Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest, Ronald L. Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden

Apresentação Baseada:

- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Teorema Mestre

- O teorema mestre é uma ferramenta útil para avaliar algoritmos de divisão e conquista, que normalmente precisam de uma análise matemática mais complexa.
- Por exemplo os algoritmos de Karatsuba, de Contagem de Inversões e o Algoritmo de Strassen.

Problema da Multiplicação de Inteiros

Multiplicação de Inteiros

Dado dois inteiros x e y de n dígitos cada. Encontrar o produto $x \cdot y$.

Dividir $x = 10^{n/2}a + b$ e $y = 10^{n/2}c + d$, dessa forma:

$$xy = 10^n ac + 10^{n/2}(ad + bc) + bd$$

- Estratégia 1:
  - calcular recursivamente ac, ad, bc e bd
  - seja $T(n)$ o tempo de execução máximo para resolver um problema de tamanho n

$$\begin{aligned} \text{Para } n > 1: &\quad T(n) \leq 4T\left(\frac{n}{2}\right) + O(n) \\ \text{Caso base}: &\quad T(1) \leq O(1) \end{aligned}$$

- Estratégia 2 (Algoritmo de Karatsuba)
  - $xy = 10^n ac + 10^{n/2}(ad + bc) + bd$
  - calcular recursivamente ac, bd e $(a + b)(c + d)$
  - obter $ad + bc = (a + b)(c + d) - ac - bd$

$$\begin{aligned} \text{Para } n > 1: \quad & T(n) \le 3T\left(\frac{n}{2}\right) + O(n) \\ \text{Caso base}: \quad & T(1) \le O(1) \end{aligned}$$

O Teorema Mestre

- Pode ser usado como uma caixa preta para resolver recorrências.
- Só funciona quando todos os subproblemas tem o mesmo tamanho.

Suponha uma recorrência da seguinte forma:

$$\begin{aligned} \text{Caso base}: \quad & T(n) \le O(1) \text{ para } n \text{ suficientemente pequeno} \\ & T(n) \le aT\left(\frac{n}{b}\right) + O(n^d) \end{aligned}$$

$$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$$

$$T(n) = \begin{cases} O(n^d \log n) & \text{se } a = b^d \\ O(n^d) & \text{se } a < b^d \\ O(n^{\log_b a}) & \text{se } a > b^d \end{cases}$$

$$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$$

$$T(n) = \begin{cases} O(n^d \log n) & \text{se } a = b^d \\ O(n^d) & \text{se } a < b^d \\ O(n^{\log_b a}) & \text{se } a > b^d \end{cases}$$

Multiplicação de Inteiros com 4 chamadas recursivas:

- $T(n) \le 4T\left(\frac{n}{2}\right) + O(n)$
- Como $4 > 2$, caímos no caso 3. Portanto:
- $T(n) = O(n^{\log_2 4}) = O(n^2)$

Algoritmo de Karatsuba

- $T(n) \le 3T\left(\frac{n}{2}\right) + O(n)$
- Como $3 > 2$, também caímos no caso 3. Portanto:
- $T(n) = O(n^{\log_2 3}) \approx O(n^{1.58496})$

MergeSort:

- $T(n) \leq 2T\left(\frac{n}{2}\right) + O(n)$
- Como $2 = 2$, caímos no caso 1. Portanto:
- $T(n) = O(n^1 \log n) = O(n \log n)$

$$T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$$

$$T(n) = \begin{cases} O(n^d \log n) & \text{se } a = b^d \\ O(n^d) & \text{se } a < b^d \\ O(n^{\log_b a}) & \text{se } a > b^d \end{cases}$$

Algoritmo de Strassen

- $T(n) \leq 7T\left(\frac{n}{2}\right) + O(n^2)$
- Como $7 > 4$, também caímos no caso 3. Portanto:
- $T(n) = O(n^{\log_2 7}) \approx O(n^{2.81})$

Prova do Teorema Mestre

- Vamos considerar que n é uma potência de b.
- Ressalva: Essa prova não está 100% rigorosa, mas funciona para entender porque e como o teorema mestre funciona
- Usaremos a árvore de Recursão

$$T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$$

- Considere um nível j da árvore de recursão. Quanto trabalho é executado nesse nível?
- Número de subproblemas: a^j
- Tamanho de cada subproblema: $\frac{n}{b^j}$

Trabalho no nível $j \leq a^j O\left(\left(\frac{n}{b^j}\right)^d\right) \leq a^j \cdot c \cdot \frac{n^d}{b^{jd}} = c \cdot n^d \cdot \frac{a^j}{b^{jd}} = c \cdot n^d \cdot \left(\frac{a}{b^d}\right)^j$

Somando todos os níveis:

$$T(n) \leq c \cdot n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

$$T(n) \leq c \cdot n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

- a é a taxa de proliferação de subproblema
- b^d é a taxa de encolhimento do trabalho no subproblema
- Intuitivamente:
  - se $a = b^d$ o trabalho é igualmente distribuído por toda a árvore e portanto $O(n^d \log n)$
  - se $a < b^d$ o trabalho mais bruto está na raiz, portanto $O(n^d)$
  - se $a > b^d$ temos muitas folhas e portanto o trabalho principal está lá, Portanto $O(\text{número de folhas})$

$$T(n) \leq c \cdot n^d \cdot \underbrace{\sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j}_{S}$$

- $r = \frac{a}{b^d}$
- se $r = 1$, $S = \sum_{j=0}^{\log_b n} 1 = \log_b n$ logo

$$T(n) \leq c \cdot n^d \cdot \log_b n \text{ e ai é fácil mostra que } T(n) = O(n^d \log n)$$

- se $r < 1$, $S = \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j \leq 1 + r + r^2 + \ldots$ isso é uma Progressão geométrica de razão $r < 1$. Portanto:

$$T(n) \leq c \cdot n^d \cdot \frac{1}{1-r} \text{ e ai é fácil mostra que } T(n) = O(n^d)$$

$$T(n) \leq c \cdot n^d \cdot \underbrace{\sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j}_{S}$$

- Se r > 1:

$$S = \sum_{j=0}^{\log_b n} r^j = 1 + r + r^2 + \ldots + r^{\log_b n}$$

Isso é uma PG de razão r.

$$S_x = \frac{a_1(1 - r^x)}{1 - r} \qquad S = \frac{1 - r^{\log_b n}}{1 - r} = \frac{r^{\log_b n} - 1}{r - 1} \leq c' r^{\log_b n}$$

logo

$$T(n) \leq c \cdot n^d \cdot c' \cdot r^{\log_b n} = c \cdot n^d \cdot c' \cdot \frac{a^{\log_b n}}{b^{\log_b n.d}} = c \cdot n^d \cdot c' \cdot \frac{a^{\log_b n}}{n^d} = c \cdot \cancel{n^d} \cdot c' \cdot \frac{a^{\log_b n}}{\cancel{n^d}}$$

$$T(n) \leq cc' a^{\log_b n} = cc' n^{\log_b a} \text{ e ai é fácil mostra que } T(n) = O(n^{\log_b a}) \quad \square$$