

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 Júnior • Fase 3

3 de outubro de 2026

A PROVA TEM DURAÇÃO DE 4 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 15 páginas (não contando a folha de rosto), numeradas de 1 a 15. 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.

Cozinheiro Real

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

Como todos sabem o casamento do Rei de Nlogônia está próximo e para agradar seu povo, além de ter muito arroz na festa (pois é símbolo de prosperidade), haverá cachorro-quente para todo mundo!

Pedro, o Rei, sabe que há uma discussão fervorosa se deve ou não colocar purê de batata no cachorro-quente, ele já ouviu falar até de um reino onde a população era dividida em dois grupos antagônicos: os Puristas, que defendem o cachorro-quente original, sem purê de batata e a dos Purê-istas, que adoram cachorro-quente com purê de batata. Para evitar brigas em seu casamento, ele solicitou a preferência de cada morador do reino.

O cozinheiro está desesperado pois ele precisa ir ao mercado comprar todos os ingredientes, e ele não sabe quanto de batata deve comprar, além disso ele tem medo de entregar um cachorro-quente errado para alguma pessoa. Dadas as preferências dos habitantes de Nlogônia, ajude o cozinheiro calculando a quantidade dos que preferem o cachorro-quente sem purê, a quantidade dos que preferem o cachorro-quente com purê, e as listas dos habitantes de cada uma dessas preferências.

Entrada

A entrada é composta por duas linhas:

- A primeira linha contém um inteiro N , representando o número de habitantes de Nlogônia. As pessoas são numeradas de 1 a N .
- A segunda linha contém N inteiros $P_1, P_2, \dots, P_N$, onde o valor P_i indica a preferência da pessoa i . Cada valor P_i pode ser 0 (prefere sem purê) ou 1 (prefere com purê).

Saída

A saída deve conter exatamente quatro linhas, no seguinte formato:

- A primeira linha deve conter a quantidade de pessoas que preferem cachorro-quente sem purê (identificadas pelo valor 0).
- A segunda linha deve conter a lista dos números das pessoas que preferem cachorro-quente sem purê.
- A terceira linha deve conter a quantidade de pessoas que preferem cachorro-quente com purê (identificadas pelo valor 1).
- A quarta linha deve conter a lista dos números das pessoas que preferem cachorro-quente com purê.

Restrições

- $2 \leq N \leq 10^5$.
- $0 \leq P_i \leq 1$ para todo $1 \leq i \leq N$.
- É garantido que haverá pelo menos uma pessoa que prefere cachorro-quente sem purê, e pelo menos uma pessoa que prefere cachorro-quente com purê.

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 (15 pontos):** $N = 2$.
- **Subtarefa 3 (29 pontos):** $N \leq 100$.
- **Subtarefa 4 (25 pontos):** Há apenas uma pessoa que prefere cachorro-quente com purê.
- **Subtarefa 5 (31 pontos):** Sem restrições adicionais.

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
7 0 1 1 0 0 1 0	4 1 4 5 7 3 2 3 6

Explicação do exemplo 1: Neste exemplo, temos $N = 7$ pessoas, com as seguintes preferências:

- A pessoa 1 prefere cachorro-quente sem purê, pois $P_1 = 0$.
- A pessoa 2 prefere cachorro-quente com purê, pois $P_2 = 1$.
- A pessoa 3 prefere cachorro-quente com purê, pois $P_3 = 1$.
- A pessoa 4 prefere cachorro-quente sem purê, pois $P_4 = 0$.
- A pessoa 5 prefere cachorro-quente sem purê, pois $P_5 = 0$.
- A pessoa 6 prefere cachorro-quente com purê, pois $P_6 = 1$.
- A pessoa 7 prefere cachorro-quente sem purê, pois $P_7 = 0$.

Portanto temos 4 pessoas que preferem sem purê: 1, 4, 5 e 7; e 3 pessoas que preferem com purê: 2, 3 e 6.

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

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

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

Explicação do exemplo 3: Este exemplo satisfaz as restrições da subtarefa 4.

Rebobinar

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

Elisa é uma influencer muito famosa que acabou de publicar o último episódio de uma série de enorme sucesso em seu canal. No total, a série publicada é composta por N pequenas cenas. No arquivo de gravação original (antes de qualquer edição), essas cenas estavam perfeitamente organizadas em ordem cronológica: a posição 1 continha a cena 1, a posição 2 continha a cena 2, e assim por diante, até a posição N com a cena N .

Durante o processo de edição da série, para criar efeitos visuais, *flashbacks* e momentos de suspense, Elisa usou a ferramenta “Rebobinar” Q vezes. A cada utilização da ferramenta, ela selecionava um trecho contínuo da série, da posição L até a posição R , e fazia o programa de edição inverter totalmente a ordem daquele bloco (a cena que estava na posição L trocava de lugar com a da posição R , a da posição $L + 1$ trocava com a de $R - 1$, e assim por diante).

Pouco tempo após o lançamento, Elisa recebeu uma notificação de direitos autorais avisando que havia uma imagem não autorizada exatamente na posição K da série completa publicada, isto é, após as edições.

Elisa possui todo o material de gravação original salvo em seu computador, mas precisa da sua ajuda para identificar a cena correta que gerou o problema. Como a ordem mudou bastante durante as edições, sua tarefa é descobrir qual era o número da cena que acabou indo parar na posição K da série publicada.

Entrada

A primeira linha da entrada contém três inteiros N , K e Q , em que N é o número de cenas, K é a posição que gerou o problema e Q é o número de vezes que a ferramenta Rebobinar foi usada.

Cada uma das Q linhas seguintes contém dois inteiros L_i e R_i , indicando que a i -ésima utilização da ferramenta inverteu o trecho da posição L_i até a posição R_i .

Saída

A saída deve conter um único inteiro: o número da cena original que está na posição K da série publicada.

Restrições

- $2 \leq N \leq 10^9$.
- $1 \leq K \leq N$.
- $1 \leq Q \leq 3 \times 10^5$.
- $1 \leq L_i < R_i \leq N$, para todo $1 \leq i \leq Q$.

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):** $N \leq 10^3$, $Q \leq 10^3$ e $R_i = L_i + 1$ para todo $1 \leq i \leq Q$ (cada operação troca apenas dois elementos adjacentes).
- **Subtarefa 3 (28 pontos):** $N \leq 10^3$ e $Q \leq 10^3$.
- **Subtarefa 4 (19 pontos):** $Q = 1$.
- **Subtarefa 5 (37 pontos):** Sem restrições adicionais.

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
10 6 3 4 8 3 6 4 9	5

Explicação do exemplo 1: Começamos com as cenas na ordem 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Após inverter o trecho [4, 8]: 1, 2, 3, **8, 7, 6, 5**, 4, 9, 10.

Após inverter o trecho [3, 6]: 1, 2, **6, 7, 8, 3**, 5, 4, 9, 10.

Após inverter o trecho [4, 9]: 1, 2, 6, **9, 4, 5, 3, 8, 7**, 10.

Como $K = 6$, a resposta é o elemento na posição 6, que é 5.

Exemplo de entrada 2	Exemplo de saída 2
8 3 4 2 3 3 4 2 3 3 4	2

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

Começamos com as cenas na ordem 1, 2, 3, 4, 5, 6, 7, 8.

Após inverter o trecho [2, 3]: 1, **3, 2**, 4, 5, 6, 7, 8.

Após inverter o trecho [3, 4]: 1, 3, **4, 2**, 5, 6, 7, 8.

Após inverter o trecho [2, 3]: 1, **4, 3**, 2, 5, 6, 7, 8.

Após inverter o trecho [3, 4]: 1, 4, **2, 3**, 5, 6, 7, 8.

Como $K = 3$, a resposta é o elemento na posição 3, que é 2.

Exemplo de entrada 3	Exemplo de saída 3
15 7 6 2 12 5 10 3 9 7 14 1 8 6 15	4

Conectar para Conquistar

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

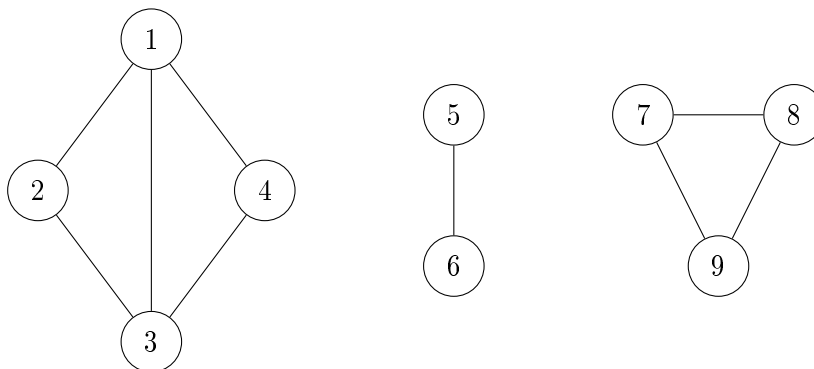
Breno é o rei de um grande território composto de N cidades, interconectadas por M estradas. Recentemente, ele adotou um lema muito famoso para desenvolver o reino: *Conectar para Conquistar*. Na cabeça dele, quanto mais estradas existirem ligando as cidades, mais rápido as pessoas podem viajar e, é claro, mais pedágios o rei pode cobrar.

No entanto, há um problema atrapalhando esses planos. Existem duas cidades com grupos antagônicos no reino que possuem ideologias extremamente incompatíveis: a cidade dos Puristas, que defendem o cachorro-quente original sem purê de batata, e a dos Purê-istas, que adoram cachorro-quente com purê de batata. A discussão sobre qual é a forma correta de comer o lanche ficou tão séria que, toda vez que pessoas de grupos diferentes se encontram, uma discussão interminável se inicia.

Por isso, para evitar confusões, Breno decretou uma nova lei. Ele ordenou que **não deve existir** um caminho entre as cidades-sede dos Puristas e dos Purê-istas. Ou seja, não deve haver uma sequência de estradas tal que seja possível viajar entre essas duas cidades. Essa lei deveria facilitar o plano de construção de novas estradas no reino, porém, coincidentemente, os dois grupos decidiram fazer uma votação para mudarem de sedes.

Breno está ansioso com o resultado da votação e se pergunta se será possível viajar entre as duas cidades e, se não for, quantas estradas novas ele conseguiria construir em diferentes situações. Mais especificamente, ele imaginou Q cenários. No i -ésimo deles, ele considerou que os Puristas elegeriam a cidade A_i para ser sua nova sede, enquanto os Purê-istas elegeriam a cidade B_i .

Por exemplo, considere o seguinte mapa do reino com $N = 9$ cidades e $M = 9$ estradas:



Suponha que o rei imaginou $Q = 2$ cenários:

- No cenário 1, $A_1 = 2$ e $B_1 = 8$. Assim, ele poderia construir no máximo 9 estradas, conectando os seguintes pares de cidades: $(2, 4)$, $(1, 5)$, $(1, 6)$, $(2, 5)$, $(2, 6)$, $(3, 5)$, $(3, 6)$, $(4, 5)$ e $(4, 6)$. Note que é impossível criar mais de 9 estradas sem que haja um caminho entre as cidades dos Puristas e dos Purê-istas.
- No cenário 2, $A_2 = 1$ e $B_2 = 4$. Nesse caso, como já existe um caminho entre as duas cidades, os dois grupos irão discutir sobre qual o jeito certo de comer cachorro-quente.

Assim, dado o mapa atual do reino, ajude Breno a calcular, para cada um dos Q cenários considerados, qual o **maior** número de estradas que ele poderia construir caso os Puristas tenham sua

sede na cidade A_i e os Purê-istas, na cidade B_i . Em especial, se já existir um caminho entre A_i e B_i , avise Breno que haverá uma grande confusão no reino.

Entrada

A primeira linha da entrada contém três inteiros N, M e Q , indicando o número de cidades, o número de estradas e o número de cenários considerados pelo rei, respectivamente.

As próximas M 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 dos cenários. A i -ésima dessas linhas contém dois inteiros A_i e B_i , representando uma situação considerada pelo rei.

Saída

A saída deve conter Q linhas.

A i -ésima linha deve conter um único inteiro. Se já existir um caminho entre A_i e B_i , imprima -1. Caso contrário, imprima o número máximo de estradas que podem ser construídas sem que exista um caminho entre A_i e B_i .

Restrições

- $2 \leq N \leq 10^5$.
- $0 \leq M \leq 2 \times 10^5$.
- $1 \leq Q \leq 10^5$.
- $1 \leq u_j, v_j \leq N$ para todo $1 \leq j \leq M$.
- $u_j \neq v_j$ para todo $1 \leq j \leq M$.
- $1 \leq A_i, B_i \leq N$ para todo $1 \leq i \leq Q$.

Para competidores que utilizam C++ ou Java: Observe que valores na saída podem ser muito grandes 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 (10 pontos):** Não existe nenhuma estrada que conecta B_i a outra cidade para todo $1 \leq i \leq Q$.
- **Subtarefa 3 (15 pontos):** $N \leq 100$ e $Q = 1$. Além disso, existem duas cidades C_1 e C_2 tal que, toda cidade p possui um caminho até C_1 ou até C_2 , mas não até ambos. Ou seja, o grafo do reino possui exatamente duas componentes.
- **Subtarefa 4 (11 pontos):** Existem duas cidades C_1 e C_2 tal que, toda cidade p possui um caminho até C_1 ou C_2 , mas não até ambos. Ou seja, o grafo do reino possui exatamente duas componentes.

- **Subtarefa 5 (18 pontos):** Existem três cidades C_1, C_2 e C_3 tal que, toda cidade p possui um caminho até exatamente uma cidade entre C_1, C_2 e C_3 . Ou seja, o grafo do reino possui exatamente três componentes.
- **Subtarefa 6 (21 pontos):** $Q = 1$.
- **Subtarefa 7 (25 pontos):** Sem restrições adicionais.

Exemplos

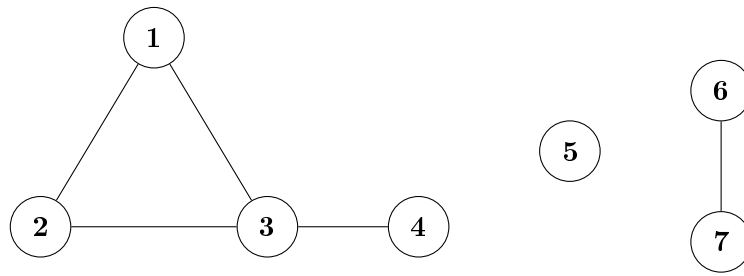
Exemplo de entrada 1	Exemplo de saída 1
9 9 2 1 2 1 3 1 4 2 3 3 4 5 6 7 8 7 9 8 9 2 8 1 4	9 -1

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

Além disso, no segundo cenário, como existe um caminho entre as cidades 1 e 4, a resposta é -1.

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

Explicação do exemplo 2: Este exemplo satisfaz as restrições da subtarefa 2. Este é o formato do reino:



No primeiro cenário, os Puristas e Purê-istas elegeram a mesma cidade como sede, portanto, a resposta é -1.

No segundo cenário, é possível construir no máximo 10 estradas, listadas a seguir: (1, 4), (2, 4), (1, 6), (1, 7), (2, 6), (2, 7), (3, 6), (3, 7), (4, 6) e (4, 7)

Exemplo de entrada 3	Exemplo de saída 3
8 7 1 6 5 6 7 3 2 2 5 4 1 4 8 1 8 4 3	6

Explicação do exemplo 3: Este exemplo satisfaz as restrições das subtarefas 3 e 4.

Exemplo de entrada 4	Exemplo de saída 4
10 8 3 10 6 7 5 7 8 9 4 8 4 2 3 2 1 3 1 6 1 10 4 2 8	21 21 16

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

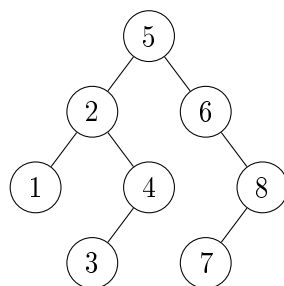
Exemplo de entrada 5	Exemplo de saída 5
10 7 10	29
9 8	29
6 7	-1
6 10	29
7 10	17
4 5	-1
4 3	22
5 3	22
1 10	29
1 10	22
2 2	
2 10	
3 7	
10 7	
4 9	
5 8	
1 10	
9 3	

BST

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

Lucas passou o final de semana inteiro estudando para a olimpíada de informática e finalmente descobriu a famosa BST (Binary Search Tree). Uma BST é uma estrutura de dados em formato de árvore onde cada número inserido segue uma regra de organização rigorosa. Partindo do topo da árvore, comparamos o novo número com o número atual, se for menor ele desce para a esquerda, se for maior, ele desce para a direita, repetindo esse processo de comparação até encontrar um espaço vazio para se fixar.

Lucas ficou empolgado e desenhou um exemplo no quadro para sua amiga Beatriz.



Ele explicou que, esse é o resultado ao inserir a sequência 5, 2, 6, 4, 1, 8, 7, 3, pois:

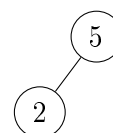
- O número 5 se torna o topo da árvore, por ser o primeiro número.
- O 2, sendo menor que 5, vai para a sua esquerda. Perceba que o 2 só é comparado com o 5.
- O 6, sendo maior que 5, vai para a sua direita. Perceba que o 6 só é comparado com o 5.
- O 4 vai para a esquerda do 5, mas como o 2 já está lá e 4 é maior que 2, ele vai para a direita do 2. Perceba que o 4 é comparado com o 5 e com o 2.
- O 1 vai para a esquerda do 5, e então para a esquerda do 2. Perceba que o 1 é comparado com o 5 e com o 2.
- O 8 vai para a direita do 5, e então para a direita do 6. Perceba que o 8 é comparado com o 5 e com o 6.
- O 7 vai para a direita do 5, depois para a direita do 6, e então para a esquerda do 8. Perceba que o 7 é comparado com o 5, com o 6 e com o 8.
- O 3 vai para a esquerda do 5, depois para a direita do 2, e então para a esquerda do 4. Perceba que o 3 é comparado com o 5, com o 2 e com o 4.

Além disso, podemos acompanhar a quantidade de comparações deste processo, da seguinte forma:

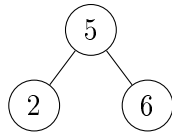
Inserindo: 5 (0 comparações)



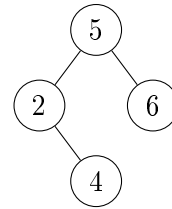
Inserindo: 2 (1 comparação)



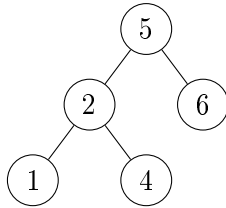
Inserindo: 6 (1 comparação)



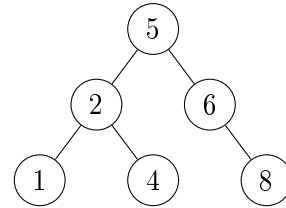
Inserindo: 4 (2 comparações)



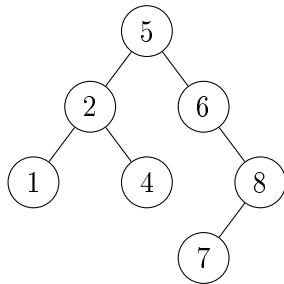
Inserindo: 1 (2 comparações)



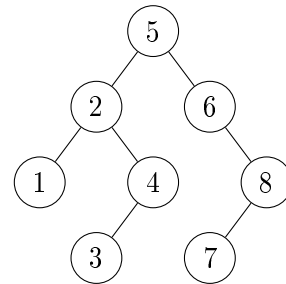
Inserindo: 8 (2 comparações)



Inserindo: 7 (3 comparações)



Inserindo: 3 (3 comparações)



Ao todo temos $0 + 1 + 1 + 2 + 2 + 2 + 3 + 3 = 14$ comparações.

Beatriz acompanhou cada passo com atenção. Apesar de não saber o que era uma BST até aquele momento, ela tem uma mente muito afiada para padrões e entendeu rapidamente como a estrutura funcionava e como as quantidades de comparações variavam.

Para testar o amigo, ela disse que saberia calcular de cabeça o o número exato de comparações que a árvore inteira iria gerar, mesmo para uma **permutação** enorme.

Vale lembrar que uma **permutação** de tamanho 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**.

Lucas ficou perplexo tentando entender como ela sabia calcular esse valor tão rápido sem desenhar a árvore. Como ele não conseguiu chegar ao resultado a tempo, Lucas está pedindo sua ajuda para testar a habilidade de Beatriz. Dada uma permutação de tamanho N , calcule quantas comparações são feitas ao criar uma BST a partir dela.

Entrada

A primeira linha da entrada contém um inteiro N , o tamanho da permutação.

A segunda linha da entrada contém N inteiros $p_1, p_2, \dots, p_N$, representando a permutação.

Saída

A saída deve conter uma única linha com um inteiro, representando o número de comparações feitas

ao construir a BST a partir da permutação dada.

Restrições

- $1 \leq N \leq 10^5$.
- p é uma permutação de tamanho N .

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 (7 pontos):** $p_i = i$ para todo $1 \leq i \leq N$.
- **Subtarefa 3 (9 pontos):** A permutação é tal que primeiro aparecem todos os números ímpares em ordem crescente, e depois todos os pares em ordem crescente (conforme o exemplo 2).
- **Subtarefa 4 (17 pontos):** $N \leq 1000$.
- **Subtarefa 5 (12 pontos):** É garantido que o número de comparações feitas ao inserir qualquer número é no máximo 100.
- **Subtarefa 6 (24 pontos):** $-10 \leq p_i - i \leq 10$ para todo $1 \leq i \leq N$.
- **Subtarefa 7 (31 pontos):** Sem restrições adicionais.

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
8 5 2 6 4 1 8 7 3	14

Explicação do exemplo 1: Este exemplo é o mesmo usado no enunciado.

Exemplo de entrada 2	Exemplo de saída 2
5 1 2 3 4 5	10

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

Exemplo de entrada 3	Exemplo de saída 3
10 1 3 5 7 9 2 4 6 8 10	29

Explicação do exemplo 3: Este exemplo satisfaz as restrições da subtarefa 3.