Project Euler - Questão 16

Sérgio de Almeida Cipriano Júnior
Graduando em Engenharia de Software · Universidade de Brasília

Março 2020

1 Introdução

A ideia principal da questão é de realizar a exponenciação 2^{1000}. Considerando que o resultado dessa operação resulta em um número com mais de 300 dígitos, é inviável utilizar métodos tradicionais em linguagens como C ou $C++$, que não possuem aritmética extendida nas bibliotecas padrões.

Em $python\ 3$ essa questão é facilmente resolvida utilizando as ferramentas nativas da linguagem. Mesmo assim, uma alternativa interessante de resolução é com exponenciação rápida que, diferente da potenciação comum, desempenha em $O(log(n))$, $\forall\ a^n$.

Sendo assim, desenvolvi duas soluções para melhor explorar os conceitos apresentados pela questão.

2 Raciocínio

2.1 C++

Inicialmente defini todas as váriaveis que utilizaria até o final do algoritmo. A váriavel arr é o vetor que representará o número resultante da potenciação, sum é a soma de todos os dígitos do número e $power$ é a pôtencia.

Primeiro eu zero todo o vetor, para garantir que não tenha nenhum lixo de memória, e defino a posição inicial do vetor como 1, pois $2^0 = 1$. A ideia é realizar multiplicação de todos os termos por 2, 1000 vezes, e depois, para cada posição do vetor, retirar o ”excesso”.

Esse ”excesso” é basicamente o que ocorre numa multiplicação comum. Como estamos efetuando operações na base 10, cada dígito pode estar entre 0 e 9. Assim, para cada posição do vetor, mantém-se apenas o último digito e o restante é acrescentado na próxima casa. Para conseguir o último dígito tira-se o módulo por 10 e para conseguir os demais divide por 10.

Por fim, basta somar todas as posições do vetor arr que teremos a resposta. A complexidade resultando é $O(power * size)$.

2.2 Python 3

A resolução em python gira em torno de dois algoritmos: exponenciação rápida e soma de digitos. A soma segue a lógica das operações modulares por 10 explicada anteriormente.

O algoritmo de exponenciação rápida é bem mais robusto. Explicando bem superficialmente, interpretamos o expoente como um número binário e multiplicamos pela base sempre que o bit for 1. A base é elevada ao quadrado a cada loop, assim é como se ao invés de elevarmos pelo número n estivéssemos elevando por potências de dois.

Visualizando com um exemplo:

$$
\begin{aligned}
&5^{117} \\
&5^{1110101}, \quad \text{em binário} \\
&5^{2^0+2^2+2^4+2^5+2^6} \\
&5^1 * 5^4 * 5^{16} * 5^{32} * 5^{64}
\end{aligned}
$$

Assim, realizamos $log_2(117)$ iterações ao invés de 117.

3 Informações extras

Repositório com resoluções: *https://github.com/SergioAlmeidaCiprianoJr/UGED*