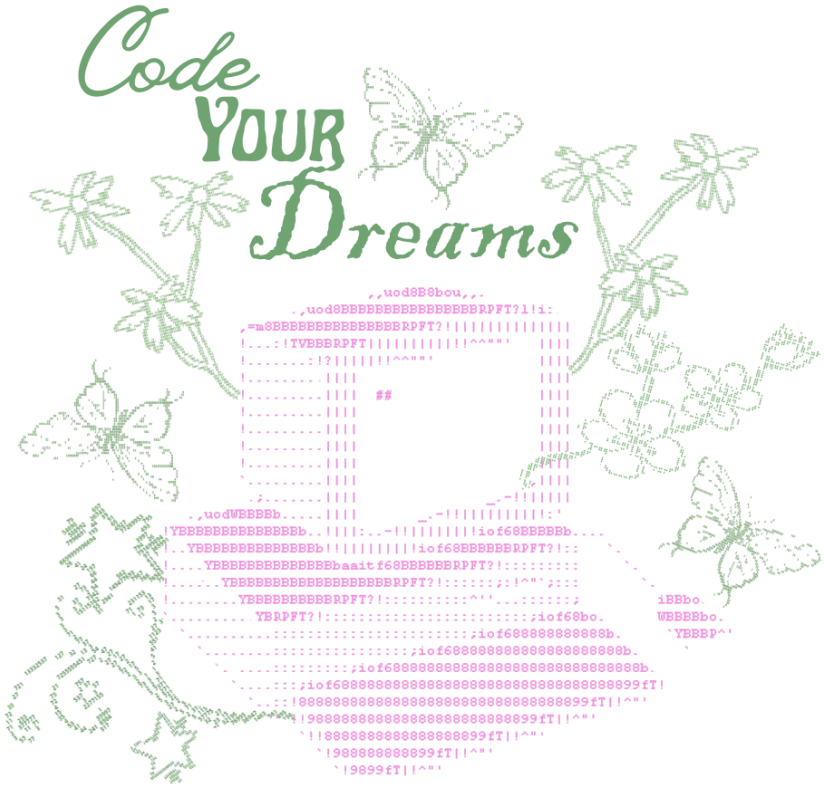
Guia da Maratona



Obs: Este não é o caderno de prova, é um guia sobre como é estruturada a prova e como funcionam as submissões.

Lembretes

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova, entretanto, o mesmo não vale para materiais dispostos eletronicamente.
- A correção é automatizada, portanto, siga atentamente as exigências da tarefa quanto ao formato da entrada e saída conforme as amostras dos exemplos. Deve-se considerar entradas e saídas padrão;
- Para cada problema, além dos testes públicos, o juiz executará a sua submissão contra uma série de testes secretos para fornecer um parecer sobre a correção do programa.
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. Lembre-se que as soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas;
- Utilize a aba *clarification* para dúvidas da prova. Os juízes podem opcionalmente atendê-lo com respostas acessíveis a todos;

C/C++

- Seu programa deve retornar zero, executando, como último comando, `return 0` ou `exit 0`.

Java

- Não declare `package` no seu programa Java.
- Note que a convenção para o nome do arquivo fonte deve ser obedecida, o que significa que o nome de sua classe pública deve ser uma letra maiúscula igual a letra que identifica o problema.

Python

- Tenha cuidado ao selecionar a versão correta na submissão.

Formato do Enunciado

O enunciado descreve todas as informações necessárias à solução de um problema, desde o formato e os limites dos dados de entrada até a descrição do que deve ser solucionado.

Geralmente abaixo do título do problema estão os limites de tempo e memória daquela questão. Caso sejam violados, geram vereditos negativos (como será abordado mais à frente).

As próximas linhas são destinadas à descrição do problema. As últimas três partes de um enunciado são compostas pelas seções de **Entrada**, **Saída** e casos de exemplo.

Na seção de **Entrada** está descrito como os dados serão fornecidos ao seu código durante a execução, bem como os limites para esses dados. Na seção de **Saída** está descrito o formato esperado para a resposta do seu código.

Casos de teste

Uma questão é composta por múltiplos casos de teste, divididos em dois formatos.

Casos de exemplo (visíveis)

São os casos que aparecem na questão. Estão lá para exemplificar o formato descrito nas seções **Entrada** e **Saída** do enunciado e permitir que o maratonista teste a solução manualmente, verificando se o resultado do seu código corresponde ao **output** esperado.

Exemplo de Entrada	**Exemplo de Saída**
3 1 2 3	6

Casos ocultos

São os casos que de fato testam a corretude da solução. Cobrem os limites máximos e mínimos descritos na seção **Entrada**, além de casos de borda para garantir que a solução atende a todos os requisitos do problema. Esses casos não são visíveis ao maratonista e seguem o mesmo formato dos casos de exemplo.

Vereditos e exemplos

TIME LIMIT EXCEEDED - TLE

Códigos cujo tempo de execução ultrapassem o limite estabelecido recebem esse veredito. Para obter uma estimativa de qual o tempo de execução do seu código é possível executar seu código precedido do seguinte comando:

```
time
```

Se o limite for 1 segundo, o seguinte código receberia esse veredito:

```
#include <iostream>
using namespace std;

int main() {
    int n, ans = 0; cin >> n;
```

```
    for(int i = 0; i < 1e6+2; i++)
        for(int j = 0; j < 1e6+2; j++)
            ans = (ans + ((i+j) % 2 ? i : n));

    cout << ans << endl;
    return 0;
}
```

RUN TIME ERROR - RTE

Códigos em que algum erro ocorre durante a execução recebem esse veredito. As causas mais comuns são: acesso inválido à memória (como índices fora dos limites de um array), divisão por zero e estouro de pilha por recursão muito profunda.

COMPILATION ERROR - CE

Códigos de linguagens compiladas que não puderam ser compilados recebem esse veredito. Erros de sintaxe e uso incorreto de tipos são causas comuns.

WRONG ANSWER - WA

Códigos cuja saída não corresponde ao resultado esperado recebem esse veredito negativo.

MEMORY LIMIT EXCEEDED - MLE

Códigos cuja soma do espaço ocupado pelas variáveis ultrapassa o limite definido recebem esse veredito. Se o limite for 256 MB, o seguinte código estoura esse limite:

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, ans = 0; cin >> n;
    vector<long long> arr((int)(4e7+1));
    for(int i = 1; i < 4e7+1; i++)
        arr[i] = arr[i-1] + (i % 2 ? n : i);

    cout << arr[(int)(4e7)] << endl;
    return 0;
}
```

ACCEPTED - AC

Códigos cuja saída corresponde exatamente à saída esperada recebem o veredito positivo.

NO-OUTPUT

Códigos que não produzem nenhuma saída recebem esse veredito. Geralmente ocorre quando o programa termina sem realizar nenhuma impressão ou entra em loop infinito sem gerar saída.

Como funciona o juiz

O que é

O juiz é uma plataforma de correção automática utilizada em competições de programação. Ele recebe o seu código, compila, executa e compara a saída produzida pelo seu programa com a saída esperada pelos juízes. Geralmente em maratonas, são utilizadas as plataformas **DOMJudge** e **BOCA**.

Etapas de submissão

- Faça login na plataforma com o usuário e senha fornecidos pela organização.
- Escolha o problema que deseja resolver.
- Selecione a linguagem correta (C, C++, Java ou Python).
- Envie apenas o arquivo com o código-fonte (sem compactar ou adicionar arquivos extras).
- O sistema compila automaticamente e testa o seu programa com várias entradas secretas.

Regras gerais

- Seu programa deve ler a entrada **a partir da entrada padrão** e escrever a saída **na saída padrão**.
- Não escreva mensagens adicionais como "Digite um número" ou "Resultado:".
- As respostas devem ter exatamente o formato pedido no enunciado (incluindo espaços e quebras de linha).
- Não é permitido abrir, ler ou escrever arquivos.
- Cada submissão é julgada automaticamente. Caso o programa não siga o formato exigido, ele será rejeitado.

Como compilar e testar o código

C/C++

- Primeiro, escreva o seu código em um arquivo, por exemplo `a.cpp`.
- Para compilar, abra o terminal na pasta onde o arquivo está e digite:

```
g++ a.cpp   # para c++
gcc a.c     # para c
```

  Isso cria um arquivo executável:

  - `a.out`, no Linux;
  - `a.exe`, no Windows.

- Depois de compilado, você pode testar o programa executando:

```
./a.out    # Linux
a.exe      # Windows
```

Java

- Escreva o seu programa em um arquivo `.java`, cujo nome deve ser igual ao da classe pública. Por exemplo, `A.java` deve conter uma classe pública chamada `A`.
- Para compilar, digite:

```
javac A.java
```

  Isso gera o arquivo `A.class`, que é o programa compilado.

- Para executar o programa compilado:

```
java A
```

  O comando `java` roda o arquivo `.class` que foi criado na compilação.

Python

- Em Python, não é necessário compilar. Basta escrever o código em um arquivo, por exemplo `a.py`.
- Para executar e testar o programa, digite:

```
python3 a.py
```

  O interpretador do Python lê o código e executa diretamente.

Dicas e exemplos de código

Python

- **Leitura e impressão:** Em Python, usamos `input()` para ler dados e `print()` para exibir resultados.

```
x = int(input())              # Lê um número inteiro
nome = input()                # Lê uma string
a, b = map(int, input().split())  # Lê dois inteiros na mesma linha

print(x)                      # Imprime o valor de x
print(a, b)                   # Imprime os dois valores separados por espaço
print("Resultado:", a + b)    # Imprime texto e cálculo juntos
```

- **Estruturas condicionais:** São usadas para executar diferentes ações dependendo de uma condição.

```
if x > 0:
    print("positivo")
elif x == 0:
    print("zero")
else:
    print("negativo")
```

- **Laços de repetição:** Servem para repetir instruções várias vezes. `for` é usado quando sabemos o número de repetições, e `while` quando repetimos até uma condição parar de ser verdadeira.

```
for i in range(5):
    print(i)   # Imprime 0, 1, 2, 3, 4

while x > 0:
    print(x)
    x = x - 1
```

C++

- **Leitura e impressão:** Em C++, usamos `cin` para ler e `cout` para imprimir.

```
#include <iostream>
using namespace std;

int main() {
    int x, a, b;
    string nome;

    cin >> x;           // Lê um número inteiro
    cin >> nome;        // Lê uma palavra
    cin >> a >> b;      // Lê dois inteiros na mesma linha

    cout << x << endl;  // Imprime x e quebra linha
    cout << a << " " << b << endl;
    cout << "Resultado: " << a + b << endl;

    return 0;
}
```

- **Estruturas condicionais:** Permitem escolher blocos de código a executar conforme uma condição.

```
if (x > 0) {
    cout << "positivo\n";
} else if (x == 0) {
    cout << "zero\n";
} else {
    cout << "negativo\n";
}
```

- **Laços de repetição:** São usados para repetir instruções. O `for` repete um número fixo de vezes, enquanto o `while` continua até a condição deixar de ser verdadeira.

```
for (int i = 0; i < 5; i++) {
    cout << i << endl;  // Imprime 0, 1, 2, 3, 4
}

while (x > 0) {
    cout << x << endl;
    x = x - 1;
}
```

Overflow

Em linguagens fortemente tipadas como **C++**, é necessário tomar cuidado com os limites definidos na seção **Entrada**. Uma variável do tipo ***int*** armazena inteiros de 32 bits com sinal,

ou seja, suporta valores de até 2.147.483.647 ($2^{31} - 1$). Números maiores que esse limite não serão armazenados corretamente nesse tipo.

Para evitar esse problema, recomenda-se calcular previamente o maior valor que uma variável da solução pode assumir e escolher a tipagem adequada — em muitos casos, o tipo ***long long*** (64 bits, suportando valores até $2^{63} - 1$) é a escolha correta. Esse cálculo também é essencial para determinar quando utilizar operações modulares a seu favor, nos casos em que o enunciado as exige.