

# Paradigmas de Resolução de Problemas

Dividir e Conquistar – Definição

---

Prof. Edson Alves - UnB/FGA

2019

1. Dividir e Conquistar
2. Exemplos práticos de divisão e conquista

# **Dividir e Conquistar**

---

# Definição

- Dividir e conquistar (ou divisão e conquista) é um paradigma de solução de problemas que divide o problema em subproblemas menores sucessivas vezes até que o problema possa ser (trivialmente) resolvido
- Em seguida, ele combina as soluções dos subproblemas na solução do problema original
- Sistemáticamente, são três etapas:
  1. dividir o problema em subproblemas menores (divisão);
  2. resolver os subproblemas recursivamente, se ainda forem grande o suficiente, ou diretamente, se forem pequenos o bastante (conquista);
  3. combinar as soluções dos subproblemas para formar a solução do problema que foi dividido (fusão).

# Complexidade de soluções baseadas em divisão e conquista

- Dada a natureza do paradigma, algoritmos baseados em divisão e conquista são implementados por meio de recursão
- Esta característica facilita a implementação e deixa evidente as etapas do paradigma
- Contudo, a complexidade assintótica do algoritmo pode não ser óbvia
- Para computar a complexidade o primeiro passo é escrever a relação de recorrência que associa o número de operações  $T(n)$  necessárias para a resolução do problema com o número de operações necessárias para a solução dos subproblemas
- Uma vez estabelecida a recorrência, deve-se utilizar, se possível, o Teorema Mestre

# Teorema Mestre

## Teorema Mestre

Sejam  $a \geq 1$  e  $b > 1$  constantes,  $f(n)$  uma função tais que

$$T(n) = aT(\lfloor n/b \rfloor) + f(n)$$

Se

1.  $f(n) = O(n^{\log_b a - \varepsilon})$  para alguma constante  $\varepsilon > 0$ , então  $T(n) = \Theta(n^{\log_b a})$
2.  $f(n) = \Theta(n^{\log_b a})$  então  $T(n) = \Theta(n^{\log_b a} \log n)$
3.  $f(n) = O(n^{\log_b a + \varepsilon})$  para alguma constante  $\varepsilon > 0$ , e se  $af(n/b) \leq cf(n)$  para alguma constante  $c > 1$  e  $n$  grande o suficiente, então  $T(n) = \Theta(f(n))$

# Variáveis do Teorema Mestre

- Considere a relação de recorrência citada no Teorema Mestre:

$$T(n) = aT(\lfloor n/b \rfloor) + f(n),$$

- $T(n)$  é o número de operações para resolver um problema cuja entrada tem tamanho  $n$
- $a$  é o número de subproblemas após a divisão
- $\lfloor n/b \rfloor$  é o tamanho de cada subproblema
- Observe que cada subproblema deve ter o mesmo tamanho, e este tamanho deve ser um número inteiro
- $f(n)$  é o número de operações necessárias para dividir o problema e para combinar as soluções dos subproblemas para formar a solução do problema

# Interpretação do Teorema Mestre

- O Teorema compara a função que representa o custo  $f(n)$  das etapas de divisão e fusão com a função  $g(n) = n^{\log_a b}$ , onde o número de subproblemas é  $O(g(n))$
- No primeiro caso, o número de subproblemas é maior do que o custo da divisão e da fusão, e domina a complexidade
- No terceiro caso, o custo da divisão e da fusão é maior do que solução dos subproblemas, e domina a complexidade
- No segundo, ambas são equivalentes, de modo que o custo de divisão e fusão é multiplicado por um fator logarítmico, pois

$$\Theta(n^{\log_b a} \log n) = \Theta(f(n) \log n)$$

- Note que o Teorema Mestre não cobre todos os cenários possíveis



## **Exemplos práticos de divisão e conquista**

---

# Mergesort

- O *mergesort* é um algoritmo de ordenação que utiliza a divisão e conquista
- Na etapa de divisão, caso o vetor tenha  $N > 1$  elementos, e subdivide este vetor em duas partes de, aproximadamente, mesmo tamanho, e aplica o algoritmo em cada uma destas partes
- A etapa da conquista acontece quando  $N = 1$ : neste caso, o vetor estará trivialmente ordenado
- A fusão constitui a parte principal do algoritmo: uma vez que as duas partes foram ordenadas, elas devem ser combinadas para formar o vetor ordenado
- A ordenação das partes permite que esta fusão seja feita em  $O(N)$
- Assim a recorrência do *mergesort* é

$$T(n) = 2T(n/2) + O(n)$$

- Segundo o Teorema Mestre (caso 2), a complexidade será  $O(N \log N)$

# Implementação da fusão do *mergesort* em C++

```
5 template<typename RandIt>
6 void merge(size_t N, RandIt first, RandIt middle, RandIt last)
7 {
8     vector<typename iterator_traits<RandIt>::value_type> temp(N);
9     auto it = first, jt = middle;
10    auto k = 0;
11
12    while (it != middle and jt != last) {
13        temp[k++] = min(*it, *jt);
14        temp[k - 1] == *it ? ++it : ++jt;
15    }
16
17    while (it != middle)
18        temp[k++] = *it++;
19
20    while (jt != last)
21        temp[k++] = *jt++;
22
23    for (const auto& elem : temp)
24        *first++ = elem;
25 }
```

# Multiplicação matricial

- Sejam  $A$  e  $B$  duas matrizes  $N \times N$
- Considere, sem perda de generalidade, que  $N = 2^k$  para algum  $k$  não-negativo
- O produto  $C = AB$  é tal que

$$c_{ij} = \sum_{k=1}^N a_{ik} b_{kj}$$

- Como  $C$  possui  $N^2$  coeficientes e cada coeficiente é calculado em  $O(N)$ , a implementação da multiplicação matricial pela definição tem complexidade  $O(N^3)$

# Multiplicação matricial usando divisão e conquista

- A divisão e conquista pode ser utilizada para multiplicar matrizes
- Se  $N > 1$ , divida as matrizes  $A$  e  $B$  em quatro submatrizes de tamanho  $N/2 \times N/2$ :

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- Se  $N = 1$ , então  $C = AB$
- Caso contrário, as submatrizes de  $C$  são dadas por

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

# Multiplicação matricial usando divisão e conquista

- Assim, cada multiplicação matricial é feita mediante a combinação de 8 multiplicações de matrizes  $N/2 \times N/2$  por meio de somas
- Estas somas tem complexidade  $O(N^2)$
- A recorrência desta abordagem é

$$T(N) = 8T(N/2) + O(N^2)$$

- Portanto, a complexidade desta abordagem, segundo o caso 1 do Teorema Mestre, é  $O(N^3)$ , a mesma da implementação direta da definição

# Algoritmo de Strassen

- O algoritmo de Strassen reduz de 8 para 7 os subproblemas da abordagem anterior, por meio das somas adicionais
- Defina as somas

$$S_1 = B_{12} - B_{22}$$

$$S_6 = B_{11} + B_{22}$$

$$S_2 = A_{11} + A_{12}$$

$$S_7 = A_{12} - A_{22}$$

$$S_3 = A_{21} + A_{22}$$

$$S_8 = B_{21} + B_{22}$$

$$S_4 = B_{21} - B_{11}$$

$$S_9 = A_{11} - A_{21}$$

$$S_5 = A_{11} + A_{22}$$

$$S_{10} = B_{11} + B_{12}$$

- As matrizes  $S_i$  podem ser computadas em  $O(N^2)$ , e são utilizadas para computar 7 submatrizes  $P_j$ , que por sua vez serão combinadas por somas para determinar as submatrizes de  $C$

# Algoritmo de Strassen

- As matrizes  $P_j$  são dadas por

$$P_1 = A_{11}S_1$$

$$P_5 = S_5S_6$$

$$P_2 = S_2B_{22}$$

$$P_6 = S_7S_8$$

$$P_3 = S_3B_{11}$$

$$P_7 = S_9S_{10}$$

$$P_4 = A_{22}S_4$$

- Já as submatrizes de  $C$  são dadas por

$$C_{11} = P_4 + P_5 + P_6 - P_2$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 - P_1 - P_3 - P_7$$



# Algoritmo de Strassen

- A recorrência do algoritmo de Strassen é

$$T(N) = 7T(N/2) + O(N^2)$$

- De acordo com o primeiro caso do Teorema Mestre, a complexidade do algoritmo é  $O(N^{\log_2 7})$
- Observe que  $\log_2 7 \approx 2,81 < 3$
- Assim a complexidade do algoritmo de Strassen é menor do que a obtida pela implementação direta da multiplicação de matrizes

1. **CORMEN**, Thomas H.; **LEISERSON**, Charles E.; **RIVEST**, Ronald; **STEIN**, Clifford. *Introduction to Algorithms*, 3rd Edition, MIT Press, 2009.
2. **LAARKSONEN**, Antti. *Competitive Programmer's Handbook*, 2017.
3. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.