Tendências em Matemática Aplicada e Computacional, **2** (2001), 125-134.
Sociedade Brasileira de Matemática Aplicada e Computacional.

Cálculo da Complexidade Exata de Algoritmos do tipo Divisão-e-Conquista via Maple

A.B. LORETO[1], L.V. TOSCANI[2], Departamento de Informática Teórica, Instituto de Informática, UFRGS, Cx.P. 15064, 91501-900 Porto Alegre, RS, Brasil

M.P. FACHIN[3], Programa de Pós-Graduação em Matemática Aplicada (PPGMap), Instituto de Matemática, UFRGS, Cx.P. 15091, 91509-900, Porto Alegre, RS, Brasil

M.M. NEGRÓN[4], Departamento de Matemática, UNISC, Cx.P. 188, 96815-900, Santa Cruz do Sul, RS, Brasil.

Resumo. A equação de complexidade de um algoritmo recursivo pode ser expressa em termos de uma equação de recorrência. A partir destas equações obtém-se uma expressão assintótica para a complexidade, provada por indução. Neste trabalho, propõe-se um esquema de solução de equações de recorrência (lineares e do tipo divisão-e-conquista) resolvidas através do aplicativo matemático Maple, resultando em uma expressão algébrica exata para a complexidade. Desenvolveu-se um procedimento no Maple (bloco de comandos) para a solução destas equações de recorrência. O objetivo é obter uma forma geral de calcular a complexidade de um algoritmo desenvolvido pelo método Divisão-e-Conquista utilizando o aplicativo matemático Maple.

1. Introdução

A recursividade é uma técnica de resolver problemas que gera algoritmos com poucas linhas de código, mas que exigem no seu desenvolvimento um alto grau de concentração. Os algoritmos recursivos são portanto mais difíceis de serem analisados, pois possuem as informações mais concentradas e dentro de um raciocínio matemático mais elaborado, que é o princípio de indução. Sabe-se que o tempo de execução do algoritmo recursivo é penalizado em alguns compiladores, o que torna o cálculo de eficiência mais interessante na comparação de dois ou mais algoritmos. Entretanto, o cálculo da complexidade para algoritmos recursivos é muito mais complicado que para algoritmos não recursivos. Em geral, a equação de recorrência que define a complexidade do algoritmo é provada por indução, o que exige uma determinação de uma fórmula candidata da complexidade, a *priori* [12].

[1] loreto@inf.ufrgs.br
[2] laira@inf.ufrgs.br
[3] mpfachin@mat.ufrgs.br
[4] pepe@polaris.unisc.br

Muitas vezes, ótimos programadores e projetistas de algoritmos possuem certa aversão ao desenvolvimento de técnicas matemáticas de prova, o que lhes torna difícil o cálculo da complexidade de algoritmos recursivos. Realmente o tratamento usual desse cálculo é numa primeira fase bastante abstrata (descoberta da função candidata à complexidade) e numa segunda fase meio massante (prova por indução).

Uma metodologia de cálculo da complexidade se faz necessária, para tornar o estudo de complexidade uma atividade habitual. É importante que sejam desenvolvidas facilidades que tornem essa tarefa mais acessível. Considera-se que o ensino de algoritmos acompanhado de facilidades para o cálculo de complexidade é capaz de desenvolver um novo perfil de projetista de algoritmos, um projetista preocupado com a eficiência do algoritmo e compromissado com o cálculo de complexidade.

Este trabalho apresenta uma metodologia para o cálculo da complexidade de algoritmos recursivos, que dispensa a prova matemática e ao invés disso guia o projetista na formulação da equação característica de complexidade e a utilização do aplicativo de programação simbólica Maple para solucioná-lo.

2. Análise da Complexidade

A análise de um algoritmo tem como objetivo melhorar, se possível, seu desempenho e escolher dentre os algoritmos disponíveis o melhor. Existem vários critérios de avaliação de um algoritmo como: quantidade de trabalho requerido, quantidade de espaço (memória) necessário, simplicidade e exatidão da resposta [7].

O termo *Complexidade* refere-se, em geral, aos requerimentos de recursos necessários para que um algoritmo possa resolver um problema sob o ponto de vista computacional, ou seja, é a quantidade de trabalho despendido pelo algoritmo. A quantidade de trabalho requerido por um algoritmo não pode ser descrita simplesmente por um número, pois o número de operações básicas efetuadas, em geral não é o mesmo para qualquer entrada, mas depende do tamanho da entrada. Para medir a quantidade de trabalho despendido por um algoritmo, é escolhida uma operação fundamental e então é contado o número de chamadas dessa operação na execução do algoritmo. Neste trabalho a medida de complexidade considerada é a do Pior Caso.

Apesar da complexidade ser tratada como uma medida muito particular da classe do algoritmo que está sendo analisado, alguns aspectos do cálculo da complexidade não dependem do que faz o algoritmo mas somente de sua estrutura [13]. Dado um algoritmo a, a complexidade é definida pela função $T(a)$: $N \rightarrow N$, a qual, dado um tamanho de entrada n, leva ao número máximo de operações básicas necessárias para executar a, considerando-se todas as entradas de tamanho n. Normalmente, o algoritmo a está bem determinado, então usa-se somente a notação $T(n)$, para a complexidade do algoritmo a para uma entrada de tamanho n.

Um algoritmo recursivo, assim como os desenvolvidos por Divisão-e-Conquista, por conter em seu corpo uma chamada para si mesmo também terá, na equação de complexidade, uma chamada a si mesmo (função de complexidade), resultando numa equação de recorrência. Esta equação de recorrência é composta pela complexidade de uma chamada mais a complexidade resultante das chamadas recursi-

vas, colocadas como uma ocorrência de uma função T para um tamanho de entrada menor que o original.

A função $T(n)$ é escrita em função de $T(r(n))$; ou seja, um problema de tamanho n usa a solução de problemas menores de tamanho $r(n)$ $(r(n) < n)$, chamado tamanho do subproblema. Assim, tem-se a equação de recorrência geral:

$$T(n) = \begin{cases} b, & n = n_0 \\ f(n) + mT(r(n)), & n > n_0 \end{cases} \tag{2.1}$$

onde b é uma constante e é a complexidade da base da recursividade quando tem-se um elemento $(T(n \leq n_0))$, $f(n)$ é a complexidade de uma chamada recursiva podendo ser constante $(f(n) = b)$, exponencial $(f(n) = b^n)$ e polinomial $(f(n) = bn^s)$, m é o número de subproblemas em que o algoritmo foi decomposto ou o número de chamadas recursivas, $r(n)$ o tamanho do subproblema e $T(r(n))$ a complexidade resultante da chamada recursiva para resolver um problema de tamanho $r(n)$.
Em [8] foram estudados vários casos para $r(n)$, ou seja, vários tamanhos de subproblemas: $r(n) = n-1$, $r(n) = n-\varepsilon$ com $\varepsilon = 2$ e $\varepsilon = 3$, $r_1(n) = (n-1)+r_2(n) = (n-2)$ (dois subproblemas de tamanhos $n-1$ e $n-2$) e $r(n) = (n/c)$. Outro fator importante na complexidade é o número (m) de subproblemas, nesse caso a dicotomia se dá em $m = 1$ e $m > 1$. Este trabalho será aqui exemplificado em dois casos frequentemente encontrados na prática.

3. Equações Características para a Complexidade

A Complexidade de um algoritmo recursivo é expressa por uma equação de recorrência, apesar de apresentar-se diferentemente de um formato algébrico. A equação de complexidade usualmente apresenta-se como a equação (2.1) ao passo que no formato algébrico correspondente: $x_n - mx_{r(n)} = f(n)$. As equações características são comumente usadas para resolver equações em diferenças, escritas na forma de equações de recorrência, que podem ser homogêneas ou não-homogêneas.
Uma recorrência linear com coeficientes constantes é assim expressa

$$a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \cdots + a_kx_{n-k} = g(n). \tag{3.1}$$

Se $g(n)$ é igual a zero, a equação é dita homogênea e tem a forma

$$a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \cdots + a_kx_{n-k} = 0. \tag{3.2}$$

À forma homogênea corresponde uma equação característica, cuja solução resolve a recorrência. A equação característica associada à recorrência homogênea linear, assume o formato algébrico

$$a_0r^n + a_1r^{n-1} + a_2r^{n-2} + \cdots + a_kr^{n-k} = 0, \tag{3.3}$$

onde cada a_i é constante [4].
O polinômio correspondente à equação (3.3) é chamado Polinômio Característico e suas raízes chamadas Raízes Características. As raízes definem a solução da

recorrência homogênea. Conforme [4], [9] e [2] a equação (3.2) tem solução geral do tipo $x_n = r^n$ onde r é a raiz da equação característica (3.3). Se o polinômio característico possui mais de uma raiz, a solução geral tem a forma de uma combinação linear das raízes, e se as raízes possuírem multiplicidade maior que 1 a solução terá a forma de um polinômio em n com o grau igual ao número de raízes [10].

A maioria dos problemas resolvidos recursivamente, apresentam equações de recorrência, de forma não homogênea, precisando, inicialmente, serem homogeneizadas para então se obter uma equação característica. Lueker [9] foi o primeiro a usar equações características para resolver equações de recorrência, mais tarde tem-se Knuth [5], Brassard [2] e outros.

4. Aplicações do Método

Nesta seção serão apresentados dois exemplos, no primeiro ilustra o caso do problema que é dividido em subproblemas e então resolvido ($r(n) = n/c,\ m = c;\ c$ constante). No segundo caso o problema original, após uma decomposição, terá problemas cada vez menores, numa cadeia de passo constante ($r(n) = n - c,\ c$ constante).

4.1. Recorrência do tipo Divisão-e-Conquista

O método será ilustrado com o algoritmo *MaxMin* [7], que destina-se a encontrar o máximo e o mínimo entre os elementos de uma lista entre as posições i e j.
O algoritmo inicializa as variáveis *fmax* e *fmin*, a lista é repartida em duas listas de tamanhos aproximadamente igual a metade da original e o procedimento *MaxMin* é chamado para cada uma delas. Esse procedimento recursivo é efetuado até as listas posuirem tamanhos 1 ou 2.

Analisando o algoritmo verifica-se que dada entrada de um problema de tamanho n, este é partido em dois subproblemas de tamanho $n/2$ (isto é $m = 2$, r(n)=n/2), além disso é fácil ver que exceto pelas chamadas recursivas o algoritmo tem complexidade constante. Tem-se então uma equação de recorrência do tipo

$$T(n) = \begin{cases} b, & n = 1 \\ b + 2T(n/2), & n > 1. \end{cases} \tag{4.1}$$

Para resolver a equação de recorrência (4.1) realiza-se uma mudança de variável n por 2^i, originando uma nova recorrência em termos de t_i, definida por $t_i = T(2^i)$. Deste modo, a recorrência em $T(n)$ é definida como uma função de $T(n/2)$, fica assim

$$t_i = T(2^i) = 2T(2^{i-1}) + b, \tag{4.2}$$

reescrevendo em termos de t_i, tem-se

$$t_i - 2t_{i-1} = bt_1 = b, \tag{4.3}$$

onde t_1 é a condição inicial do algoritmo.

Agora é necessário homogeneizar a equação (4.3), isto é, zerar o lado direito da equação. Isto se faz reescrevendo a equação (4.3) para $i-1$ e somando as duas:

$$t_i - 3t_{i-1} + 2t_{i-2} = 0. \tag{4.4}$$

Como equação característica, associada a equação (4.4), tem-se

$$r^{i-2}(r^2 - 3r + 2) = 0,$$

cujas raízes não nulas são $r_1 = 2$ e $r_2 = 1$.

A partir deste momento, o aluno pode usar o Maple para finalizar o cálculo da complexidade do algoritmo. O restante do método será explicado por etapas, onde cada etapa corresponderá ao desenvolvimento teórico e o comando referente a este procedimento.

Inicialização: Inicia-se a sessão do Maple, criando uma Worksheet (nome relativo a cada sessão que se inicializa no aplicativo).

Entrada do Polinômio Característico: Utiliza-se o comando **factor e expand** para que o polinômio característico (r^2-3r+2) seja exibido no formato matemático, assim dá-se como entrada o seguinte comando:

>f:=expand(factor(r^ 2-3*r+2));

(";" indica que a resposta deve ser exibida na tela).

Cálculo das Raízes Características: Utiliza-se o comando **solve** e a variável **raiz**. A entrada do comando fica:

>raiz:=solve(f,r);

Solução geral em termos de $\mathbf{t}_i$: Com as raízes características constrói-se a solução geral da equação (4.2):

$$t_i = c_1 2^i + c_2 1^i. \tag{4.5}$$

No Maple entra-se com:

>t[i]:=c1*raiz[1]^ i + c2*raiz[2]^ i;

Solução Geral em termos de $T(n)$: Considerando que $T(2^i) = t_i$ e, em conseqüência,

$$T(n) = t_{\log_2 n}, \tag{4.6}$$

(pois $n = 2^i$ e $i = \log_2 n$, tem-se a solução geral da equação de recorrência (4.2):

$$T(n) = c_1 n + c_2. \tag{4.7}$$

No Maple, atribui-se à variável **i** o logaritmo de base 2 de n e à **T(n)** a solução geral, correspondendo aos seguintes comandos:

>i:=log[2](n);

>T(n):=c1*raiz[1]^ i + c2*raiz[2]^ i;

Mudança de Base: Caso seja necessário realizar uma mudança na base, por exemplo: $3^{\log_2 n} = n^{\log_2 3}$, no Maple corresponderá aos comandos

>ii:=log[2](3);

>T(n):=c1*n^ ii+c2*raiz[2]^ i;

Construção do Sistema de equações: Substituindo $n = 1$ e $n = 2$ na equação de recorrência (4.2) obtém-se $T(1) = b$ e $T(2) = 3b$, os termos independentes do

sistema. Substituindo $n = 1$ e $n = 2$ na equação (4.7) obtém-se $T(1) = c_1 + c_2$ e $T(2) = 2c_1 + c_2$, assim tem-se o sistema:

$$\begin{cases} T(1) = c_1 + c_2 = b \\ T(2) = 2c_1 + c_2 = 3b \end{cases}$$

que possui como solução $c_1 = 2b$ e $c_2 = -b$.

No Maple, primeiro deve-se chamar a biblioteca que contém os procedimentos para resolução de problemas de álgebra linear, o comando é:

>**with(linalg):**

(utiliza-se ":" para que não apareçam na tela todos os procedimentos)

E para construção do sistema linear usa-se uma variável **sys** que receberá o sistema de equações, do seguinte modo:

>**sys:=(c1+c2=b, 2*c1+c2=3*b);**

Solução do Sistema de equações: A resolução do sistema de equações ocorre através do comando **solve** em relação as constantes c_1 e c_2. Para melhor visualizar a solução do sistema de equações, atribuí-se à variável **sol1** os comandos **factor** e **simplify**, que, como os nomes sugerem, fatoram e simplificam a solução final:

>**sol:=solve(**{**sys**},{**c1,c2**}**):**

>**sol1:=factor(simplify(sol)):**

Solução Exata da Complexidade: Usa-se os comandos **simplify** (para simplificar a solução final), **subs** (que substitui a solução do sistema de equações em $T(n)$) e **power** (para apresentar a solução final em ordem crescente de expoente).

> **Tn:=simplify(subs(sol1,T(n)),power);**

O aplicativo retornará: $Tn = b(2n - 1)$ como solução da equação (4.2).

4.2. Recorrência Linear

Nesta seção apresenta-se recorrência linear de primeira ordem, conforme a classificação de [10]. Para este caso, o método será ilustrado com o algoritmo de Partição Recursivo, [8] que consiste em particionar os elementos de um vetor de tal forma que ao final todos os elementos maiores ou iguais a um certo elemento (*pivot*) estarão à direita dele enquanto os menores estarão à esquerda. Este exemplo é uma versão do algoritmo Partição devido a C.A. Hoare [6].

Este caso é interessante porque tem mais de uma operação fundamental, então a complexidade é obtida considerando as duas operações, com pesos diferentes. Chame p_1 o peso da operação de comparação e p_2 o peso da operação de troca. O pior caso ocorre quando todos os elementos são trocados, então a cada execução para um problema de tamanho n são realizadas duas comparações, uma troca e uma chamada recursiva para um problema de tamanho $r(n) = (n - 2)$. A equação de recorrência fica

$$T(n) = \begin{cases} 2p_1, & n \leq 1 \\ 2p_1 + p_2 + T(n-2), & n > 1 \end{cases} \tag{4.8}$$

onde $f(n) = 2p_1 + p_2$, $m = 1$ e $r(n) = (n - 2)$.

Para resolver a equação de recorrência (4.8), coloca-se esta equação no formato algébrico e define-se as condições iniciais:

$$x_0 = 2p_1, \quad x_1 = 2p_1, \quad x_n - x_{n-2} = 2p_1 + p_2. \tag{4.9}$$

Agora é necessário homogeneizar a equação (4.9), isto é, zerar o lado direito da equação. Isto se faz reescrevendo a equação (4.9) para $n-1$ e somando as duas:

$$x_n - x_{n-1} - x_{n-2} + x_{n-3} = 0. \tag{4.10}$$

Como equação característica, associada a equação (4.10), tem-se

$$r^{n-3}(r^3 - r^2 - r + 1) = 0,$$

cujas raízes, não nulas, são $r_1 = r_2 = 1$ e $r_3 = -1$.
Para finalizar o cálculo da complexidade do algoritmo utiliza-se o Maple.

Inicialização: Inicia-se a sessão do Maple, criando uma Worksheet.

Entrada do Polinômio Característico: Utiliza-se o comando **factor e expand** para que o polinômio característico $(r^3 - r^2 - r + 1)$ seja exibido no formato matemático, assim dá-se como entrada o seguinte comando:

>**f:=expand(factor(r^ 3-r^ 2-r+1));**

Cálculo das Raízes Características: Utiliza-se o comando **solve** e a variável **raiz**. A entrada do comando fica:

>**raiz:=solve(f,r);**

Solução geral em termos de $\mathbf{x}_n$: Com as raízes características constrói-se a solução geral da equação (4.9):

$$x_n = c_1 + c_2 n + c_3(-1)^n. \tag{4.11}$$

No Maple entra-se com:

>**x[n]:=c1*raiz[1]^ n+c2*raiz[2]^ n*n+c3*raiz[3]^ n;**

Construção do Sistema de equações: Substituindo $n = 0$, $n = 1$ e $n = 2$ na equação de recorrência (4.9) obtém-se $x_1 = 2p_1$, $x_2 = 2p_1$ e $x_3 = 4p_1 + p_2$, os termos independentes do sistema. Substituindo $n = 0$, $n = 1$ e $n = 2$ na equação (4.11) obtém-se $x_0 = c_1 + c_3$, $x_1 = c_1 + c_2 - c_3$ e $x_2 = c_1 + 2c_2 + c_3$, assim tem-se o sistema:

$$\begin{cases} x_0 = c_1 + c_3 = 2p_1 \\ x_1 = c_1 + c_2 - c_3 = 2p_1 \\ x_2 = c_1 + 2c_2 + c_3 = 4p_1 + p_2. \end{cases}$$

No Maple, primeiro deve-se chamar a biblioteca que contém os procedimentos para resolução de problemas de álgebra linear, o comando é:

>**with(linalg):**

E para construção do sistema linear usa-se uma variável **sys** que receberá o sistema de equações, do seguinte modo:

>**sys:=(c1+c3=2*p1,c1+c2-c3=2*p1,c1+2*c2+c3=4*p1+p2);**

Solução do Sistema de equações: A resolução do sistema de equações ocorre através do comando **solve** em relação as constantes c_1, c_2 e c_3. Para melhor visualizar a solução do sistema de equações, atribuí-se à variável **sol1** os comandos **factor** e **simplify**, que fatoram e simplificam a solução final:

>sol:=solve({sys},{c1,c2,c3}):

>sol1:=factor(simplify(sol)):

Solução Exata da Complexidade: Usa-se os comandos **simplify** (para simplificar a solução final), **subs** (que substitui a solução do sistema de equações em **x[n]**) e **power** (para apresentar a solução final em ordem crescente de expoente).

>T(n):=simplify(subs(sol1,xn),power);

O aplicativo retornará:

$$T(n) = n\left(p_1 + \frac{1}{2}p_2\right) + \frac{3}{2}p_1 - \frac{1}{4}p_2 + (-1)^n\left(\frac{1}{2}p_1 + \frac{1}{4}p_2\right) \tag{4.12}$$

como solução da equação (4.8),

$$(c_1 = \frac{6p_1 - p_2}{4};\ c_2 = \frac{4p_1 + 2p_2}{4};\ c_3 = \frac{2p_1 + p_2}{4}). \tag{4.13}$$

5. Conclusão

O cálculo da complexidade de algoritmos recursivos não é uma tarefa trivial. Da análise de um algoritmo recursivo à obtenção da equação de recorrência que descreve sua complexidade, é um procedimento mais ou menos direto, que não requer muito esforço, pois na realidade os dois tem o mesmo formato matemático: algoritmo recursivo, equação de recorrência. Essa etapa fica muito clara quando se trata de algoritmos desenvolvidos por Divisão-e-Conquista, como em [12]. O mais difícil é resolver a equação de recorrência. A solução da equação de recorrência através de equações características dispensa a prova por indução, tornando desnecessário o conhecimento a *priori* da solução da complexidade. Através deste método a equação de complexidade é resolvida diretamente pelo aplicativo matemático Maple, obtendo-se, em muitos casos, a solução exata, ou seja, o número exato de operações efetuadas pelo algoritmo, e não somente a ordem de complexidade. A solução não será exata quando a equação de recorrência não é exata.

Existem alguns algoritmos recursivos que não apresentam uma equação de recorrência direta e nesses casos equações características não são aplicados tão diretamente para sua solução, entretanto, grande parte dos algoritmos mais importantes podem ser resolvidos com expressões características, tanto para o caso médio como para o pior caso.

O principal resultado deste trabalho é a apresentação de um método para o cálculo da complexidade de algoritmos recursivos, que dispensa a prova matemática por indução e ainda utiliza um aplicativo de programação simbólica, que substitui o trabalho manual, para obter a complexidade exata do algoritmo recursivo analisado.

Outro ponto interessante a ser ressaltado é a utilização do aplicativo Maple para cálculo da complexidade, sendo que este foi desenvolvido, a princípio para resolução

de problemas simbólicos como: derivadas, integrais, solução de equações lineares, álgebra, funções trigonométricas, etc.

Lueker [9] utilizou equações características para resolver equações de recorrência, Brassard [2], usou equações características para calcular a ordem de complexidade dos tipos mais usuais de equações de recorrência (recorrência completa: com todos os termos) que definem complexidade no pior caso de algoritmos recursivos. Em [8] são desenvolvidos muitos casos (os mais representativos), com a solução exata da equação, evitando o cálculo massante da resolução do sistema de equações e substituição das constantes na solução final. Essa tarefa em casos mais complicados como por exemplo, na falta de algum termo na recorrência (caso não tratado por Brassard) pode ser muito trabalhoso e então a utilização de um aplicativo matemático como o Maple é quase indispensável. No caso de $r(n) = (n-3)$ por exemplo, [8] que dá origem a um sistema de cinco equações e um polinômio característico com duas raízes complexas e três raízes reais, a solução sem o uso de um aplicativo de programação simbólica seria impraticável.

Abstract. The complexity equation of a recursive algorithm can be expressed by a recurrence equation. From these equations, an asymptotic expression for complexity can be obtained and proved by induction. In this work, a scheme for the solution of recurrence equations (linear and divide-and-conquer type) is proposed solved using the mathematical package Maple, resulting in an exact algebraic expression for complexity. A procedure was developed in Maple (block of commands) to the solution of these recurrence equations. The objective is to obtain a general form to calculate the complexity of an algorithm developed using Divide-and-Conquer method using the mathematical package Maple.

Referências

[1] J.L. Bentley, D. Haken e J.B. Saxe, A general method for solving divide-and-conquer recurrence, *SIGACT News* **12** (1980), 36-44.

[2] G. Brassard e P. Bratley, "Fundamentals of Algorithmics", Prentice-Hall, 1995.

[3] T.H. Cormen, C.E. Leiserson e R.L. Rivest, "Introduction to Algorithms", Second edition, Massachusetts Institute of Technology, 1990.

[4] S. Goldberg, "Introduction to Difference Equations", Library of Congress Catalog card:58-10223, United States of America, 1958.

[5] D.H. Greene e D.E. Knuth, Mathematics for the Analysis of Algorithms, em "Progress in Computer Science", Vol. 1, Boston, Basel, Suttgart: Birkhasuser, 1981.

[6] C.A.R. Hoare, "Partition (Algorithm63)", CACM, Vol. 4, No. 7, 1961.

[7] E. Horowitz e S. Sahni, "Fundamentals of Computer Algorithms", Computer Science Press, 1978.

[8] A.B. Loreto, "Cálculo da Complexidade Exata de Algoritmos do tipo Divisão-e-Conquista através das Equações Características", Dissertação de Mestrado, (PPGMAp/UFRGS), 2000.

[9] G.S. Lueker, Some Techniques for Solving Recurrences, *Computing Surveys* **12** (1980).

[10] R. Sedgewick e P. Flajolet, "An introduction to analysis of algorithms", Addison-Wesley, 1996.

[11] L.V. Toscani, V. Laira e P.A.S. Veloso, Divisão e Conquista: Análise da complexidade, em "Seminário Integrado de Software e Hardware", Anais de SBC/UFPE, 13, pp. 89-104, Recife, Olinda, 1986.

[12] L.V. Toscani, "Métodos de Desenvolvimento de Algoritmos: Análise comparativa e de Complexidade", Tese de doutorado, Depto. de Informática, PUC/RJ, 1988.

[13] L.V. Toscani, V. Laira e D.S. Rosa, "Complexidade Computacional", Curso de Informática da Universidade Federal de Pelotas e Escola de Informática da Universidade Católica de Pelotas, 1997.