INTERCALAÇÃO E ORDENAÇÃO DE ARQUIVOS POR ALGORITMOS DE FUSÃO

Wagner Arbex*
Leonardo Evangelista Reis Izaias**
Fernando Vilarino**
Gabriella Castro Barbosa Costa**
Mayara Mendes Paiva**
Fernanda Rebelatto Miranda**
Fábio Lima de Siqueira**

RESUMO

Os algoritmos que utilizam técnicas de fusão (*merge*) de arquivos para a intercalação ou a ordenação de dois ou mais arquivos seqüenciais são conhecidos há muitos anos. Entretanto, as facilidades atualmente disponíveis para manipulação de bases de dados e arquivos como, por exemplo, a ampla e crescente utilização de sistemas gerenciadores de banco de dados, fazem com que esses algoritmos, em geral, rápidos e de fácil desenvolvimento e entendimento, sejam pouco considerados por profissionais e pouco estudados em cursos de graduação. Contudo, essas soluções de intercalação e ordenação são facilmente implementadas em ambientes de computação com poucos recursos e podem ser utilizadas em memória principal ou secundária, de modo interativo ou recursivo, gerando eficientes soluções que, praticamente, não dependem do aporte computacional do sistema hospedeiro.

Palavras-chave: Merge. Merge sort. K-way merge sort. Ordenação de arquivos. Fusão de arquivos.

ABSTRACT

Algorithms which use file merging techniques for either interleaving or sorting two or more files are known for a long time. Nevertheless, the resources and tools currently available for file handling, such as database management systems, cause these usually fast and easily-developing algorithms to be poorly studied by both professionals and undergraduate students. However, these solutions for file interleaving and sorting are easily implemented in poor computing environments and can be used, either recursively or interactively, in primary or secondary memories, creating efficient solutions which are almost independent of the host computer capacity.

Keywords: Merge. Merge sort. K-way merge sort. File sorting. File merging.

*Doutorando em Engenharia de Sistemas e Computação. Professor do Centro de Ensino Superior de Juiz de Fora e Analista da Empresa Brasileira de Pesquisa Agropecuária
**Graduandos em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora.

1 INTRODUÇÃO

A utilização de técnicas de fusão na manipulação de arquivos seqüenciais é antiga e obras clássicas da ciência computação, como o terceiro volume da **The Art of Programming Computer** (KNUTH, 1998), cuja primeira edição foi lançada 1973, já traziam esses algoritmos, bem como a análise deles. No Brasil, mais recentemente, destacam-se as obras Ziviani (2005) e Ziviani (2006) em que os algoritmos de fusão e outras técnicas de manipulação de arquivos foram abordados e analisados, visando à implementação dessas nas linguagens Pascal, C, C++ e Java.

As técnicas de fusão destacam-se pela facilidade de implementação de rotinas para manipulação de arquivos seqüenciais em memória principal ou secundária, permitindo o desenvolvimento de rotinas interativas ou recursivas, assim como pela flexibilidade, por não exigirem muitos recursos computacionais e pelo fato de serem facilmente implantadas em ambientes computacionais de diversos portes.

Atualmente, essas técnicas têm seu uso e estudo reduzidos, em parte, frente às facilidades de manipulação de arquivos que oferecem os sistemas gerenciadores de banco de dados e as linguagens de consulta. Realmente, essas facilidades devem ser utilizadas, entretanto, o conhecimento das técnicas de fusão não devem ser negligenciadas, visto, segundo Arbex (2008), a naturalidade com a qual são implementadas rotinas para, por exemplo, execução de *balance line*, ou para indexação ou geração de referências em sistemas de pequeno aporte computacional.

O objetivo deste texto é discutir alguns aspectos e características da implementação de três variantes de algoritmos de fusão para arquivos seqüenciais, especificamente, o *merge* interativo e recursivo, para execução em memória principal, o *merge sort*, para ordenação de dois arquivos e o *k-way merge sort*, para ordenação de múltiplos arquivos.

Para tanto, a organização deste trabalho se baseia em cinco seções e a primeira seção, que traz a presente "Introdução", especifica a motivação, a importância e o objetivo do estudo proposto e, ainda, delimita a abordagem e a estrutura do texto. Em seguida, a seção "Merge" aborda o algoritmo básico de fusão com intercalação, nas formas interativa e recursiva. A terceira seção, "Merge sort", apresenta uma rotina para ordenação de dois arquivos, baseada no algoritmo básico de fusão com intercalação, sendo esse mesmo assunto também tratado na seção seguinte, "K-way merge sort", cuja rotina de ordenação é ampliada para múltiplos arquivos. Por fim, a quinta seção, "Conclusão", discute o conhecimento adquirido, faz a análise final e apresenta uma sugestão para trabalhos futuros.

2 MERGE

O *merge*, também chamado de fusão com intercalação ou, simplesmente, intercalação, é baseado em algoritmos que seguem a técnica de divisão e conquista[1], em que uma instância do problema é sucessivamente dividida em várias instâncias, até que a solução para as instâncias resultantes seja trivial ou pelo menos o mais próximo disso, e, então, as soluções para essas instâncias são "combinadas", para que seja possível obter a solução do problema inicial.

Basicamente, a intercalação consiste em comparar dois elementos de uma estrutura de dados, seja uma tabela ou um arquivo e, dependendo do critério de comparação, por exemplo, *identificar o elemento de menor chave*, armazenar o elemento que guardará sempre o elemento que atenderá ao critério de comparação, identificando-o em uma terceira estrutura de dados. (ILUSTRAÇÃO 1).

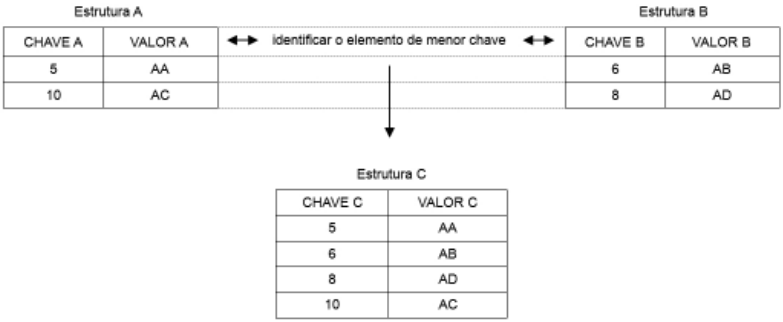

ILUSTRAÇÃO 1 – Comparação dos elementos de duas estruturas, gerando uma terceira estrutura com os elementos intercalados.

A ilustração 1 mostra o resultado final de uma intercalação entre as estruturas A e B, que resultou na estrutura C, entretanto, pela técnica de divisão e conquista, o procedimento completo passa pela divisão sucessiva das estruturas até que elas sejam reduzidas a n estruturas de um elemento cada uma. Em seguida, as estruturas são intercaladas duas a duas, e as estruturas resultantes dessas intercalações são sucessivamente intercaladas, originando a estrutura final.

[1]Inúmeros textos referenciam e apresentam trabalhos desenvolvidos sob a técnica de "divisão e conquista", aplicada pela primeira vez, em 1960, por Karatsuba Anatolii Alexeevich, quando desenvolveu o primeiro método e algoritmo, generalizado, para multiplicação de números em qualquer base. Seu algoritmo original pode ser visto na página "The Karatsuba Method 'Divide and Conquer'" (KARATSUBA, 2005).

3 MERGE SORT

Pelo exemplo apresentado na ilustração 1, assim como pela discussão do procedimento fundamental da intercalação ao fim da Seção 2, percebe-se facilmente que a intercalação pode ser utilizada para a ordenação de arquivos e, na verdade, o *merge sort* é o método de ordenação originado a partir dessa idéia.

O *merge sort* baseia-se, também, na técnica de divisão e conquista e seu fundamento consiste em criar uma estrutura ordenada a partir de duas outras também ordenadas. Para isso, parte-se do mesmo princípio de dividir a estrutura original tantas vezes quantas forem necessárias até que elas se tornem triviais e, então, são ordenadas e intercaladas sucessivas vezes, até que seja obtida a estrutura ordenada por completo.

O *merge sort* em sua forma interativa, *merge sort* interativo, resume-se a essa operação, pois consiste em intercalar duas estruturas disjuntas, previamente ordenadas, para produzir uma estrutura, cujo tamanho corresponde à soma das anteriores.

Em sua forma recursiva, o *merge sort* recursivo, o procedimento consiste em dividir a estrutura inicial em duas subestruturas, cada qual com metade do tamanho da estrutura original, considerando que tenha um número par de elementos, pois, caso contrário, obviamente, uma das subestruturas será maior do que a outra. Essa operação é repetida até que as estruturas resultantes sejam triviais.

A partir desse ponto, cada uma das subestruturas é ordenada recursivamente e, em seguida, são intercaladas, e todo o procedimento é repetido até que se obtenha uma única estrutura completamente ordenada.

3.1 MERGE SORT EXTERNO

Uma das grandes vantagens do *merge sort* é a possibilidade de ser implementado, de forma simples, quando os dados se encontram em memória secundária, e a memória principal não comporta o volume dos dados a ser tratado, sendo chamado de *merge sort* externo. Desse modo, o conjunto original dos dados deve ser subdividido, até que a memória principal comporte os subconjuntos gerados.

Como visto, o procedimento fundamental da intercalação passa por sucessivas divisões da estrutura original. Assim, é natural para o *merge sort* externo trabalhar com a estrutura em sucessivas divisões, até que as estruturas resultantes dessas divisões caibam na memória principal e, então, possam ser ordenadas.

Em seguida, é necessária a aplicação de um método auxiliar de ordenação, por exemplo, o *merge sort* interativo ou recursivo nas várias subestruturas e, em seguida, basta promover a intercalação das estruturas que couberam na memória

principal e foram ordenadas uma a uma. A estrutura final, após esse passo, estará ordenada.

De modo geral, os estudos desenvolvidos para elaboração desse texto mostram que, apesar do merge sort apresentar a complexidade O(n log n), segundo Knuth (1998) e Ziviani (2005), portanto, baixa, não pode ser considerado um método de ordenação rápido, principalmente quando implementado para memória secundária, visto que sua implementação é complementada de outro método.

A escolha do *quick sort*[2] para essa tarefa complementar de ordenação das partes da estrutura que couberam na memória principal é freqüente. Assim, apesar do *quick sort* também apresentar baixa complexidade, a execução de ambos aumenta o custo computacional. Contudo, o aspecto natural do *merge sort* de trabalhar com estruturas fracionadas e sua implementação fácil tornam-se seus atrativos.

A ilustração 2 mostra uma listagem, em linguagem Pascal, do procedimento básico de intercalação de dois arquivos, baseado em Vilarino e Reis (2008), considerando que ambos já foram ordenados. Para esse exemplo específico, os dois arquivos possuem o mesmo tamanho físico, o que não significa que, em uma operação de intercalação um deles não possa terminar de ser processado antes do outro. Para isso, basta que um dos arquivos possua a maioria elementos com as suas respectivas chaves com valores, em geral, menores do que as chaves dos elementos do segundo arquivo.

```
if (ponteiroA < filesize (A)) and (ponteiroB < filesize (B)) then
        if B.chave <= A.chave then begin
                { Se a chave B for menor ou igual à chave A, inclui o elemento de B na nova estrutura e avança a leitura em B }
                write (C, elementoB);
                inc (ponteiroA);
        end
        else begin
                { Se a chave A for menor do que a chave B, inclui o elemento de A na nova estrutura e avança a leitura em A }
                write (C, elementoA);
                inc (ponteiroB);
        end
else
        if ponteiroB >= filesize (B) then begin
                { Se B já estiver sido totalmente incluído, inclui o elemento de A e avança a leitura em A }
                write (C, elementoA);
                inc (ponteiroA);
        end
        else
                if ponteiroA >= filesize (A) then begin
                        { Se A já estiver sido totalmente incluído, inclui o elemento de B e avança a leitura em B }
                        write (C, elementoB);
                        inc (ponteiroB);
                end;
```

ILUSTRAÇÃO 2 – Procedimento básico de intercalação de dois arquivos, A e B, gerando o arquivo C e adotando como critério a inclusão em C do elemento de menor chave, entre A e B.

[2]Desenvolvido por Charles Antony Richard Hoare em 1960, sendo, posteriormente, publicado em Hoare (1962).

4 K-WAY MERGE SORT

O procedimento básico do *merge sort* externo (Seção 3) utiliza a estratégia de dividir a estrutura em blocos de entrada que caibam, o mais justo possível, na memória principal. Esses blocos são ordenados na memória principal e, posteriormente, são devolvidos para a memória secundária, gerando vários pequenos arquivos já ordenados. Em seguida, esses arquivos são intercalados em pares.

Assim, para o caso do *merge sort* externo, a implementação exemplificada na ilustração 2 mostra a intercalação de duas estruturas, sendo denominado de 2-*way merge sort* (KNUTH, 1998), que consiste em combinar duas seqüências ordenadas em uma única seqüência ordenada. Ou seja, o *merge sort* para apenas dois arquivos.

Entretanto, não é difícil estender esse conceito para o *k-way merge sort* (KNUTH, 1998), onde k significa a quantidade de blocos em que a estrutura de

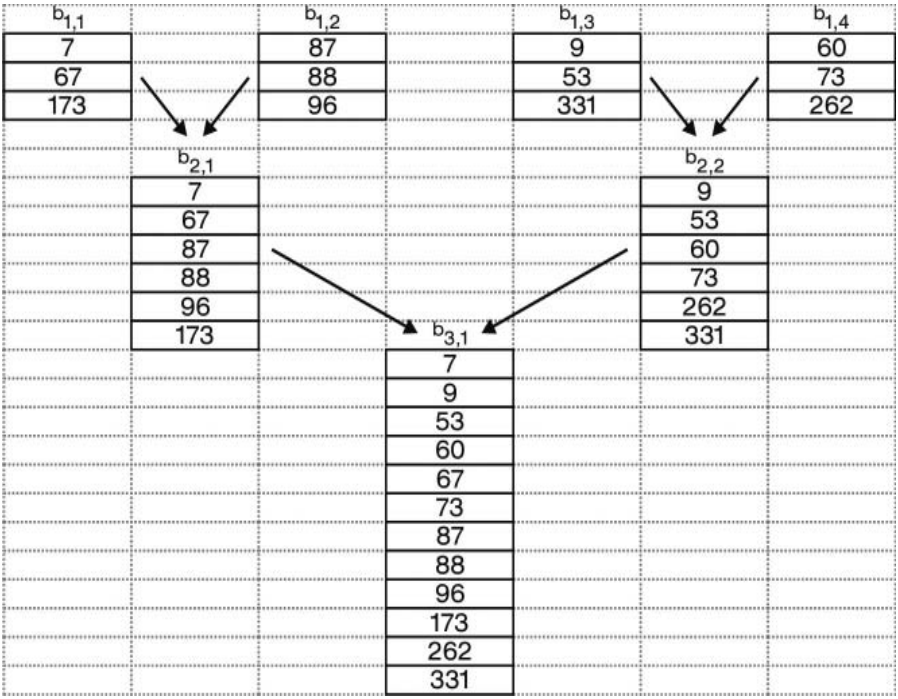

ILUSTRAÇÃO 3 – Procedimento de intercalação do *k-way merge sort* para quatro blocos.

entrada será dividida para, posteriormente, serem intercalados. A ilustração 3 mostra esse procedimento de intercalação para o *k-way merge sort*, supondo k = 4, isto é, foi necessária a divisão da estrutura original em quatro blocos $b_{1,j}$, $1 \leq j \leq k$, para que esses fossem ordenados na memória principal.

Esse procedimento pode ter sua implementação ainda mais simples, sacrificando a eficiência, caso um bloco seja sempre intercalado com o bloco resultante da intercalação dos seus dois blocos imediatamente anteriores, como pode ser visto na ilustração 4.

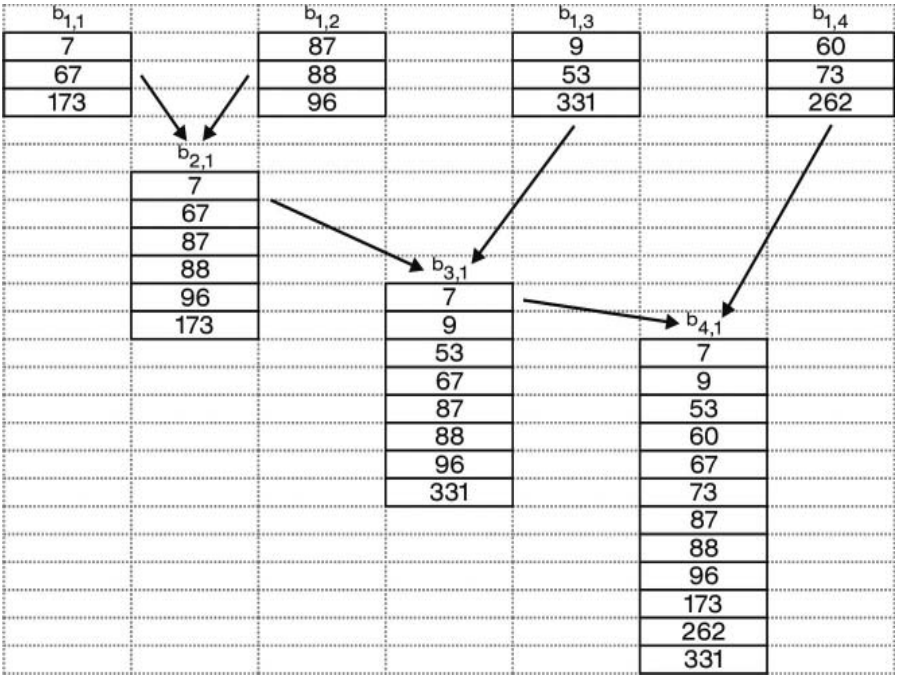

ILUSTRAÇÃO 4 – Outra opção de intercalação do *k-way merge sort* para quatro blocos.

4.1 ANÁLISE DO CUSTO COMPUTACIONAL DAS OPÇÕES PARA O PROCEDIMENTO DE INTERCALAÇÃO

A verificação da complexidade das duas propostas de intercalação do *k-way merge sort* (ILUSTRAÇÕES 3 e 4) mostra que ambas possuem complexidade linear, em

termos de *n*, isto é, em relação ao número de elementos de entrada nos procedimentos de intercalação. Essa análise pode ser verificada a partir da consideração que, em qualquer um dos procedimentos, a operação básica é a leitura das *n* entradas[3] de tamanho, sendo *n* número de elementos.

Para o procedimento ilustrado na ilustração 3, são feitas *kn* leituras para as entradas dos blocos $b_{1,j},\ 1 \le j \le \frac{k}{2^{1-1}}$, em seguida, para os blocos do nível 2, $b_{2,j},\ 1 \le j \le \frac{k}{2^{2-1}}$ são feitas $\frac{k}{2} 2n = kn$ leituras e, assim, sucessivamente. Dessa forma, determina-se um termo geral em que, para os blocos do nível i, $b_{i,j}$, sejam feitas $\frac{k}{2^{i-1}} 2^{i-1} n = kn$

O número total de leituras é obtido com o somatório das leituras nos blocos de todos os níveis, lembrando que o bloco do último nível não será lido, pois ele é o próprio resultado da intercalação. Dessa forma, para o procedimento representado pela ilustração 3, o "esforço" (E_1) é dado por:

$$E_1 = \left(kn + \frac{k}{2} 2n + \frac{k}{4} 4n + \ldots + \frac{k}{2^{(i-1)-1}} 2^{(i-1)-1} n \right) = (i-1)kn$$

Sabendo-se que i representa o número de níveis, k o número de blocos no início da intercalação, nível 1, e n o número de elementos em cada bloco do nível 1.

Se for considerado que cada bloco $b_{i,j}$ é um nó de uma árvore binária completa, como sugere a própria ilustração 3, o número de níveis pode ser relacionado com o número de nós, no caso, número de blocos. Assim, segundo Szwarcfiter e Markenzon $i = 1 + \lfloor \log k \rfloor$, portanto, $E_1 = kn \lfloor \log k \rfloor$ com $n \le 0$ e $k \le 1$ (1994),

Para o procedimento ilustrado na ilustração 4, são feitas 2n leituras para as entradas dos blocos $b_{1,1}$ e $b_{1,2}$ para que seja gerado o bloco $b_{2,1}$. Em seguida, para gerar o bloco $b_{3,1}$, são feitas 3n leituras; nos blocos $b_{1,3}$ e $b_{2,1}$, para gerar o bloco $b_{4,1}$, são feitas 4n leituras; nos blocos $b_{1,4}$ e $b_{3,1}$ e, seguindo esse raciocínio, chega-se ao termo geral, onde para cada bloco $b_{i,1}$, $i \ge 2$, são feitas *i.n* leituras.

Da mesma forma que o procedimento anterior, o número total de leituras desse procedimento é obtido com o somatório das leituras que foram feitas nos bloco de todos os níveis, para gerar os blocos predecessores. Assim, para o procedimento

$$E_2 = (2n + 3n + 4n + \ldots + i.n) = n(2 + 3 + 4 + \ldots + i) = n \sum_{i=2}^{k} i$$

Entretanto, esse somatório é equivalente à soma de uma progressão aritmética finita, de razão 1, com termo inicial 1 e k elementos, excluindo o primeiro elemento da progressão. Assim, $E_2 = n \sum_{i=2}^{k} i = n \left[(1+k) \frac{k}{2} - 1 \right] = \frac{n}{2} (k^2 + k - 2)$ com $n \le 0$ e $k \le 1$.

[3] A título de simplificação, considera-se que todos os k bloco possuem o mesmo número n de entradas.

representado pela ilustração 4, o esforço (E_2) é dado por:

Por E_1 e E_2, observa-se que o termo n, que representa a operação fundamental nos procedimentos das ilustrações 3 e 4 é de primeira ordem, o que sugere uma equivalência na complexidade de ambos. Porém, o termo k, igual para os dois casos, em E_2, influencia muito significativamente de forma a aumentar o custo computacional da solução exposta na ilustração 4. Isto é, a solução exemplificada na ilustração 4, apresenta pior desempenho no processamento, apesar da sua complexidade ter a mesma ordem de grandeza da solução mostrada na ilustração 3.

5 CONCLUSÃO

O estudo aqui desenvolvido discute como as técnicas de fusão podem ser úteis na solução de problemas comuns, como intercalação e ordenação de arquivos seqüenciais, devido a seus bons tempos de execução, tomando por base a complexidade de seus algoritmos, que são clássicas e podem ser encontradas em diversas obras tais como, Knuth (1998) e Ziviani (2005). Sobre a análise do custo computacional, apresentada na Seção 4, enfatiza-se o fato de que foi discutido somente o procedimento de intercalação, posterior à ordenação de cada um dos blocos de dados que couberam em memória.

Essa análise provou que o procedimento padrão de intercalação, opção 1, apresentado na lustração 3, sem dúvida, possui um desempenho melhor do que a variação apresentada na ilustração 4. Entretanto, ambos possuem custo de execução linear. A vantagem da opção 2 (ILUSTRAÇÃO 4), é a sua facilidade de implementação, mas o número de elementos a serem processados influencia de forma decisiva no seu desempenho. Ou seja, com poucos elementos, as duas opções podem ser executadas em um tempo aceitável, porém, com muitos elementos, a opção 2 tende a se tornar inviável.

A boa complexidade apresentada pelos algoritmos de fusão comprova a flexibilidade de sua implementação no tocante aos poucos recursos computacionais que exige, sendo esse fator um dos destaques positivos desses algoritmos. Entretanto, em suas restrições encontra-se o fato de somente tratar arquivos seqüenciais e, portanto, qualquer arquivo fora dessa especificação de acesso deverá sofrer um pré-processamento para adequá-lo especificidades desses algoritmos

O presente texto não abordou a variante *disk sort* dos algoritmos de fusão, que, em termos amplos, é a variante *k-way merge sort* para ordenação em memória secundária e, assim sendo, sugere-se a discussão do *disk sort* em trabalhos futuros, em complemento às variantes abordadas nesse texto.

Cabe ainda dizer que as três variantes tratadas nesse trabalho, bem como o *disk sort* são "hierarquicamente derivadas" entre si, no sentido de que o *disk sort*, utiliza o algoritmo *merge sort*, como parte de sua implementação e, ele por sua vez, utiliza a algoritmo fundamental do *merge* como parte de implementação.

Artigo recebido em: 08/09/2008
Aceito para publicação: 20/10/2008

REFERÊNCIAS

ARBEX, W. **Aula de Estrutura de Dados II.** Curso de Sistemas de Informação. Centro de Ensino Superior de Juiz de Fora. Juiz de Fora, 2008. (Notas de aula).

HOARE, C. A. R. Quicksort. **The Computer Journal**, v. 5, n. 1, p. 10–15, 1962.

KARATSUBA, E. **The Karatsuba method "divide and conquer"**. Sep. 14, 2005. Disponível em: <http://www.ccas.ru/personal/karatsuba/divcen.htm>. Acesso em: 8 set. 2008.

KNUTH, D. **The art of computer programming:** sorting and searching. 2. ed. Reading, Massachusetts: Addison-Wesley, 1998.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos.** 2. ed. LTC: Rio de Janeiro, 1994.

VILARINO, I.; REIS, L. **Fusão de arquivos utilizando merge sort externo.** Centro de Ensino Superior de Juiz de Fora. Juiz de Fora, 2008. (Atividade da disciplina Estrutura de Dados II).

ZIVIANI, N. **Projeto de algoritmos com implementações em Pascal e C.** 2. ed. São Paulo: Pioneira Thomson Learning, 2005.

ZIVIANI, N. Projeto de algoritmos com implementações em Java e C++. 2. ed. São Paulo: Pioneira Thomson Learning, 2006.