

Competidor(a): \_\_\_\_\_

Número de inscrição: \_\_\_\_\_ – \_\_\_\_\_ (opcional)



# OBI2021

## Caderno de Tarefas

Modalidade Programação • Nível 2 • Fase 2 (Turno B)

1 de setembro de 2021

A PROVA TEM DURAÇÃO DE 3 horas

**Promoção:**



Sociedade Brasileira de Computação

**Apoio:**



# Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 7 páginas (não contando a folha de rosto), numeradas de 1 a 7. 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.

# Cálculo rápido

Nome do arquivo: “calculou.x”, onde  $x$  deve ser `c`, `cpp`, `pas`, `java`, `js` ou `py`

Algumas pessoas conseguem fazer cálculos matemáticos com uma velocidade impressionante. Laurinha tem essa habilidade! Um cálculo que ela consegue fazer muito rapidamente é, dados três números inteiros  $S$ ,  $A$ , e  $B$ , determinar quantos números do intervalo  $[A, B]$  têm a soma de seus dígitos igual a  $S$ .

Por exemplo, se  $S = 3$ ,  $A = 10$  e  $B = 30$ , então a resposta é 3, pois existem três números no intervalo  $[10, 30]$  cuja soma dos dígitos é igual a três: 12, 21 e 30.

Sua tarefa é escrever um programa de computador para, dados os três números, tentar calcular a resposta mais rapidamente do que Laurinha consegue.

## Entrada

A primeira linha da entrada contém um número inteiro  $S$ , o valor da soma dos dígitos. A segunda e a terceira linhas contêm respectivamente os inteiros  $A$  e  $B$ .

## Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, quantos números no intervalo dado têm a soma de dígitos indicada.

## Restrições

- $1 \leq S \leq 36$
- $1 \leq A \leq 10000$
- $1 \leq B \leq 10000$
- $A \leq B$

## Exemplos

<b>Exemplo de entrada 1</b> 3 10 30	<b>Exemplo de saída 1</b> 3
<b>Exemplo de entrada 2</b> 15 1 20	<b>Exemplo de saída 2</b> 0
<b>Exemplo de entrada 3</b> 1 1 10000	<b>Exemplo de saída 3</b> 5

# Lista palíndroma

Nome do arquivo: “lista.x”, onde x deve ser c, cpp, pas, java, js ou py

Uma palavra é chamada de palíndromo se a primeira letra da palavra é igual à última letra da palavra, a segunda letra é igual à penúltima letra, a terceira letra é igual à antepenúltima letra, e assim por diante. Por exemplo, as palavras *osso* e *sopapos* são palíndromos.

Nesta tarefa estamos interessados não em palavras, mas em listas de números inteiros. Nesse caso, vamos definir que uma lista é palíndroma se  $L[i] = L[N - i + 1]$ , onde  $L[i]$  representa o  $i$ -ésimo elemento da lista (note que nesta notação o índices variam de 1 a  $N$ ).

Você pode modificar uma lista usando a operação de *contração*, que é definida da seguinte forma: escolha dois elementos adjacentes da lista e substitua os dois elementos por um único elemento de valor igual à soma dos elementos substituídos. Note que ao efetuar uma operação de contração o número de elementos da lista decresce de um elemento.

Dada uma lista de números inteiros, você deve escrever um programa para determinar o menor número de operações de contração que devem ser realizadas de modo que a lista resultante seja palíndroma.

## Entrada

A primeira linha da entrada contém um inteiro  $N$ , o número de elementos da lista. A segunda linha contém  $N$  inteiros  $L_i$ , os elementos da lista.

## Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o menor número de operações de contração necessárias para tornar a lista palíndroma.

## Restrições

- $1 \leq N \leq 10^6$
- $1 \leq L_i \leq 10^9$ , para  $1 \leq i \leq N$

## Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 30 pontos,  $N \leq 10$ .
- Para um conjunto de casos de testes valendo outros 30 pontos  $N \leq 10^3$ .
- Para um conjunto de casos de testes valendo outros 40 pontos, nenhuma restrição adicional.

## Exemplos

<b>Exemplo de entrada 1</b> 5 10 60 20 40 10	<b>Exemplo de saída 1</b> 1
<b>Exemplo de entrada 2</b> 5 999 1 999 1 999	<b>Exemplo de saída 2</b> 0

<b>Exemplo de entrada 3</b>	<b>Exemplo de saída 3</b>
4 10 40 30 20	2

# Poligrama

Nome do arquivo: “poligrama.x”, onde  $x$  deve ser `c`, `cpp`, `pas`, `java`, `js` ou `py`

Duas palavras  $A$  e  $B$  são *anagramas entre si* se podemos transformar a palavra  $A$  na palavra  $B$  apenas trocando de posição as letras da palavra  $A$ . Por exemplo, “duetos” e “estudo” são anagramas entre si. Um outro exemplo é “bba” e “bab”.

Vamos chamar de *poligrama* uma palavra que consiste na concatenação de duas ou mais palavras que são anagramas entre si. A primeira dessas palavras é chamada de *raiz* do poligrama. Por exemplo, a palavra “bbabab” é um poligrama com raiz “bba”, pois ela é a concatenação dos anagramas “bba” e “bab”.

Dada uma palavra, escreva um programa que determine se ela é um poligrama e encontre a sua raiz.

## Entrada

A primeira linha da entrada contém um inteiro  $N$ , indicando o número de letras da palavra. A segunda linha contém a palavra  $P$ .

## Saída

Seu programa deve produzir uma única linha. Se a palavra dada é um poligrama, a linha deve conter a raiz do poligrama. Caso contrário, a linha deve conter o caractere asterisco (\*). Se houver mais de uma raiz possível, seu programa deve imprimir a de menor comprimento.

## Restrições

- $1 \leq N \leq 100000$
- O número de caracteres de  $P$  é igual a  $N$ .
- Os únicos caracteres em  $P$  são letras minúsculas não acentuadas.

## Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 40 pontos,  $N \leq 1000$ .
- Para um conjunto de casos de testes valendo outros 70 pontos, nenhuma restrição adicional.

## Exemplos

<b>Exemplo de entrada 1</b> 5 xxxxx	<b>Exemplo de saída 1</b> x
<b>Exemplo de entrada 2</b> 2 xy	<b>Exemplo de saída 2</b> *
<b>Exemplo de entrada 3</b> 6 bbabab	<b>Exemplo de saída 3</b> bba

# Senha da Vó Zinha

Nome do arquivo: “`senha.x`”, onde  $x$  deve ser `c`, `cpp`, `pas`, `java`, `js` ou `py`

Vó Zinha foi sempre muito cuidadosa com as senhas que usa para suas atividades na Internet, como compras, redes sociais e correio eletrônico, e é especialmente cuidadosa com a senha do banco. No entanto, como está ficando um pouco esquecida das coisas, ela resolveu deixar sua senha do banco escrita, para o caso de necessidade. Obviamente, ela não escreveu simplesmente a senha num papel! Ela inventou uma forma de proteger a senha, mesmo estando escrita, e contou somente para você como fazer para recuperar a senha.

Com um pedaço de papel que Vó Zinha guardou na gaveta onde guarda também suas meias ela fez o seguinte:

- inicialmente escreveu a senha do banco no papel;
- então borrou algumas das letras da senha que tinha escrito de forma que não possam ser lidas;
- para cada uma das letras borradas, ela escreveu no papel uma palavra com  $K$  letras;
- por fim, ela escreveu no papel um número inteiro  $P$ .

Vó Zinha então contou para você como recuperar a senha:

- utilizando as listas de palavras no papel, substitua cada letra borrada da senha por uma das letras da respectiva lista, obtendo assim possíveis senhas;
- crie uma lista contendo todas as possíveis senhas obtidas no passo anterior;
- ordene a lista de possíveis senhas em ordem lexicograficamente crescente;
- a senha correta é a  $P$ -ésima possível senha na lista ordenada.

Por exemplo, considere que no papel esteja escrito (● representa uma letra borrada):

```
x●yy●z
ab
cd
3
```

Fazendo as substituições, a lista das possíveis senhas é  $xayycz$ ,  $xbyycz$ ,  $xayydz$ ,  $xbyydz$ . Ordenando as possíveis senha obtemos  $xayycz$ ,  $xayydz$ ,  $xbyycz$ ,  $xbyydz$ , e portanto a senha correta é  $xbyycz$  (a terceira da lista ordenada).

Hoje Vó Zinha precisa pagar uma conta pela internet e não se recorda da senha do banco. Ela pediu que você pegue o pedaço de papel guardado na gaveta e a ajude a recuperar a senha.

## Entrada

A primeira linha da entrada contém três números inteiros  $N$ ,  $M$  e  $K$ , respectivamente o número de caracteres da senha, o número de letras borradas da senha e o comprimento de cada palavra. A segunda linha contém uma cadeia de caracteres de comprimento  $N$ , a senha escrita no papel, com o caractere ‘#’ (cerquilha) representando as letras borradas. Cada uma das  $M$  linhas seguintes contém uma palavra  $S_i$ , sendo que a  $S_i$ -ésima palavra contém as letras para substituir a  $i$ -ésima letra borrada da senha. A última linha contém um número inteiro  $P$ , o número de ordem da senha correta na lista ordenada de possíveis senhas.

**Saída**

Seu programa deve produzir uma única linha, contendo uma única cadeia de caracteres, a senha correta.

**Restrições**

- $1 \leq N \leq 500$
- $1 \leq M \leq N$
- $1 \leq K \leq 26$
- comprimento de  $S_i = K$ , para  $1 \leq i \leq M$
- $1 \leq P \leq 10^9$
- Na senha com as letras “borradas”, cada caractere é uma letra minúscula não acentuada ou o caractere #.
- Nas palavras com as letras que podem substituir as letras borradas da senha, cada caractere é uma letra minúscula não acentuada.
- $P \leq$  número total de possíveis senhas

**Informações sobre a pontuação**

- Para um conjunto de casos de testes valendo 30 pontos,  $M = 1$ .
- Para um conjunto de casos de testes valendo outros 30 pontos,  $M \leq 8$  e  $K \leq 6$ .
- Para um conjunto de casos de testes valendo outros 40 pontos, nenhuma restrição adicional.

**Exemplos**

<p><b>Exemplo de entrada 1</b></p> <pre>6 2 2 x#yy#z ab cd 3</pre>	<p><b>Exemplo de saída 1</b></p> <pre>xbyycz</pre>
<p><b>Exemplo de entrada 2</b></p> <pre>4 1 3 #gof abc 2</pre>	<p><b>Exemplo de saída 2</b></p> <pre>bgof</pre>