Algoritmo Merge-Sort

DANILA MARRIEL BAETA NASCIMENTO
GLAYSON PARAISO MACEDO
JHEAN MARCEL SOARES COSTA
JONATAS LEITE SOUZA
LEONARDO VIEIRA GUIMARÃES
MATHEUS SILVEIRA BORGES
NARA MIRANDA DE OLIVEIRA CANGUSSU

UNIMONTES

19 de Julho de 2017

Sumário

- A ordenação de dados é uma das funções mais utilizadas no desenvolvimento de programas de computador.
- Para tal, estão disponíveis na literatura inúmeros algoritmos de ordenação, com diferentes estratégias e complexidades.
- Encontrar o melhor entre eles depende de algumas condicionantes, como por exemplo o tamanho de entrada.

Cormen (2015) e Cormem *et al* (2002) relatam os seguintes Algoritmos de Ordenação:

- Inserção ou Insertion-Sort
- Seleção ou Selection-Sort
- Intercalação ou Merge-Sort
- Quick-Sort
- Bubble-Sort

Objetivos

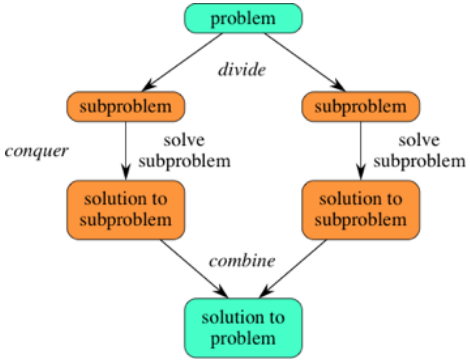
Este seminário tem como objetivo geral apresentar o algoritmo de intercalação Merge-Sort.

Também são objetivos específicos deste trabalho:

- Apresentar a estratégia Divisão e Conquista.
- Apresentar o funcionamento do algoritmo Merge-Sort pela ótica da recursividade;
- Comprovar analiticamente a complexidade deste algoritmo;

- O algoritmo Merge-Sort foi desenvolvido pelo matemático John Von Neumann em 1945;
- O algoritmo utiliza a estratégia "**dividir para conquistar**"como metodologia fundamental para solução do problema de ordenação;
- Na abordagem **divisão e conquista**, em um primeiro momento, o problema de desmembrado ao meio até não ser mais possível sua divisão.
- Em um segundo momento, as soluções dos subproblemas são mescladas entre si, gerando a solução do problema original.

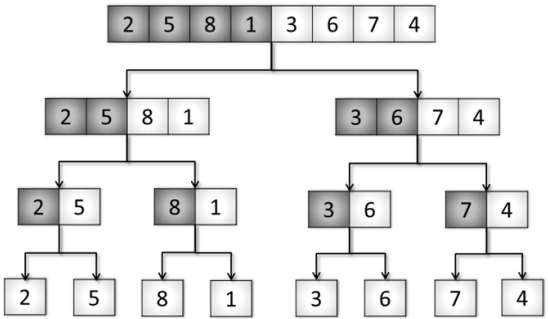
Figura: Estratégia de Divisão e Conquista.

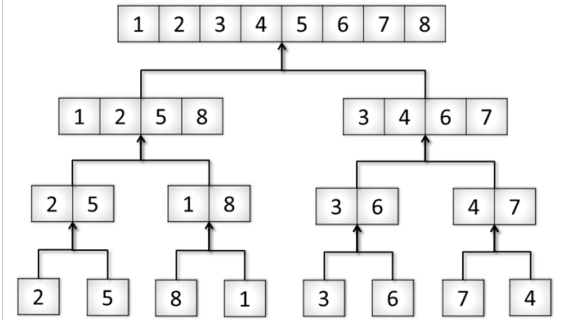


Fonte: Khan Academy.

- Segundo Cormem (2015), a divisão repetitiva do tamanho do subvetor ao meio dá origem a ideia de tempo de execução igual à O(lg n) para o Algoritmo Merge-Sort, o que iremos provar mais adiante.
- No caso do algoritmo de ordenação, na primeira estapa, os elementos a serem ordenados (nosso problema) são divididos até não ser mais possível.
- Em um segundo momento, os menores elementos de cada subvetores são comparados entre si, gerando um novo grupo já ordenado até. Isso ocorre até que todos os subvetores, ao fim, sejam unidos novamente em um só vetor, já ordenado.

Figura: Demonstração das duas etapas que constituem a estratégia de **Divisão e Conquista**.





Fonte: Stack Overflow. Adaptado pelos Autores.

O Algoritmo Merge-Sort

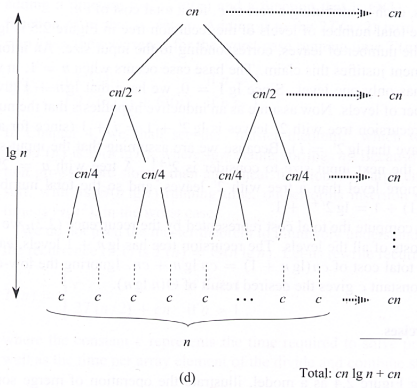
Pseudocódigo

Algorithm 1: MERGE-SORT(A, p, q, r)

```
1  n1 = q - p + 1
2  n2 = r - q
3  for i crescendo de p até q do
4  |  B[i] = A[i]
5  end
6  for j crescendo de q + 1 até r do
7  |  B[r+q+1-j] = A[j]
8  end
9  i = p
10 j = r
11 for k crescendo de p até r do
12 |  if B[i] ¡= B[j] then
13 |  |  A[k] = B[i]
14 |  |  i = i + 1
15 |  else
16 |  |  A[k] = B[j]
17 |  |  j = j - 1
18 |  end
19 end
```

- O funcionamento do Merge-Sort se dá pela metodologia recursiva, onde a função é chamada por ela mesma até que o critério de parada seja satisfeito.
- Neste método, todas as chamadas são feitas (ocupando espaços adicionais de memória) até que a resposta inicialmente solicitada seja finalmente apresentada.
- No nosso algoritmo, por exemplo, antes que seja apresentado o vetor ordenado, ele irá se subdividir e posteriormente se combinar de forma já ordenada $\lg(n)$ vezes.

Figura: Análise da árvore de recursividade.



Fonte: Cormem (2002)

A análise do consumo de tempo do algoritmo Merge-Sort pode ser feita da seguinte forma:

Algorithm 2: MERGE-SORT(A,p,r)

1	**if** $p < r$ **then**	1
2	q = [(p+r)/2]	1
3	MERGE-SORT (A, p, q)	T([n/2])
4	MERGE-SORT (A, q+1, r)	T([n/2])
5	MERGE-SORT (A, p, q, r)	an+c
6	**end**	

Com base na estratégia de **divisão e conquista**, pode-se chegar a seguinte equação de recorrência (CORMEN, 2015):

$$T(n) = T(n/2) + T(n/2) + n \tag{1}$$

Que podem ser representada apenas por:

$$T(n) = 2T(n/2) + n \tag{2}$$

Simulando a execução do algoritmo, podemos considerar o valor de n em um próximo passo como sendo $n/2$:

$$T(n/2) = 2T(n/2^2) + (n/2) \tag{3}$$

Com T(n/2) encontrado, o substituiremos na equação (2):

$$T(n) = 2[2T(n/2^2) + (n/2))] + n \tag{4}$$

Que pode ser representada apenas por:

$$T(n) = 2^2 T(n/2^2) + 2n \tag{5}$$

Agora, iremos aplicar *n/4* como sendo *n* na equação (2).

$$T(n/4) = 2T(n/2^3) + (n/4) \tag{6}$$

Agora que temos T(n/4), podemos aplicar na equação (5):

$$T(n) = 2[2^2 T((n/2^3 + (n/4)) + (n/2)))] + n \tag{7}$$

Que pode ser representada simplesmente por:

$$T(n) = 2^3 T(n/2^3) + 3n \tag{8}$$

Aqui, já se consegue encontrar um padrão de crescimento da função:

$$T(n) = 2^k T(n/2^k) + kn \tag{9}$$

Sabe-se também que quando a função atingir seu último passo de divisão, $(n/2^k)$ será igual a 1, menor valor que pode ser assumido, sendo este, inclusive, o critério de parada da recursividade.
Se igualarmos então $(n/2^k)$ por 1, temos que:

$$\frac{n}{2^k} = 1 \tag{10}$$

$$n = 2^k \tag{11}$$

Então teremos também que:

$$\lg n = k \tag{12}$$

Substituindo os valores encontrados em (11) e (12) em (9), temos que:

$$T(n) = nT(1) + n \lg n \tag{13}$$

Como sabemos que T(1) = 1, então:

$$T(n) = n + n \lg n \tag{14}$$

Que pode ser entendido apenas por:

O(n.lg n)

Podemos tratar o problema também com o Teorema Mestre, foco do trabalho anterior. Para tal, consideraremos que muitas das recorrências que ocorrem na análise de algoritmos de divisão e conquista têm a forma:

$$F(n) = aF\left(\frac{n}{b}\right) + cn^k \tag{15}$$

Theorem

Sejam $a \geq 1$, $b \geq 2$, $k \geq 0$ e $n_0 \geq 1$ números naturais e seja $c > 0$ um número real. Seja F é uma função que leva números naturais em números reais positivos e satisfaz a recorrência para $n = n_0 b^1$, $n_0 b^2$, $n_0 b^3$,

Theorem

Suponha que F é assintoticamente não decrescente, ou seja, que existe n_1 tal que F(n) e F(n+1) para todo $n \geq n_1$. Nessas condições:

- Se $\frac{\lg a}{\lg b} > k$ então F está em θ(nloga);
- Se $\frac{\lg a}{\lg b} = k$ então F está em θ(nklogn);
- Se $\frac{\lg a}{\lg b} < k$ então F está em θ(nk).

No caso do Merge-Sort, temos que:

$$F(n) = aF\left(\frac{n}{b}\right) + cn^k \tag{16}$$

$$= 2F\left(\frac{n}{2}\right) + n \tag{17}$$

Assim a $= 2$, b $= 2$ e k $= 1$.

Efetuando as operações de verificação do Teorema Mestre:

$$\lg(a) = \lg(b) = \lg(2) = 1 \tag{18}$$

Assim temos que:

$$\frac{\lg a}{\lg b} = k = 1 \tag{19}$$

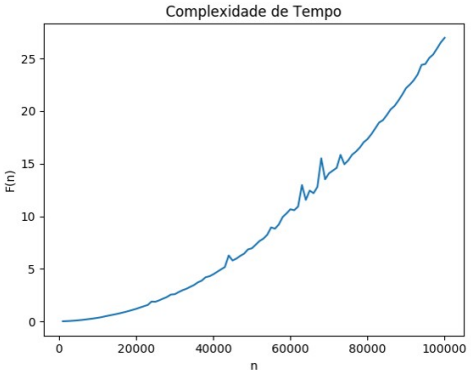
Portanto, como provado anteriormente, o algoritmo Merge-Sort pode ser entendido como: **O(n.lg n)**

Foi implementado o algoritmo Merge-Sort em Python 3.6 utilizando a IDE PyCharm Community. Para executar os testes de consumo de tempo, foi utilizado uma máquina com as seguintes configurações:

Tabela: Arquitetura de Máquina

Parâmetro	**Configuração**
Sistema Operacional	Windows 10 Pro
Processador	Intel Core i7-5500U CPU 2.40 GHz
Memória RAM	8 GB

Figura: Gráfico de tempo de execução por número de entradas.



Fonte: Os Autores

Ao fim, pode-se chegar as seguintes considerações sobre o algoritmo tratado neste seminário:

1. O algoritmo Merge-sort faz a ordenação ocupando uma grande quantidade de memória, pelo fato de usar a estratégia de recursividade.
2. Cormem (2014) instrui que, se a memória for um requisito de alto custo, este algoritmo não constitui uma boa escolha quando comparado com os demais algoritmos de ordenação (por inserção e seleção).
3. Uma outra desvantagem é o fator constante, que se apresenta maior que os dos demais algoritmos. Entretanto, para grandes entradas n, esse valor se torna desconsiderável.

Não obstante, pode-se listar as seguintes vantagens do algoritmo Merge-Sort:

1. O algoritmo Merge-Sort possui tempo de execução para o pior caso ($On \log n$) menor do que os demais algoritmos de ordenação (n^2), tornando-se uma excelente escolha quando o curto de memória não foi importante.
2. Portanto, o algoritmo é apropriado para listas de ordenação extensas.

Figura: Gráfico comparativo entre as funções $n^2, nlog_2$ e n.

Fonte: Khan Academy

Referências Bibliográficas

1. CORMEN, Thomas H. et al. **Algoritmos:** teoria e prática. Editora Campus, v. 2, 2002.
2. CORMEN, Thomas. **Desmistificando algoritmos.** Elsevier Brasil, 2015.