



**Kaio Christaldo**  
**Fabricio Matsunaga**

# Apresentação Problema Motivador

## Problema C

### Collatz polinomial

Todos conhecem (ou já ouviram falar) da famosa Conjectura de Collatz: pegue um número inteiro positivo. Se ele for ímpar, multiplique por 3 e some 1. Se for par, divida por 2. Repita o processo até chegar em 1. Apesar de sua simplicidade, ninguém sabe provar se a sequência realmente sempre alcança 1, qualquer que seja o número inicial.

Aline, fã desse tipo de curiosidade, decidiu criar uma variação usando polinômios em vez de números. Para não complicar, ela trabalha apenas com polinômios cujos coeficientes são 0 ou 1, ou seja, cada potência de  $x$  aparece no máximo uma vez.

A brincadeira funciona assim:

- Se o polinômio possui termo constante (um termo que não depende de  $x$ ), Aline multiplica o polinômio por  $(x + 1)$  e depois soma 1. Caso algum coeficiente resultante seja igual a 2, o termo correspondente é descartado (observe que coeficientes maiores que 2 não podem surgir).
- Se o polinômio não possui termo constante, Aline divide o polinômio por  $x$ .

Esse processo se repete até que o polinômio se reduza a  $P(x) = 1$ .

Considere  $P(x) = x^3 + 1$ . No primeiro passo há termo constante, então calculamos:

$$(x^3 + 1) \cdot (x + 1) + 1 = x^4 + x^3 + x + 1 + 1.$$

Como o coeficiente do termo constante resulta em 2, esse termo é descartado, restando:

$$x^4 + x^3 + x.$$

Em seguida, como não há termo constante, dividimos por  $x$ :

$$x^3 + x^2 + 1.$$

Continuando:

- Passo 3:  $x^4 + x^2 + x$

- Passo 4:  $x^3 + x + 1$

- Passo 5:  $x^4 + x^3 + x^2$

- Passo 6:  $x^3 + x^2 + x$

- Passo 7:  $x^2 + x + 1$

- Passo 8:  $x^3$

- Passo 9:  $x^2$

- Passo 10:  $x$

- Passo 11: 1

No total, foram necessárias 11 operações para chegar ao polinômio  $P(x) = 1$ .

Aline precisa de ajuda para estudar essa variação da Conjectura de Collatz. Como fazer essas contas manualmente é suscetível a erros, escreva um programa que determine o número de operações necessárias até o polinômio se tornar  $P(x) = 1$ .

## ◀bitmask▶

**Bitmask (ou máscara de bits) é uma técnica que utiliza operações em nível de bits para representar conjuntos, estados ou combinações de forma compacta e eficiente.**

**Em C++, isso é feito manipulando inteiros (int, long long) como se fossem vetores de bits.**

## ◀bitmask▶ – Vantagens

### Vantagens:

- **Armazenar subconjuntos em um único número.**
- **Manipulação rápida (operações em  $O(1)$ ).**
- **Muito útil em DP, subconjuntos, grafos e problemas combinatórios.**

## ◀bitmask▶ – Operações Bitmask em C++

**Usamos Operações Bitwise para fazer bitmask**

**Bitwise AND (&) – retorna verdadeiro apenas se os dois bits estão ligados.**

**Bitwise OR (|) – retorna verdadeiro se pelo menos um dos dois bits esta ligado**

**Bitwise XOR (^) – retorna verdadeiro se os bit tem estados diferentes**

**Bitwise NOT (~) – retorna a negação.**

**Bitwise Left Shift (<<) – Movimenta os bits para a esquerda.**

**Bitwise Right Shift (>>) – Movimenta os bits para a direita.**

## ◀bitmask▶ – Operações Bitmask em C++

1. **Ligando um bit específico**
2. **Desligando um bit específico**
3. **flipando um bit específico**
4. **checando se um bit está ligado ou desligado**
5. **Checar o bit menos significativo ligado**

## ◀bitmask▶ – Ligando um bit específico

Setting a Specific bit

00001011	(11)
00100000	(1 << 5)
<hr/>	
00101011	(43)
<hr/>	

## ◀bitmask▶ – Ligando um bit específico



```
1  // C++ program to illustrate how to set a particular bit
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      // 11 = 00001011
8      int x = 11;
9
10     // setting fifth bit using bitmask
11     x = x | 1 << 5; // or x |= 1 << 5;
12     cout << "Result after setting the fifth bit: " << x ;
13
14     return 0;
15 }
```



## ◀bitmask▶ – Desligar um bit específico

Clearing a Specific bit

00001011	(11)
& 11110111	~(1 << 3)
<hr/>	
00000011	(3)
<hr/>	

## ◀bitmask▶ – Desligar um bit específico



```
1 // C++ program to illustrate how to clear a particular bit
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // 11 = 00001011
8     int x = 11;
9
10    // clearing bit at third position
11    x = x & ~(1 << 3); // or x &= ~(1 << 3);
12    cout << "Result after clearing the 3rd bit: " << x;
13
14    return 0;
15 }
```

## ◀bitmask▶ – flipando um bit específico

Toogling a Specific bit

$$\begin{array}{rcl} & 00001011 & (11) \\ \wedge & 00000001 & (1 \ll 0) \\ \hline & 00001010 & (10) \\ \hline \end{array}$$

## ◀bitmask▶ – flipando um bit específico



```
1  // C++ program to illustrate how to toggle a bit
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      // 11 = 00001011
8      int x = 11;
9      // toggling zeroth bit
10     x = x ^ 1 << 0; // or x = x ^ (1 << 0);
11     cout << "Result after toggling the zeroth bit: " << x;
12
13     return 0;
14 }
```

◀bitmask▶ – checando se um bit esta ligado ou desligado

Checking if a bit is Set

00001011	(11)
& 00001000	(1 << 3)
<hr/>	
00001000	(8)
<hr/>	

# ◀bitmask▶ – checando se um bit esta ligado ou desligado

```
1 // C++ program to check if the bit is set or not
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // 11 = 00001011
8     int x = 11;
9
10    // the AND will return a non zero number if the bit is
11    // set, otherwise it will return zero
12    if (x & (1 << 3)) {
13        cout << "Third bit is set\n";
14    }
15    else {
16        cout << "Third bit is not set\n";
17    }
18    return 0;
19 }
```

# ◀bitmask▶ – checando o bit menos significativo LSB



```
1 // C++ program to return the least significant set bit
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // 11 = 00001011
8     int x = 11;
9
10    x = x & (~x);
11    return 0;
12 }
```

# Lista de Exercícios

**1915A – Odd One Out**

**579A – Raising Bacteria**

**467B – Fedor and New Game**

**1915B – Not Quite Latin Square**

**1527A – And Then There Were K**

Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊