja analisar as propostas dos nobres, mas existem muitas cidades e muitos nobres no reino. Assim, ele deseja sua ajuda em duas situações:

- **Situação 0:** Rodrigo deseja saber se **todos** os pares de propostas x, y (com $x < y$) são **coerentes**.
- **Situação 1:** Rodrigo deseja saber **quantos** pares de propostas x, y (com $x < y$) são **coerentes**.

Entrada

A primeira linha da entrada contém três inteiros N, Q e T , indicando o número de cidades, o número de nobres e a situação requisitada pelo rei, respectivamente.

As próximas $N - 1$ linhas contêm as descrições das estradas. A j -ésima dessas linhas contém dois inteiros distintos u_j e v_j , indicando que a j -ésima estrada conecta as cidades u_j e v_j .

As próximas Q linhas contêm as descrições das propostas. A i -ésima dessas linhas contém dois inteiros a_i e b_i , representando a proposta do i -ésimo nobre. É garantido que $a_i \neq b_i$ e que não há duas propostas que descrevam o mesmo caminho.

Saída

A saída deve conter uma única linha com um único inteiro.

Se $T = 0$, a saída deve ser 1 se **todos** os pares de propostas forem coerentes e 0 caso contrário.

Se $T = 1$, a saída deve conter a **quantidade** de pares de propostas que são coerentes.

Restrições

- $2 \leq N, Q \leq 10^5$.
- $0 \leq T \leq 1$
- $1 \leq u_j, v_j \leq N$ para todo $1 \leq j \leq N - 1$.
- $1 \leq a_i, b_i \leq N$ e $a_i \neq b_i$ para todo $1 \leq i \leq Q$.
- Não existem duas propostas que descrevem o mesmo caminho.

Para competidores que utilizam C++ ou Java: Observe que o valor na saída pode ser muito grande para caber em um inteiro de 32 bits. É recomendado o uso de inteiros de 64 bits (`long long` em C++; `long` em Java). *(Competidores usando Python podem ignorar este aviso.)*

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 (6 pontos):** $T = 1$ e $Q = 2$.
- **Subtarefa 3 (7 pontos):** $T = 1$ e $N, Q \leq 100$.
- **Subtarefa 4 (9 pontos):** $T = 1$ e $Q \leq 10^3$.
- **Subtarefa 5 (9 pontos):** $T = 0$ e as cidades i e $i + 1$ estão conectadas para todo $1 \leq i < N$. Ou seja, as cidades do reino formam uma linha.
- **Subtarefa 6 (16 pontos):** $T = 1$ e as cidades i e $i + 1$ estão conectadas para todo $1 \leq i < N$. Ou seja, as cidades do reino formam uma linha.
- **Subtarefa 7 (17 pontos):** $T = 0$.
- **Subtarefa 8 (11 pontos):** $T = 1$ e cada cidade está contida em no máximo 250 propostas.
- **Subtarefa 9 (25 pontos):** $T = 1$.

Exemplos

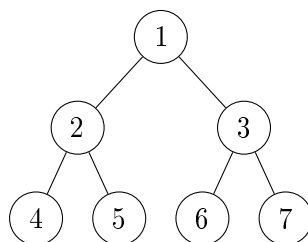
Exemplo de entrada 1	Exemplo de saída 1
5 4 1 1 2 1 3 3 4 3 5 1 2 2 5 1 5 3 4	3

Explicação do exemplo 1: Note que esse é o exemplo mostrado no enunciado.

Exemplo de entrada 2	Exemplo de saída 2
7 2 1 1 2 1 3 2 4 2 5 3 6 3 7 4 7 5 6	0

Explicação do exemplo 2: Este exemplo satisfaz as restrições da Subtarefa 2.

Este é o formato do reino:

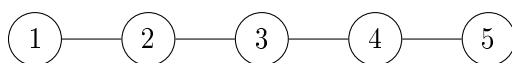


Nesse caso, a proposta 1 inclui as cidades 1, 2, 3, 4 e 7, enquanto a proposta 2 inclui as cidades 1, 2, 3, 5 e 6. Como as propostas não são coerentes, a resposta é 0.

Exemplo de entrada 3	Exemplo de saída 3
5 3 0 1 2 2 3 3 4 4 5 1 2 4 5 1 5	1

Explicação do exemplo 3: Este exemplo satisfaz as restrições da Subtarefa 5.

Este é o formato do reino:



Nesse caso, a proposta 1 inclui as cidades 1 e 2, a proposta 2 inclui as cidades 4 e 5, e a proposta 3 inclui as cidades 1, 2, 3, 4 e 5. Note que:

- As propostas dos nobres 1 e 2 são coerentes, pois eles não compartilhariam controle de nenhum território.
- As propostas dos nobres 2 e 3 são coerentes, pois o nobre 3 controlaria todos os territórios do nobre 2.
- As propostas dos nobres 1 e 3 são coerentes, pois o nobre 3 controlaria todos os territórios do nobre 1.

Como $T = 0$ e todos os pares de propostas são coerentes, a resposta é 1.

Exemplo de entrada 4	Exemplo de saída 4
5 4 0 1 2 2 3 3 4 4 5 1 2 4 5 1 5 2 4	0

Explicação do exemplo 4: Este exemplo satisfaz as restrições da Subtarefa 5.

Esse caso é igual ao anterior, porém com a inclusão da proposta 4, composta pelas cidades 2, 3 e 4.

Como $T = 0$ e existem dois pares de propostas incoerentes, dos nobres 1 e 4 e dos nobres 2 e 4, a resposta é 0.

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

Explicação do exemplo 5: Este exemplo satisfaz as restrições da Subtarefa 6.

Esse caso é igual ao anterior, porém com $T = 1$.

Exemplo de entrada 6	Exemplo de saída 6
6 4 0 1 2 1 3 3 4 3 5 1 6 2 6 1 6 3 4 4 5	1

Explicação do exemplo 6: Este exemplo satisfaz as restrições da Subtarefa 7.

Exemplo de entrada 7	Exemplo de saída 7
6 15 1 1 5 6 1 5 4 5 2 1 3 1 3 2 1 1 6 3 2 5 2 2 4 5 3 4 6 3 4 1 4 1 5 5 4 5 6 2 6 6 3	41