Olimpíada Brasileira de Informática

OBI2026

Caderno de Tarefas

Modalidade Programação • Nível Sênior • Fase 1

10 a 12 de Junho de 2026

A PROVA TEM DURAÇÃO DE 2 horas

Promoção:

Sociedade Brasileira de Computação

Apoio:

Coordenação:

v1.0

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 9 páginas (não contando a folha de rosto), numeradas de 1 a 9. 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 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).
- Você pode submeter até 50 soluções para cada tarefa. A pontuação total de cada tarefa é a melhor pontuação entre todas as submissões. Se a tarefa tem sub-tarefas, para cada sub-tarefa é considerada a melhor pontuação entre todas as submissões.
- 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 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.

Elevador

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

Devido ao crescente destaque do país nas competições internacionais de programação, a OBI (Olimpíada Brasileira de Informática) passou por um processo de expansão e agora é dona de um prédio de 100 andares! No entanto, a construção é um pouco antiga e possui um erro básico em seu projeto: há apenas um elevador. Consequentemente, seus funcionários perdem muito tempo para se mover entre os andares do prédio, o que afeta a produtividade da empresa.

A fim de analisar o impacto desse problema, a OBI decidiu estudar o tempo gasto na movimentação do elevador ao longo de um dia de trabalho. Nesse período, o elevador fez N paradas em sequência, denotadas pela lista $A_1, A_2, \ldots, A_N$. O deslocamento entre essas paradas é simples e preciso, de forma que o tempo gasto para subir ou descer um andar é de **exatamente** 1 segundo. Além disso, para os propósitos dessa pesquisa, o tempo de entrada e saída de pessoas no elevador é irrelevante.

Por exemplo, para $N = 3$ e as seguintes paradas: $A_1 = 1, A_2 = 9$ e $A_3 = 5$, o elevador demoraria 8 segundos para subir do 1º ao 9º andar e, em seguida, 4 segundos para descer do 9º ao 5º andar. Ou seja, seu tempo total de deslocamento seria $8 + 4 = 12$ segundos.

Essa análise é muito importante para a OBI, mas seu departamento de pesquisa e desenvolvimento está muito ocupado com a criação de um novo ambiente de provas para a olimpíada. Assim, dada a quantidade N e a lista $A_1, A_2, \ldots, A_N$ de paradas do elevador, ajude a OBI a calcular por quanto tempo o elevador se moveu durante o dia.

Entrada

A primeira linha da entrada contém um único inteiro N, a quantidade de paradas.

A segunda linha da entrada contém N inteiros $A_1, A_2, \ldots, A_N$, os andares pelos quais o elevador passou ao longo do dia, em ordem.

Saída

A saída deve conter um único inteiro, o tempo total de deslocamento do elevador.

Restrições

- $3 \leq N \leq 100$.
- $1 \leq A_i \leq 100$, para todo $1 \leq i \leq N$.
- $A_i \neq A_{i+1}$, para todo $1 \leq i < N$.
- $A_1 = 1$, ou seja, o elevador sempre começa no andar 1.

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 (20 pontos):** $N = 3$.
- **Subtarefa 3 (30 pontos):** $A_i < A_{i+1}$, para todo $1 \leq i < N$.
- **Subtarefa 4 (50 pontos):** Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
3 1 9 5	12

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

Exemplo de entrada 2	**Exemplo de saída 2**
4 1 7 9 6	11

Exemplo de entrada 3	**Exemplo de saída 3**
5 1 3 8 11 20	19

Explicação do exemplo 3: Perceba que este exemplo satisfaz as restrições da subtarefa 3.

Restaurante

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

Você trabalha no restaurante que faz o melhor pão com ovo da cidade, e por isso está sempre lotado. Sendo assim, você ficou responsável por cuidar da alocação das mesas do restaurante.

As mesas desse restaurante são sempre de 4 lugares, por conta disso, o sistema de reserva só aceita grupos de 1 a 4 pessoas. Devido a fama do restaurante, todos aceitam dividir mesas com pessoas desconhecidas, desde que as pessoas em um grupo se sentem na mesma mesa.

Sua tarefa é, dadas as quantidades de grupos com uma, duas, três e quatro pessoas, do sistema de reserva para o dia atual, determine a menor quantidade de mesas para alocar todos os grupos.

Entrada

A primeira e única linha da entrada contém quatro inteiros G_1, G_2, G_3, G_4, as quantidades de grupos com uma, duas, três, e quatro pessoas, respectivamente.

Saída

A saída deve conter uma única linha contendo um inteiro, representando a quantidade mínima de mesas necessária para alocar todos os grupos.

Restrições

É garantido que todo caso de teste satisfaz as restrições a seguir.

- $0 \leq G_1,\ G_2,\ G_3,\ G_4 \leq 10$

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 (20 pontos):** $G_2 = 0,\ G_3 = 0,\ G_4 = 0$, ou seja, só existem grupos de uma pessoa.
- **Subtarefa 3 (80 pontos):** Sem restrições adicionais.

Exemplos

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

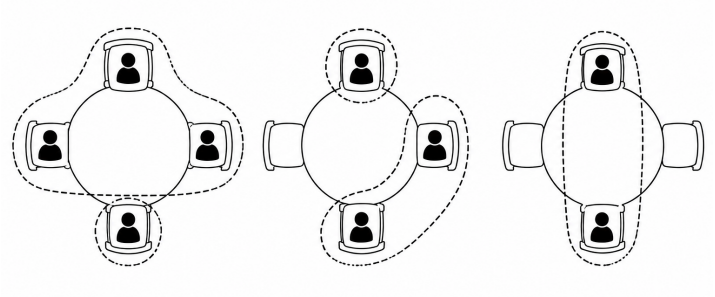
Explicação do exemplo 1: Nesse exemplo, temos:

- 2 grupos com uma pessoa
- 2 grupos com duas pessoas
- 1 grupo com três pessoas
- nenhum grupo com quatro pessoas

Com 3 mesas podemos alocar os grupos da seguinte forma:

- Mesa 1: um grupo com três pessoas e um grupo com uma pessoa
- Mesa 2: um grupo com uma pessoa e um grupo com duas pessoas
- Mesa 3: um grupo com duas pessoas

Conforme a imagem a seguir:



Não é possível alocar todos os grupos com menos de 3 mesas.

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

Explicação do exemplo 2: Nesse exemplo temos apenas 7 grupos com uma pessoa cada.
Com 2 mesas podemos alocar os grupos da seguinte forma:

- Mesa 1: quatro grupos com uma pessoa
- Mesa 2: três grupos com uma pessoa

Perceba que esse exemplo satisfaz as restrições da subtarefa 2.

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

Encontro de Amigas

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

Ana, Beatriz e Carolina são grandes amigas de infância e estudaram juntas a vida toda. Elas tinham notas excelentes e, além disso, competiram diversas vezes na Olimpíada Brasileira de Informática, ganhando diversas medalhas tanto nacionais quanto internacionais. Por causa disso, ao se formarem no ensino médio, as amigas foram aceitas em universidades ao redor de todo o mundo!

Elas comemoraram muito as aprovações, mas ficaram com medo de perderem o contato com o tempo. Isso porque, por estarem em universidades e países diferentes, a distância faz com que elas nem sempre possam se ver. Assim, Ana sugeriu que elas se encontrassem durante as férias de julho, quando estariam de volta no Brasil, para poderem colocar a conversa em dia. Beatriz e Carolina gostaram da ideia, mas repararam que, como cada amiga estará no país em um período de tempo diferente, marcar um encontro com **todas** presentes pode ser complicado.

Solucionar isso não seria difícil para programadoras experientes como elas, mas as garotas decidiram desafiar os atuais competidores da OBI a resolver esse problema. Assim, dados os intervalos de dias em que Ana, Beatriz e Carolina estarão no Brasil, calcule em quantos dias elas podem marcar um encontro com as três presentes.

Entrada

A primeira linha da entrada contém um inteiro A_1, o primeiro dia em que Ana estará no Brasil. A segunda linha da entrada contém um inteiro A_2, o último dia em que Ana estará no Brasil.

A terceira linha da entrada contém um inteiro B_1, o primeiro dia em que Beatriz estará no Brasil. A quarta linha da entrada contém um inteiro B_2, o último dia em que Beatriz estará no Brasil.

A quinta linha da entrada contém um inteiro C_1, o primeiro dia em que Carolina estará no Brasil. A sexta linha da entrada contém um inteiro C_2, o último dia em que Carolina estará no Brasil.

Saída

A saída deve conter uma única linha com um único inteiro, a quantidade de dias em que as amigas podem marcar um encontro com todas presentes.

Restrições

É garantido que todo caso de teste satisfaz as restrições abaixo.

- $1 \leq A_1 \leq A_2 \leq 31$
- $1 \leq B_1 \leq B_2 \leq 31$
- $1 \leq C_1 \leq C_2 \leq 31$

Informações sobre a pontuação

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

- **Subtarefa 1 (0 pontos)**: Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.

- **Subtarefa 2 (30 pontos):** $A_1 = A_2$.
- **Subtarefa 3 (70 pontos):** Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
1 15 5 20 12 18	4

Explicação do exemplo 1: Nesse caso os únicos dias em que as três amigas podem estar juntas são: 12, 13, 14 e 15.

Exemplo de entrada 2	**Exemplo de saída 2**
7 7 1 5 10 30	0

Explicação do exemplo 2: Nesse caso, não há nenhum dia em que todas as amigas estarão no Brasil. Perceba que esse exemplo satisfaz as restrições da subtarefa 2.

Torre de Números

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

Ricardinho é um menino prodígio e, com apenas 3 anos, aprendeu a contar até 9999. Vendo seu potencial, os pais do garoto compraram uma caixa com diversos blocos numerados de 0 a 9 para ele brincar. A criança adorou o presente e, em apenas alguns dias, já havia aprendido adição e subtração.

Para testar seu aprendizado, o garoto decidiu construir uma torre de números. Inicialmente, Ricardinho pega 4 blocos **não todos iguais** na caixa e os coloca em alguma ordem na base da torre, formando um número N. Em seguida, o garoto realiza o seguinte processo:

- Ricardinho pega quatro blocos na caixa, iguais aos quatro blocos atualmente no topo da torre. Depois, ele reorganiza esses blocos do **menor para o maior**, formando um número X_1 de quatro algarismos.
- Analogamente, o menino pega outros quatro blocos na caixa, também iguais aos blocos atualmente no topo da torre. Em seguida, ele reorganiza esses blocos do **maior para o menor**, formando um número X_2 de quatro algarismos.
- Por fim, ele calcula $X = X_2 - X_1$. Caso X já esteja na torre, ele para de construir a torre. Caso contrário, Ricardinho pega quatro blocos na caixa, coloca o número X no topo da torre e repete o processo.

O garoto sabe que todos os números que ele conhece podem ser formados por quatro blocos, desde que ele coloque blocos com zeros à esquerda quando necessário. Além disso, a caixa de blocos é muito grande, de forma que há blocos suficientes para construir qualquer torre de números.

Por exemplo, para $N = 8082$, a torre será formada pelos seguintes passos:

- Inicialmente, a torre começa com 8082.
- O número no topo da torre é formado pelos blocos $8, 0, 8$ e 2. Assim, $X_1 = 0288$ e $X_2 = 8820$. Portanto, $X = X_2 - X_1 = 8820 - 0288 = 8532$. Como 8532 não está na torre, ele é adicionado no topo.
- O número no topo da torre é formado pelos blocos $8, 5, 3$ e 2. Assim, $X_1 = 2358$ e $X_2 = 8532$. Portanto, $X = X_2 - X_1 = 8532 - 2358 = 6174$. Como 6174 não está na torre, ele é adicionado no topo.
- O número no topo da torre é formado pelos blocos $6, 1, 7$ e 4. Assim, $X_1 = 1467$ e $X_2 = 7641$. Portanto, $X = X_2 - X_1 = 7641 - 1467 = 6174$. Como 6174 já está na torre, ele não é adicionado no topo, e a torre é finalizada.

No fim, esta é a aparência da torre formada:

6	1	7	4
8	5	3	2
8	0	8	2

Ricardinho está muito animado com sua nova brincadeira, mas ainda é uma criança pequena e está cansado depois de fazer tantas contas. Dessa forma, dado o número N na base da torre, ajude Ricardinho a construir a torre, e determine qual será a aparência final da torre de números.

Entrada

A entrada contém uma única linha com um único inteiro N, o número na base da torre.

O número será dado sem zeros à esquerda, mesmo que ele possua menos de quatro algarismos.

Saída

A saída deve conter a torre de números formada pelo procedimento criado por Ricardinho.

A primeira linha da saída deve conter o número na base da torre, a segunda linha deve conter o número no segundo andar da torre, e assim sucessivamente.

Imprima os números sem zeros à esquerda, mesmo que ele possua menos de quatro algarismos.

Restrições

É garantido que todo caso de teste satisfaz as restrições abaixo.

- $1 \leq N \leq 9999$
- N não possui todos os algarismos iguais.

Informações sobre a pontuação

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

- **Subtarefa 1 (0 pontos)**: Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (30 pontos):** $N = 2026$.
- **Subtarefa 3 (70 pontos):** Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
8082	8082 8532 6174

Explicação do exemplo 1: Note que este é o exemplo descrito no enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
71	71 7083 8352 6174