

Projeto e Análise de Algoritmos I

Pedro Hokama

- [cls] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
 - [timr] Algorithms Illuminated Series, Tim Roughgarden
 - Desmistificando Algoritmos, Thomas H. Cormen.
 - Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
 - 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.

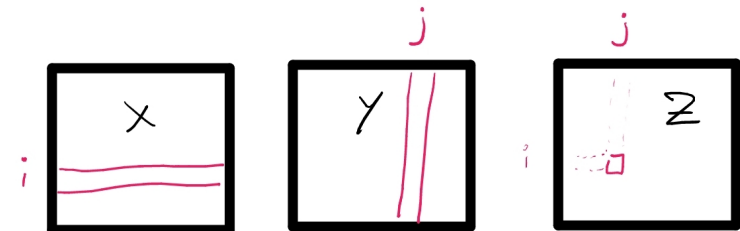
1 / 27

O Problema da Multiplicação de matrizes é extremamente importante pois é a essencial para aplicações em diversas áreas:

- Computação Gráfica
- Machine Learning
- Biologia Computacional
- Algoritmos Matemáticos e Físicos.
- e muitas outras.

3 / 27

2 / 27



Problema de Multiplicação de Matrizes

Sejam X e Y duas matrizes quadrada $n \times n$, desejamos obter a matriz $Z = X \cdot Y$ também $n \times n$, tal que

$$Z_{ij} = \sum_{k=1}^n X_{ik} \cdot Y_{kj}$$

4 / 27

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

Para obter cada número Z_{ij} precisamos multiplicar os n pares de elementos formados por X_{ik} e Y_{kj} para $k = 1 \dots n$. Qual a complexidade para se obter cada Z_{ij} ?

- a $\Theta(n)$
- b $\Theta(n \log n)$
- c $\Theta(n^2)$
- d $\Theta(n^3)$

E qual o tempo de execução total do algoritmo para obter todos os $O(n^2)$ valores de Z em função da largura das matrizes n ?

- a $\Theta(n \log n)$
- b $\Theta(n^2)$
- c $\Theta(n^3)$
- d $\Theta(n^4)$

5 / 27

6 / 27

Primeira Ideia

- Será possível fazer em tempo menor que cubico?
- Vamos tentar o paradigma de Divisão e Conquista!
- Identificar os passos de Divisão, Conquista e Combinação.
- Será que conseguimos dividir a matriz de alguma forma como no algoritmo de contagem de inversões.

Dividir cada matriz em quadrantes:

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \text{ e } Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

A multiplicação de matrizes nesse caso se comporta como se multiplicássemos elementos isolados.

$$Z = X.Y = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

7 / 27

8 / 27

$$XY = \begin{pmatrix} a_{11} & a_{12} & b_{11} & b_{12} \\ a_{21} & a_{22} & b_{21} & b_{22} \\ c_{11} & c_{12} & d_{11} & d_{12} \\ c_{21} & c_{22} & d_{21} & d_{22} \end{pmatrix} \begin{pmatrix} e_{11} & e_{12} & f_{11} & f_{12} \\ e_{21} & e_{22} & f_{21} & f_{22} \\ g_{11} & g_{12} & h_{11} & h_{12} \\ g_{21} & g_{22} & h_{21} & h_{22} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}e_{11} + a_{12}e_{21} + b_{11}g_{11} + b_{12}g_{21} & a_{11}e_{12} + a_{12}e_{22} + b_{11}g_{12} + b_{12}g_{22} & z_{13} & z_{14} \\ a_{21}e_{11} + a_{22}e_{21} + b_{21}g_{11} + b_{22}g_{21} & a_{21}e_{12} + a_{22}e_{22} + b_{21}g_{12} + b_{22}g_{22} & z_{23} & z_{24} \\ z_{31} & z_{32} & z_{33} & z_{34} \\ z_{41} & z_{42} & z_{43} & z_{44} \end{pmatrix}$$

De fato o primeiro quadrante do resultado é:

$$AE + BG$$

$$Z = XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

- Podemos criar um algoritmo recursivo que calcula os 8 subproblemas de maneira recursiva, cada um com matrizes de dimensão $n/2$ e depois combina com as somas que podem ser feitas em tempo $O(n^2)$.
- A notícia ruim é que esse algoritmo também é $O(n^3)$ igual ao direto. Pfft! 🙄
- Talvez aplicar algum *truque* como o de Gauss no algoritmo de Karatsuba?

9 / 27

10 / 27

Volker Strassen

- Volker Strassen, é um matemático alemão nascido em 1936.
- Professor emérito do Departamento de Matemática e Estatística da University of Konstanz.
- Recebeu diversos prêmios por suas contribuições em projeto e análise de algoritmos
- Em 1969 apresentou o primeiro algoritmo para fazer multiplicações de matrizes em tempo de execução inferior a $O(n^3)$.
- Causou um grande *frisson* na época, pois não acreditavam que tal multiplicação pudesse ter tempo subcúbico.



11 / 27

Algoritmo de Strassen - 1969

- Ideia: Reduzir o número de chamadas recursivas!
- Iremos computar apenas 7 chamadas recursivas.
- Faremos as adições e subtrações necessárias. O que ainda vai requisitar $O(n^2)$.

12 / 27

Os 7 produtos a serem computados são:

- $P_1 = A(F - H),$
- $P_2 = (A + B)H,$
- $P_3 = (C + D)E,$
- $P_4 = D(G - E),$
- $P_5 = (A + D)(E + H),$
- $P_6 = (B - D)(G + H)$ e
- $P_7 = (A - C)(E + F).$

$$X \cdot Y = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix} = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

$$\begin{aligned} &P_5 + P_4 - P_2 + P_6 \\ &= (A + D)(E + H) + D(G - E) - (A + B)H + (B - D)(G + H) \\ &= AE + AH + DE + DH + DG - DE - AH - BH + BG + BH - DG - DH \\ &= AE + \cancel{AH} + \cancel{DE} + \cancel{DH} + \cancel{DG} - \cancel{DE} - \cancel{AH} - \cancel{BH} + BG + \cancel{BH} - \cancel{DG} - \cancel{DH} \\ &= AE + BG \end{aligned}$$

13 / 27

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.

15 / 27

- Pelo teorema mestre a complexidade desse algoritmo é $O(n^{\log_2 7})$ (quando está no expoente a base do logaritmo importa sim) que é $\approx O(n^{2.8})$.
- Portando subcubico!
- As constantes no algoritmo de Strassen são bem maiores que as da multiplicação tradicional. Por isso só valem a pena para matrizes a partir de um certo tamanho. Para matrizes pequenas vale mais a pena a multiplicação tradicional.
- Minha sugestão para implementar o Algoritmo de Strassen é que na recursão, a partir de matrizes menores de 32 você use a multiplicação tradicional.
- Existem algoritmos teóricos mais eficientes como o de Coppersmith–Winogura, que é $O(n^{2.375})$, e de Williams *et al.* $O(n^{2.371552})$ mas não são usados na prática porque só seria melhor que o Strassen para matrizes tão grandes que não são possíveis de serem processadas.

14 / 27

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

$$\text{Para } n > 1 : \quad T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

$$\text{Caso base :} \quad T(1) \leq O(1)$$

16 / 27

- 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 $ad + bc = (a + b)(c + d) - ac - bd$

Para $n > 1$: $T(n) \leq 3T\left(\frac{n}{2}\right) + O(n)$

Caso base : $T(1) \leq O(1)$

- 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:

Caso base : $T(n) \leq O(1)$ para n suficientemente pequeno

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

17 / 27

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

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

18 / 27

Multiplicação de Inteiros com 4 chamadas recursivas:

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

Algoritmo de Karatsuba

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

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

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

19 / 27

20 / 27

Prova do Teorema Mestre

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:
- $T(n) = O(n^{\log_2 7}) \approx O(n^{2.81})$

21 / 27

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

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

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

$$= c \cdot n^d \cdot \left(\frac{a}{b^d}\right)^j$$

Somando todos os níveis:

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

23 / 27

- 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

22 / 27

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

- 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 $O(\text{número de folhas})$

24 / 27

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

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

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

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

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

25 / 27

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

- Se $r > 1$:

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

Isso é uma PG de razão r .

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

26 / 27

logo

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

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

$$T(n) = O(n^{\log_b a}) \quad \square$$

27 / 27