<bitset> e <bit>

Kaio Christaldo
Fabricio Matsunaga

‹Bitset›

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 $P(x) = 1$.

Apresentação Problema Motivador

Entrada

A primeira linha contém um inteiro N $(0 \leq N \leq 20)$, indicando o grau do polinômio.

A segunda linha contém $N + 1$ inteiros a_N, a_{N-1}, ..., a_0 (cada um igual a 0 ou 1), onde $a_i = 1$ indica que o termo x^i está presente no polinômio, e $a_i = 0$ indica que não está. Note que $a_N = 1$, já que o grau do polinômio é N.

Saída

Seu programa deve produzir uma única linha com um inteiro representando o número de operações necessárias até o polinômio se tornar $P(x) = 1$.

Exemplo de entrada 1	**Exemplo de saída 1**
3 1 0 0 1	11

Exemplo de entrada 2	**Exemplo de saída 2**
2 1 0 1	6

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

Exemplo de entrada 4	**Exemplo de saída 4**
0 1	0

‹bitset›

Bitset é um container que representa a sequencia de bits com tamanho fixo.
Um bitset permite a manipulação individual dos bits eficientemente, fazendo ele ser util em problemas relacionados com operações bitwise como checar flags e implementar representações binarias.

Bitset é definido como o modelo de classe std::bitset dentro do arquivo de cabeçalho ‹bitset›.

<bitset> Sintaxe

```
1  #include <bitset>
```

```
1  bitset<n> name;
```

n é o número de bits a serem alocados e name é o nome atribuído

‹bitset› Inicializando

Por padrão, quando um bitset de um determinado tamanho é criado, todos os seus bits são reinicializados, ou seja, 0. Podemos inicializá-lo com algum valor simplesmente passando-o ao seu construtor.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {

    // Default Initialization
    bitset<5> bnum(18);
    cout << bnum;
    return 0;
}
```

Aqui, o bitset bnum tem tamanho de 5 bits e armazena o valor decimal 18 em formato binário

‹bitset› Inicializando

Números binários também são representados como strings, então o bitset também fornece a facilidade de se inicializar a partir das strings.

```cpp
int main() {

    // Initialize bitset with value
    bitset<5> bs2("10010");

    cout << bs2 << endl;
    return 0;
}
```

A string deve representar um número binário válido, ou seja, todos os caracteres devem ser 0 ou 1. Caso contrário, poderá ocorrer um erro.

‹bitset› Setting, Resetting e Flipping

Setting significa colocar o bit na posição 1, e resetting significa torná-lo 0. Essas operações podem ser feitas usando as funções set() e reset() . A função flip() pode ser usada para setar o bit se ele não estiver setado e para reseting se estiver.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    bitset<5> bs(18);

    // Set 1st bit
    bs.set(0);
    cout << bs << endl;

    // Reset 2nd bit
    bs.reset(1);
    cout << bs << endl;

    // Flip 5th bit
    bs.flip(4);
    cout << bs;
    return 0;
}
```

‹bitset› Operadores Bitset

Operator	Operation
&	Bitwise **AND**
|	Bitwise **OR**
^	Bitwise **XOR**
>>=	Binary **Right shift** and assign
<<=	Binary **Left shift** and assign
&=	Assign the value of bitwise **AND** to the first bitset.
|=	Assign the value of bitwise **OR** to the first bitset.
^=	Assign the value of bitwise **XOR** to the first bitset.
~	Bitwise **NOT**

‹bitset› Operadores Bitset

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {

    // 18 = (10010)
    bitset<5> bs1(18);

    // 5 = (00101)
    bitset<5> bs2(5);

    // AND Operator
    cout << (bs1 & bs2) << endl;

    // OR Operator
    cout << (bs1 | bs2) << endl;

    // XOR operator
    cout << (bs1 ^ bs2);
    return 0;
}
```

‹bitset› Por que o Bitset é preferido

- **Memória eficiente – Armazena bits de forma compacta usando menos espaço do que matrizes de bools.**
- **Operações rápidas – Operações bit a bit como AND, OR, XOR são muito rápidas.**
- **Fácil de usar – Fornece funções integradas como count(), any(), none() e indexação with[].**

Biblioteca <bit>

Biblioteca <bit>

- A biblioteca **<bit>** é um conjunto de ferramentas e funções introduzido no C++20 para a manipulação de bits de forma padronizada e de alto desempenho. Ela oferece funções para operações bit-a-bit comuns, a maioria com tempo de complexidade O(1), pois geralmente são mapeadas para uma única instrução de hardware.
- As operações principais incluem:
  - Contar bits '1' em um inteiro (std::popcount).
  - Verificar se um número é potência de 2 (std::has_single_bit).
  - Contar zeros à esquerda ou à direita (std::countl_zero, std::countr_zero).
  - Encontrar a quantidade de bits necessária para representar um número
- **Cabeçalho necessário:**

Biblioteca <bit>

Operações

Nota: Todas as funções que operam sobre inteiros esperam tipos unsigned (como unsigned int, uint32_t, etc.) para garantir um comportamento bem definido e portável.

Principais Operações Usadas:

- **popcount(value)** - Conta o número de bits com valor 1 no inteiro. - **O(1)**
- **has_single_bit(value)** - Verifica se o inteiro é uma potência de dois (possui exatamente um bit 1). - **O(1)**
- **countl_zero(value)** - Conta o número de bits 0 consecutivos a partir da esquerda (do bit mais significativo). - **O(1)**
-

Biblioteca <bit>

Operações

Principais Operações Usadas:

- **countr_zero(value)** -Conta o número de bits 0 consecutivos a partir da direita (do bit menos significativo). - **O(1)**
- **bit_width(value)** - Retorna o número mínimo de bits necessários para representar o valor. - **O(1)**
- **bit_cast<ToType>(from_value)** - Reinterpreta os bits de um objeto (from_value) como se fossem de outro tipo (ToType), sem alterar os bits. Os dois tipos devem ter o mesmo tamanho - **O(1)**

Problema

2187

beecrowd | 2187

Bits Trocados

Por OBI - Olimpíada Brasileira de Informática 2000 Brazil

Timelimit: 1

As Ilhas Weblands formam um reino independente nos mares do Pacífico. Como é um reino recente, a sociedade é muito influenciada pela informática. A moeda oficial é o Bit; existem notas de B$ 50,00, B$10,00, B$5,00 e B$1,00. Você foi contratado(a) para ajudar na programação dos caixas automáticos de um grande banco das Ilhas Weblands.

Os caixas eletrônicos das Ilhas Weblands operam com todos os tipos de notas disponíveis, mantendo um estoque de cédulas para cada valor (B$ 50,00, B$10,00, B$5,00 e B$1,00). Os clientes do banco utilizam os caixas eletrônicos para efetuar retiradas de um certo número inteiro de Bits.

Sua tarefa é escrever um programa que, dado o valor de Bits desejado pelo cliente, determine o número de cada uma das notas necessário para totalizar esse valor, de modo a minimizar a quantidade de cédulas entregues. Por exemplo, se o cliente deseja retirar B$50,00, basta entregar uma única nota de cinquenta Bits. Se o cliente deseja retirar B$72,00, é necessário entregar uma nota de B$50,00, duas de B$10,00 e duas de B$1,00.

Entrada

A entrada é composta de vários conjuntos de teste. Cada conjunto de teste é composto por uma única linha, que contém um número inteiro positivo **V** ($0 \leq V \leq 10000$), que indica o valor solicitado pelo cliente. O final da entrada é indicado por **V** = 0.

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste **n**", onde **n** é numerado a partir de 1. Na segunda linha devem aparecer quatro inteiros **I**, **J**, **K** e **L** que representam o resultado encontrado pelo seu programa: **I** indica o número de cédulas de B$50,00, **J** indica o número de cédulas de B$10,00, **K** indica o número de cédulas de B$5,00 e **L** indica o número de cédulas de B$1,00. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Lista de Exercícios

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