Projeto e Análise de Algoritmos (2025-1)

Flávio L. C. de Moura*

30 de abril de 2025

1 O algoritmo *mergesort*

Algoritmos recursivos desempenham um papel fundamental em Computação. O algoritmo de ordenação *mergesort* é um exemplo de algoritmo recursivo, que se caracteriza por dividir o problema original em subproblemas que, por sua vez, são resolvidos recursivamente. As soluções dos subproblema são então combinadas para gerar uma solução para o problema original. Este paradigma de projeto de algoritmo é conhecido com *divisão e conquista*. Este algoritmo foi inventado por J. von Newmann em 1945.

O algoritmo *mergesort* é um algoritmo de ordenação que utiliza a técnica de divisão e conquista, que consiste das seguintes etapas:

1. **Divisão**: O algoritmo divide a lista (ou vetor) l recebida como argumento ao meio, obtendo as listas l_1 e l_2;

2. **Conquista**: O algoritmo é aplicado recursivamente às listas l_1 e l_2 gerando, respectivamente, as listas ordenadas l_1' e l_2';

3. **Combinação**: O algoritmo combina as listas l_1' e l_2' através da função *merge* que então gera a saída do algoritmo.

Por exemplo, ao receber a lista $(4 :: 2 :: 1 :: 3 :: nil)$, este algoritmo inicialmente divide esta lista em duas sublistas, a saber $(4 :: 2 :: nil)$ e $(1 :: 3 :: nil)$. O algoritmo é aplicado recursivamente às duas sublistas para ordená-las, e ao final deste processo, teremos duas listas ordenadas $(2 :: 4 :: nil)$ e $(1 :: 3 :: nil)$. Estas listas são, então, combinadas para gerar a lista de saída $(1 :: 2 :: 3 :: 4 :: nil)$.

*flaviomoura@unb.br

Algorithm 1: mergesort(A, p, r)

```
1  if p < r then
2  |  q = ⌊(p+r)/2⌋;
3  |  mergesort(A, p, q);
4  |  mergesort(A, q + 1, r);
5  |  merge(A, p, q, r);
6  end
```

A etapa de combinar dois vetores ordenados (algoritmo *merge*) é a etapa principal do algoritmo *mergesort*. O procedimento $merge(A, p, q, r)$ descrito a seguir recebe como argumentos o vetor A, e os índices p, q e r tais que $p \leq q < r$. O procedimento assume que os subvetores $A[p..q]$ e $A[q+1..r]$ estão ordenados.

Algorithm 2: merge(A, p, q, r)

```
1  n1 = q − p + 1 ;                          // Qtd. de elementos em A[p..q]
2  n2 = r − q ;                              // Qtd. de elementos em A[q + 1..r]
3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays;
4  for i = 1 to n1 do
5  |  L[i] = A[p + i − 1];
6  end
7  for j = 1 to n2 do
8  |  R[j] = A[q + j];
9  end
10 L[n1 + 1] = ∞;
11 R[n2 + 1] = ∞;
12 i = 1;
13 j = 1;
14 for k = p to r do
15 |  if L[i] ≤ R[j] then
16 |  |  A[k] = L[i];
17 |  |  i = i + 1;
18 |  end
19 |  else
20 |  |  A[k] = R[j];
21 |  |  j = j + 1;
22 |  end
23 end
```

Exercício 1.1. *Prove que o algoritmo merge é correto.*

Exercício 1.2. *Prove que o algoritmo mergesort é correto.*

Exercício 1.3. *Faça a análise assintótica do algoritmo merge.*

Exercício 1.4. *Faça a análise assintótica do algoritmo mergesort.*

Equações de recorrência

Nesta seção estudaremos as equações de recorrência utilizadas no paradigma de divisão de conquista [2]:

Definição 1.5. *Seja $f(n)$ uma função não-negativa definida no conjunto dos números naturais. Dizemos que $f(n)$ é* eventualmente não-decrescente *se existir um número inteiro n_0 tal que $f(n)$ é não-decrescente no intervalo $[n_0, \infty)$, ou seja,*

$$f(n_1) \leq f(n_2), \forall n_2 > n_1 \geq n_0.$$

Definição 1.6. *Seja $f(n)$ uma função não-negativa definida no conjunto dos números naturais. Dizemos que $f(n)$ é* suave *se for eventualmente não-decrescente e*

$$f(2.n) = \Theta(f(n))$$

Teorema 1.7. *Sejam $f(n)$ uma função suave, e c e n_0 constantes positivas. Se $f(2n) \leq c.f(n), \forall n \geq n_0$ então $f(2^k n) \leq c^k.f(n), \forall n \geq n_0$ e $k \geq 1$.*

Teorema 1.8. *Seja $f(n)$ uma função suave. Então para qualquer $b \geq 2$ fixado,*

$$f(b.n) = \Theta(f(n))$$

O teorema a seguir é conhecido como *regra da suavização*

Teorema 1.9. *Seja $T(n)$ uma função eventualmente não-decrescente, e $f(n)$ uma função suave. Se $T(n) = \Theta(f(n))$ para valores de n que são potências de b ($b \geq 2$), então*

$$T(n) = \Theta(f(n)), \forall n.$$

A regra da suavização nos permite expandir a informação sobre a ordem de crescimento estabelecida para $T(n)$ de um subconjunto de valores (potências de b) para o domínio inteiro. O teorema a seguir é um resultado muito útil nesta direção conhecido como *teorema mestre*:

Teorema 1.10. *Seja $T(n)$ uma função eventualmente não-decrescente que satisfaz a recorrência*
$T(n) = a.T(n/b) + f(n), \quad$ *para* $n = b^k, k = 1, 2, 3, \ldots$
$T(1) = c$
onde $a \geq 1, b \geq 2$ *e* $c \geq 0$ *. Se* $f(n) = \Theta(n^d)$*, onde* $d \geq 0$*, então*

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & se\ a > b^d \\ \Theta(n^d . \lg n), & se\ a = b^d \\ \Theta(n^d), & se\ a < b^d \end{cases}$$

Demonstração. Considere que $f(n) = n^d$. Aplicando o método da substituição para a recorrência do teorema, obtemos:

$$T(b^k) = a^k.[T(1) + \sum_{j=1}^{k} f(b^j)/a^j]$$

Como $a^k = a^{\log_b n} = n^{\log_b a}$, podemos reescrever a equação acima como:

$$T(n) = n^{\log_b a}.[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j]$$

e para $f(n) = n^d$, temos:

$$T(n) = n^{\log_b a}.[T(1) + \sum_{j=1}^{\log_b n} (b^j)^d/a^j] = n^{\log_b a}.[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j]$$

A soma acima forma uma série geométrica, e portanto:

$$\sum_{j=1}^{\log_b n} (b^d/a)^j = (b^d/a)\frac{(b^d/a)^{\log_b n} - 1}{(b^d/a) - 1}, \text{ se } b^d \neq a.$$

Quando $b^d \neq a$, temos que $\sum_{j=1}^{\log_b n} (b^d/a)^j = \log_b n$. Agora basta analisarmos cada um dos casos: $a < b^d$, $a > b^d$ e $a = b^d$.

□

Apresentaremos agora uma versão um pouco mais geral do teorema mestre[1]. Consideraremos como anteriormente uma recorrência da forma:

$$T(n) = a.T(n/b) + f(n)$$

on $a \geq 1$ e $b > 1$ são constantes, e $f(n)$ é uma função assintoticamente positiva.

Teorema 1.11. *Sejam $a \geq 1$ e $b \geq 2$ constantes, $f(n)$ uma função assintoticamente positiva, e $T(n)$ definida nos inteiros não-negativos pela recorrência $T(n) = a.T(n/b) + f(n)$, onde n/b deve ser interpretado como $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$. Então $T(n)$ tem as seguintes cotas assintóticas:*

1. *Se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$;*

2. *Se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = \Theta(n^{\log_b a}.\lg n)$;*

3. *Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$, e se $a.f(n/b) \leq c.f(n)$ para alguma constante $c < 1$, então para todo n suficientemente grande, temos que $T(n) = \Theta(f(n))$.*

A prova será dividida em três lemas, onde inicialmente consideraremos que n é potência de b.

Lema 1.12. *Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função não-negativa definida para potências de b. Defina $T(n)$ para potências de b pela recorrência:*

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1; \\ a.T(n/b) + f(n), & \text{se } n = b^i \end{cases}$$

onde i é um inteiro positivo. Então

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j.f(n/b^j).$$

Demonstração. Analise a árvore de recorrência da equação dada.

□

Em termos da árvore de recorrência, os três casos do teorema mestre correspondem aos casos onde o custo total da árvore é:

1. dominado pelo custo das folhas;

2. uniformemente distribuído ao longo da árvore;

3. dominado pelo custo da raiz.

Lema 1.13. *Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função não-negativa definida para potências de b. A função $g(n)$ definida para potências de b por:*

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j . f(n/b^j).$$

tem as seguintes cotas assintóticas para potências de b:

1. *Se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $g(n) = O(n^{\log_b a})$;*

2. *Se $f(n) = \Theta(n^{\log_b a})$, então $g(n) = \Theta(n^{\log_b a} . \lg n)$;*

3. *Se $a.f(n/b) \leq c.f(n)$ para alguma constante $c < 1$ e para todo n suficientemente grande, então $g(n) = \Theta(f(n))$.*

Demonstração. Exercício.

□

Exercício 1.14. *Resolva as seguintes relações de recorrência:*

1. $T(1) = 1$, $T(n) = 3T(n/2) + n^2$, $n \geq 2$

2. $T(1) = 1$, $T(n) = 2T(n/2) + n$, $n \geq 2$

3. $T(1) \in \Theta(1)$, $T(n) = 3T(n/3 + 5) + n/2$

4. $T(1) = 1$, $T(n) = 2T(n-1) + 1$, $n \geq 2$

5. $T(1) \in \Theta(1)$, $T(n) = 9T(n/3) + n$

6. $T(1) \in \Theta(1)$, $T(n) = T(2n/3) + 1$

7. $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + 1$

8. $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + \sqrt{n}$

9. $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + \sqrt{n}\lg^2 n$

10. $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + n$

11. $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + n^2$

12. $T(1) \in \Theta(1)$, $T(n) = 3T(n/2) + nln(n)$

13. $T(1) \in \Theta(1)$, $T(n) = 3T(n/4) + nln(n)$

14. $T(1) \in \Theta(1)$, $T(n) = 2T(n/2) + nln(n)$

15. $T(1) \in \Theta(1)$, $T(n) = 2T(n/2) + n/ln(n)$

16. $T(1) \in \Theta(1)$, $T(n) = T(n-1) + 1/n$

17. $T(1) \in \Theta(1)$, $T(n) = T(n-1) + ln(n)$

18. $T(1) \in \Theta(1)$, $T(n) = \sqrt{n}T(\sqrt{n}) + n$

19. $T(n) = 8T(n/2) + \Theta(n^2)$

20. $T(n) = 8T(n/2) + \Theta(1)$

21. $T(n) = 7T(n/2) + \Theta(n^2)$

Referências

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 4 edition, April 2022.

[2] A. V. Levitin. *Introduction to the Design and Analysis of Algorithms, Third Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2012.