METODOLOGIA PARA O CÁLCULO DA COMPLEXIDADE DE ALGORITMOS E O PROCESSO DE AVALIAÇÃO DAS EQUAÇÕES DE COMPLEXIDADE

Marco Antonio de Castro Barbosa
UNICRUZ - Universidade de Cruz Alta
Departamento de Informática
Cx. Postal 858 – CEP 98.025-810 Cruz Alta (RS) – Tel.(55)3321-1541
marco@unicruz.edu.br

Laira Vieira Toscani
Leila Ribeiro
UFRGS – Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Teórica
Cx. Postal 15064 – CEP 91.501-970 Porto Alegre (RS) – Tel.(55)3316-6165
{laira,leila}@inf.ufrgs.br

Resumo: Este artigo apresenta um método de cálculo da complexidade de algoritmos para o pior caso. Tal método baseia-se em estruturas algorítmicas que são identificadas e expressas na forma de equações de complexidade. É apresentado também o processo de resolução dessas equações na análise da complexidade de algoritmos. Um exemplo de utilização da metodologia é o protótipo de sistema ANAC, que realiza a análise automática do cálculo da complexidade de algoritmos para o pior caso.

Palavras Chaves: Complexidade de algoritmos, análise de algoritmos, equações de complexidade.

Abstract: This paper show a methodology to calculate the complexity of algorithms to the worst case. This methodology is baseaded in the algorithms structures. This structures are identified and express in the way of complexity equations. The system ANAC is an example of this methodology.

Key Words: Complexity of algorithms, algorithms analisys, complexity equations.

1 Introdução

A análise da complexidade de um algoritmo é usualmente tratada de maneira muito particular, ou seja, é uma atividade muito dependente da classe dos algoritmos a serem analisados. O cálculo depende essencialmente da função tamanho da entrada e das operações fundamentais. No entanto, alguns aspectos do cálculo da complexidade dependem das estruturas que compõem o algoritmo, podendo ser assim generalizados. Esta idéia motivou o desenvolvimento de uma metodologia de cálculo de complexidade para estruturas algorítmicas para o pior caso [1]. Essa idéia serviu de motivação para o desenvolvimento do protótipo de sistema ANAC [2], que constitui-se em uma ferramenta para análise automática da complexidade de algoritmos.

Este artigo está dividido em 7 seções. A seção 2 apresenta alguns conceitos básicos de Complexidade de Algoritmos. Na seção 3 são vistos alguns sistemas para automatização da cálculo da complexidade de algoritmos. A seção 4 trata do método desenvolvido baseado nas estruturas algorítmicas. A seção 5 trata do protótipo de sistema ANAC para análise automática de algoritmos. Na seção 6 é apresentado o processo de resolução das equações de complexidade. Na seção 7 são apresentadas as conclusões.

2 Definições de Complexidade de Algoritmos

A análise de algoritmos é uma atividade que contribui para o entendimento fundamental da Ciência da Computação. Segundo Aho [3], a complexidade é o coração da computação. A construção de um algoritmo visa não apenas à solução de um determinado problema, mas à construção de um algoritmo bom, ou seja, que solucione o problema e seja eficiente.

A complexidade de algoritmos consiste na quantidade de trabalho necessária para a sua execução, expressa em função das operações fundamentais, as quais variam de acordo com o algoritmo, e em função do volume de dados. As operações fundamentais são aquelas que, dentre as demais operações que compõe o algoritmo, expressam a quantidade de trabalho, portanto extremamente dependentes do problema e do algoritmo.

De acordo com Knuth [4], um *algoritmo* é um método abstrato para computar a solução de um problema. Um *problema*, segundo Veloso & Veloso [5], é uma 3-upla p = <D,R,q>, onde D é o domínio das instâncias do problema, R é o domínio dos resultados e p é uma relação binária entre D e R que define o problema. Uma *solução* de p é uma função total $\alpha : D \rightarrow R$, tal que $(\forall d \in D)((d, \alpha(d)) \in q)$. Desta forma, dado um problema p = <D,R,q> e uma solução α para p, um algoritmo a_α é um método abstrato que computa α.

Dado um problema p = <D,R,q> com solução α considera-se a o conjunto de todos os algoritmos que computam α. Para calcular a complexidade de um algoritmo $a \in a$, deve-se determinar as operações fundamentais e definir a função tamanho do problema. Se houver mais de uma operação fundamental é necessário que se defina o peso de cada operação. Considere E o conjunto de todas as seqüências de execução das operações fundamentais. Obtém-se o seguinte diagrama:

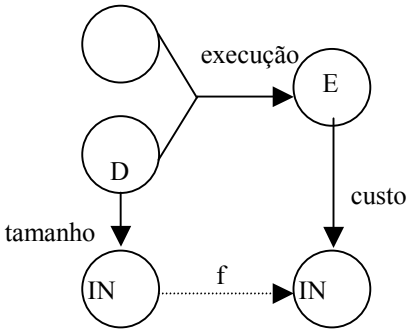


Figura 1 – Diagrama de Complexidade

onde:

- execução: x D $\rightarrow$ E; *execução*(a, d) := seqüência de execuções de operações fundamentais efetuadas na execução do algoritmo a, com entrada d.
- custo: E $\rightarrow$ IN; *custo*(s) := comprimento da seqüência s, definido conforme o peso estabelecido para as operações fundamentais.
- tamanho: D $\rightarrow$ IN; *tamanho*(d) := tamanho da entrada d.

Determinadas as funções fundamentais e a função *tamanho*, o conjunto é particionado (os algoritmos com mesmas funções fundamentais e mesma função *tamanho* ficam na mesma classe). A complexidade é então definida dentro de cada classe. Considere A como uma dessas classes e defina a complexidade de cada elemento de A, como uma função do tipo IN $\rightarrow$ IN. A *complexidade* é portanto um funcional do tipo A $\rightarrow$ (IN$\rightarrow$IN), isto é:

complexidade: A $\rightarrow$ *f* e *f* :={f/f: IN $\rightarrow$ IN}. Para a $\in$ A, *complexidade*(a) = f.

A função *tamanho* não é inversível. Dado n $\in$ IN, para avaliar f(n) é necessário considerar todas as entradas d $\in$ D com *tamanhod*(d):=n, é calculado *custo*(*execução*(a,d)) e então, conforme o critério de complexidade desejado é estabelecido f(n). Pode-se dizer que:

f(n) := *avaliação*({*custo*(*execução*(a,d))/d ∈ D e *tamanho*(d) = n}). E *custo*(*execução*(a,d)) pode ser denominado desempenho de d em a, ou simplesmente desempenho de d, quando está claro quem é a, i.é., *desempenho*(d) := *custo*(*execução*(a,d)).

Dados a ∈ e n ∈ IN, sejam

E_n := {d/d ∈ D e *tamanho*(d) = n} e prob(d) a probabilidade de d ocorrer, então:

avaliaçãoCPC(n) := max{*desempenho*(d) / d ∈ E_n}

define a Complexidade no Pior Caso, e:

$$avaliaçãoCM(n) := \sum_{d \in E_n} prob(b).desempenho(d)$$

define a Complexidade no Caso Médio.

A complexidade assintótica de tempo de computação de um algoritmo ignora todos os fatores que são dependentes da linguagem de programação e da máquina e concentra-se em determinar a ordem de magnitude da freqüência de execução das operações. O comportamento assintótico de um algoritmo é o mais procurado, já que para um volume grande de dados é que a complexidade torna-se mais importante. O algoritmo assintoticamente mais eficiente é melhor para todas entradas, exceto entradas relativamente pequenas. Várias medidas de complexidade assintótica podem ser definidas, a mais usada é a notação *0* .[6]

A notação *0:* Esta notação define um limite assintótico superior, isto é

$$f(n) \text{ é } 0(g(n)) \text{ sss } (\exists c)\ (\exists n_0)\ (\forall n \geq n_0)\ f(n) \leq c.g(n),$$

Isto é, a partir de um certo n_0, f(n) não cresce mais que c.g(n) ; ou f(n) tem o mesmo tipo de crescimento que g(n). Onde *n* é um parâmetro que caracteriza o volume de dados de entrada do algoritmo.

Se *f*(*n*) é *0*(*g*(*n*)), por abuso de linguagem também se escreve *f*(*n*) = *0*(*g*(*n*)). Assim o cálculo da complexidade se concentra em determinar a ordem de magnitude do número de operações fundamentais na execução do algoritmo em função do tamanho da entrada.

Uma das mais importantes medidas de complexidade de algoritmos é a medida de tempo. Isto se justifica em razão de boa parte da pesquisa em Ciência da Computação consistir do projeto e análise de algoritmos em relação à eficiência. Neste trabalho chamar-se-à de simplesmente complexidade à complexidade de tempo. Existem duas principais medidas de complexidade de tempo de um algoritmo: complexidade no Pior Caso e complexidade no Caso Médio. A complexidade no pior caso é geralmente a medida mais empregada na prática. Fixado um tamanho de entrada, a análise no pior caso é feita em relação ao número máximo de operações fundamentais necessárias para a resolução de qualquer problema do tamanho fixado. Seu valor pode ser considerado como um limite de complexidade que não será ultrapassado, sendo portanto, uma garantia de qualidade mínima do algoritmo. O aspecto negativo da análise no pior caso é que leva em consideração os casos caóticos que dificilmente irão ocorrer na prática.

3 Análise Automática de algoritmos

Nas últimas décadas muitos estudos foram realizados sobre a análise de algoritmos. Entretanto estes estudos focalizavam, em sua maioria, à análise de algoritmos específicos, tais como algoritmos de classificação, pesquisa e etc. Poucos estudos foram destinados à automatização do processo de análise de forma mais genérica.

Na década de 70 surgiram as primeiras tentativas de automatizar o processo de análise de algoritmos. Algumas destas tentativas resultaram em sistemas, como é o caso dos sistemas METRIC [7], ACE [8] e o Lambda-Upsilon-Omega[9].

O sistema METRIC, desenvolvido por Wegbreit foi o primeiro sistema criado com a finalidade de analisar algoritmos de forma totalmente mecânica. Este sistema transforma especificações de programas em equações de recorrência e ao resolver estas equações obtém a complexidade do programa analisado. O METRIC analisa programas segundo as medidas de desempenho apresentadas por Knuth [10], sendo elas a complexidade mínima (melhor caso), a complexidade

máxima (pior caso) e a complexidade média. Wegbreit estabeleceu uma análise mecânica de programas como uma atividade viável, o que foi inovador e motivou novas pesquisas neste campo.
O sistema ACE (*Automatic Complexity Evaluator*), desenvolvido por Le Métayer possui uma abordagem um tanto similar ao escopo de Wegbreit aplicada, no entanto, à análise no pior caso. O ACE analisa programas funcionais usando uma base de dados com regras que convertem as especificações de programas em equações de complexidade.
O sistema Lambda-Upsilon-Omega - ΛΥΩ, desenvolvido por Flajolet, Salvy e Zimmermann é um sistema projetado para realizar análise automática de classes restritas de procedimentos puramente funcionais. O sistema baseia-se na conjunção de duas idéias:
1) Recentes metodologias em análise combinatorial, que mostraram uma correspondência geral entre estruturas de dados e especificações de algoritmos e equações operando sobre funções geradoras. A computação automática destas equações de funções geradoras é realizada pelo Sistema Analisador Algébrico - ALAS.
2) As equações obtidas pelo ALAS são passadas para um Sistema Analítico – ANANAS, que é uma coleção de rotinas algébricas escritas na linguagem Maple, para serem resolvidas.

4 Complexidade das Estruturas Algorítmicas

Alguns aspectos do cálculo da complexidade de algoritmo dependem das estruturas que o compõem [1]. Serão consideradas a seguir as estruturas:
Atribuição: **a** := **b ;**
Seqüência: **a** ; **b** ;
Condicional: if **a** then **b** else **c** ;
Iteração: for **k** = **i** to **j** do **a** ;
Iteração Condicional: while **a** do **b** ;
A descrição das metodologias utiliza as seguintes notações:

c(a) - Complexidade de a no pior caso
(n) - Tamanho de entrada
$t^k(n)$ - Tamanho de entrada após k-ésima iteração

A complexidade de um algoritmo pode ser definida a partir da soma das complexidades de suas partes. A complexidade de uma parte pode ser absorvida pela de outra parte no seguinte caso:
A complexidade de **a** é "absorvida" pela complexidade de **b**, se e somente se

$\exists c\ \exists N\ \forall n \geq N$ complexidade(**a**)(n) $\leq$ c . complexidade(**b**)(n). Neste caso, diz-se que:
complexidade(**a**) + complexidade (**b**) = complexidade(**b**)
Por exemplo, considere complexidade(**a**) e complexidade(**b**) polinômios, n^2 e n^3 respectivamente. Neste caso a complexidade de menor grau é absorvida pela de maior grau. Desta forma, complexidade(**a**) é absorvida por complexidade(**b**), ou seja, n^2 é absorvida por n^3.
A seguir cada uma das estruturas algorítmicas citadas é analisada e é definida uma expressão para a complexidade.

4.1 Atribuição

A atribuição é apresentada na forma:

a := **b** ;

A complexidade associada a esta estrutura depende do tipo dos dados envolvidos.

c (**a** := **b**) = c(:=) + c (**b**)

A atribuição pode ser uma operação simples, como a atribuição de um valor a uma variável, ou uma atribuição mais complexa, como a inserção de um nodo num grafo, a atualização de uma matriz, dentre outros. A atribuição pode ainda requerer uma avaliação de **b**, que pode ser uma expressão ou uma função. Nestes casos, existe um esforço computacional associado a operação de atribuição c(:=) propriamente dita e um esforço associado à avaliação de **b**.

4.2 Seqüência

Este estrutura tem a forma:

a ; **b** ;

A complexidade da seqüência é a soma das complexidades componentes. A execução de **a**, entretanto pode alterar o volume de dados para **b**, então:

c(**a** ; **b**) (*n*) = c(**a**) (*n*) + c(**b**) (*t*(*n*))

onde, *t*(*n*) é o tamanho da entrada após a execução de **a**, dado que antes da execução era *n*.

4.3 Condicional

A estrutura condicional pode apresentar-se de diversas formas, sendo a mais usual:

if **a** then **b** else **c**

A complexidade desta estrutura é definida pela complexidade da avaliação da condição **a** mais a complexidade de **b** ou a complexidade de **c**, conforme o critério de complexidade a ser utilizado. Como esta se tratando de complexidade no pior caso, a complexidade é definida como a complexidade de **a** mais a complexidade máxima entre **b** e **c**. Ou seja;

c(if **a** then **b** else **c**) = c(**a**) + *max*(c(**b**),c(**c**))

Ocorre, porém, que *max* não tem uma interpretação padrão nesse caso, que c(**b**) e c(**c**) não são simples naturais, mas sim funções de IN em IN.

Freqüentemente, a partir de certo ponto, f(n) fica sempre maior do que g(n), ou vice-versa. Então, parece razoável tomar f ou g como *max*(f,g), conforme o caso. Entretanto, pode acontecer, como no caso abaixo, que não exista essa dominância.

Para exemplificar este situação, pode-se imaginar um algoritmo que manipula grafos e efetua duas operações, uma cuja complexidade depende exclusivamente do número de arestas e outra cuja complexidade varia exclusivamente com o número de nodos. A função da entrada do algoritmo tem duas componentes: número de arestas e número de nodos, combinados de alguma forma (somados, por exemplo). O algoritmo constitui-se de um condicional cujo ramo **then** efetua uma das operações, por exemplo aquela dependente do número de arestas, e o ramo **else** efetua a outra operação. Aumentando o número de arestas, aumenta somente a complexidade do ramo **then** e aumentando o número de nodos, somente aumenta a complexidade do ramo **else**. Desta forma, aumentando convenientemente a entrada, a complexidade de cada ramo pode superar a do outro. Nesse ponto, o máximo ponto a ponto pode ser usado, i.e., a função:

max(f,g)(n) := max(f(n),g(n)).

Uma solução simplista, mas muitas vezes usada, é utilizar como máximo a soma ponto a ponto das duas funções: max (f,g) = f + g, com

(f+g)(n) := f(n) + g(n).

Na verdade, há várias possíveis escolhas para máximo assintótico de funções de naturais. Para definir o máximo entre funções, é preciso ter uma relação de ordem entre elas. Em [6] há uma discussão bastante completa sobre o assunto.

A estrutura condicional também pode apresentar-se num modo mais simples sem a presença do **else**:

if **a** then **b**

neste caso, a complexidade desta estrutura é simplificada:

c(if **a** then **b**) = c(**a**) + c(**b**)

4.4 Iteração Não Condicional

O caso mais simples de iteração não condicional (ou definida) é:

for **k** = **i** to **j** do **a**

A execução da iteração causa a execução de **a** (j-i+1) vezes, com o valor de **k** variando de **i** até **j**. Considerando-se que as valores de **i** e **j** não são alterados na execução de **a**, o número de iterações é determinado e é (j-i+1). Pode ocorrer, entretanto, a situação onde a complexidade de execução de **a** varia a cada iteração, por exemplo, alterando o tamanho da entrada, então tem que ser considerada a complexidade de cada iteração executada. Por estas razões, a complexidade desta estrutura tem dois casos a serem considerados:

Se a complexidade de execução de **a** não varia durante a iteração:

c(for **k** = **i** to **j** do **a**) = (**j**-**i**+1).c(**a**)

Se a complexidade da execução de **a** varia durante a iteração, tem-se que:

$$c(\text{for } \mathbf{k} = \mathbf{i} \text{ to } \mathbf{j} \text{ do } \mathbf{a})(n) = \sum_{k=0}^{j-i} c(\boldsymbol{a})(t^k(n))$$

onde $t^k(n) = t(t^{k-1}(n))$ se $k \geq 1$ e $t^0(n) = n$ e $t^{k+1}(n) = t^k(t(n))$, (n) conforme definido em 4.2.

4.5 Iteração Condicional (ou Indefinida)

As estruturas de iteração condicional (ou indefinida), podem assumir várias formas. A forma vista a seguir será o **while**, mas o tratamento para as demais estruturas é similar.

while **a** do **b**

Neste tipo de iteração **b** será executado sucessivamente enquanto a condição **a** for satisfeita, possivelmente com alteração no volume dos dados.

$$c(\text{while } \mathbf{a} \text{ do } \mathbf{b})(n) = c(\boldsymbol{a})(t^k(n)) + \sum_{i=0}^{k-1} (c(\boldsymbol{a})c(\boldsymbol{b}))(t^i(n))$$

5 O Analisador de Complexidade – ANAC

A construção de algoritmos que, não apenas solucione determinado problema, mas o faça em tempo viável, constituíram-se em motivação para a construção de um sistema que realiza a análise algorítmica de forma semi-automática, o Analisador de Complexidade - ANAC. [2]

O sistema ANAC é uma ferramenta de apoio ao ensino de complexidade de algoritmos, constituindo-se num ambiente de aplicação prática ao embasamento teórico adquirido no ensino de complexidade algorítmica, que tem como principal objetivo estimular o cálculo da complexidade de algoritmos.

O protótipo de sistema ANAC baseia-se na metodologia de cálculo de complexidade para estruturas algorítmicas proposta em [1]. O sistema analisa algoritmos não recursivos escritos em uma linguagem Pascal-like pré-estabelecida para o sistema. Essa linguagem engloba as principais estruturas presentes em linguagens imperativas: atribuição, seqüência, condicional, iteração e iteração condicional, calculando de forma semi-automática a equação de complexidade e resolvendo-a sempre que forem fornecidos os dados necessários para este processo.

O ANAC consiste de um processo de análise-síntese que recebe como entrada uma especificação algorítmica escrita numa linguagem Pascal-like pré-estabelecida e à medida que a análise sintática vai sendo efetuada a equação de complexidade vai sendo construída. O processo de análise sintática ocorre de maneira *top-down* e ao ser identificada uma estrutura a sua equação de complexidade correspondente é construída como a expressão de uma árvore e são geradas variáveis de complexidade para representar a complexidade das partes ainda não identificadas. Quando o sistema encontra uma estrutura que não apresenta mais ramificações, ou seja, é uma estrutura que representa um nodo folha, é feito o cálculo da complexidade da estrutura corresponde e o seu valor é atribuído à variável correspondente a aquela estrutura. Nesse ponto o processo sobe na árvore resolvendo as complexidades estabelecidas naquele ramo, que eram dependentes do valor calculado na folha. Quando todos os ramos são completados é atingido a raiz e o cálculo de complexidade do algoritmo é finalizado. Essa complexidade pode, entretanto, não ser uma expressão simples, o sistema irá simplificar a expressão sempre que possível. Tal processo é melhor detalhado na seção 6.

6 O Processo de Resolução das Equações de Complexidade no sistema ANAC

O sistema ANAC implementa a metodologia apresentada na seção 4, através de uma análise simbólica *top-down*. Quando uma estrutura algorítmica é identificada a equação de complexidade referente a ela é definida. As complexidades das partes são identificadas por variáveis que são detalhadas (calculadas por outras equações de complexidade) à medida que a estrutura algorítmica correspondente é identificada.

À medida que o algoritmo vai sendo analisado e suas estruturas são identificadas é construída uma árvore-sintática correspondente. Nesta árvore a raiz é a complexidade do algoritmo, que é constituído de partes que podem ser: atribuições, estruturas condicionais, iterações definidas ou indefinidas, ou seqüências. Uma estrutura por sua vez é também composta por partes cada uma gerando um ou mais filhos.

A seguir será detalhado este processo.
A operação fundamental é a atribuição associada ao valor 1. A complexidade dos testes é requerida ao usuário e aos delimitadores de escopo é atribuído o valor zero.

Notação:

- S_i é uma estrutura do algoritmo.
- x_i = complexidade de uma estrutura algorítmica S_i;
- t(n) = tamanho da entrada

A seqüência é considerada a estrutura básica. O algoritmo é considerado sempre, em princípio, como uma seqüência. É identificada a primeira estrutura e a continuação é atribuída a uma variável.

Seqüência: c(**a**;**b**)(n) = c(**a**) (n) + c(**b**) (t(n))

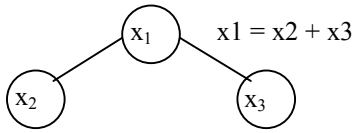
Estrutura	Análise do programa	Árvore da Seqüência
S_1 ; S_2 ;	x1 = x2 + x3	

Tabela 1 – Seqüência

O cálculo da complexidade de uma seqüência, constitui-se na soma das complexidades das partes. A Tabela 1, ilustra uma representação da complexidade da seqüência, onde x1 representa a soma das partes S_1, S_2, cujas complexidades estarão definidas em x_2 e x_3.

Condicional: if <*condição*> then S_1 else S_2
Considerando x1 = c(if <*condição*> then S_1 else S_2) ou seja, x1 é igual a complexidade da estrutura: (if <*condição*> then S_1 else S_2).
Para se obter a complexidade desta estrutura as complexidades da condição e das estruturas S_1 e S_2 devem ser avaliadas. Serão utilizadas as variáveis x2 e x3 para representar as complexidades de S2 e S3, respectivamente. Para exemplificar esse processo a Tabela 2 mostra a estrutura condicional, a análise da complexidade desse programa no pior caso, pelo sistema ANAC e a ilustração da árvore criada para a estrutura condicional.
O sistema ANAC, no cálculo da complexidade da estrutura condicional usa a seguinte equação:

c(if <*condição*> then S_1 else S_2) = c(*condição*) + c(S_1) + c(S_2)

Programa	Análise do programa	Árvore da Estrutura Condicional

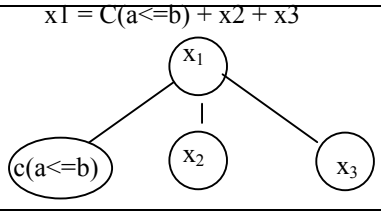
if a <= b then
S_1;
else
S_2;
endif;
	x1 = C (a<=b) + x2 + x3
	x2 = C (S_1) + x4
	x4 = C (S_2) + x5
	x5 = C (endif) = 0
		

Tabela 2 – Estrutura Condicional

O sistema irá construir a árvore com a raiz x1 e os filhos c(a<=b), x2 e x3. Para a construção da equação o sistema irá percorrer o ramo correspondente a x2 até que encontre um nodo folha, ou seja, que não possua mais filhos e que o seu valor correspondente possa ser definido. Ao ocorrer essa situação o sistema retorna na árvore atribuindo o valor calculado à variável de complexidade correspondente a esta estrutura, no nodo pai. No nodo pai duas situações podem ocorrer: 1) não haver mais filhos naquele nodo pai, neste caso já existe informação suficiente para o valor da complexidade ser definido. 2) o pai pode ter mais filhos, neste caso o sistema irá percorrer o caminho definido pelos filhos até que encontre um nodo folha e possa retornar na árvore e ter, desta

forma, subsídios para a avaliação da equação de complexidade correspondente. Isto é, a árvore é percorrida em profundidade, da esquerda para a direita.

Atribuição: **a** := **b** ;

A avaliação desta estrutura pode ser bastante simples, como o caso de uma atribuição de um valor a uma variável, ou pode ser necessário a avaliação de uma expressão ou função.

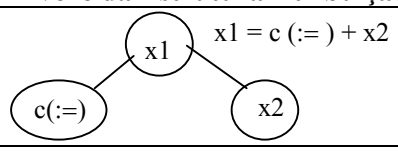
Programa	Análise do programa	Árvore da Estrutura Atribuição
a := b;	x1 = c (:=) + x2 x2 = c (b) + x3	

Tabela 3 – Estrutura Atribuição

A complexidade do algoritmo da Tabela 3 será a complexidade da estrutura de atribuição c(:=) mais a complexidade da avaliação de b e x3 a complexidade da próxima estrutura ainda não entificada e que será representada por x2.

Iteração: **for k** = **i** to **j** do **a** ;

Programa	Análise do programa	Árvore da Iteração

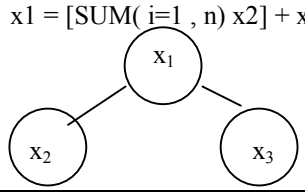
for i := 1 to n do
S_1;
endfor;
S2;
	x1 = c (for i:= 1 to n do x2) + x3
	x1 = [SUM(i=1 , n) x2] + x3
	x2 = c (S_1) + x4 = 1 + x4
	x4 = c (endfor)
		

Tabela 4 – Estrutura de Iteração

No exemplo da Tabela 4, a primeira estrutura encontrada foi a iteração, logo, a complexidade deste algoritmo é o somatório das complexidades das estrutura S1 que serão executadas por esta iteração, que foram identificadas por x2, mais a complexidade de uma próxima estrutura (S2), identificada por x3. Para se obter x1, o sistema irá percorrer a árvore até conseguir obter a definição de x2 e seus filhos e irá retornar para obter x3, desta forma o sistema consegue obter x1.

Iteração Condicional: while **a** do **b** ;

Programa	Análise do programa	Árvore da Iteração Condicional

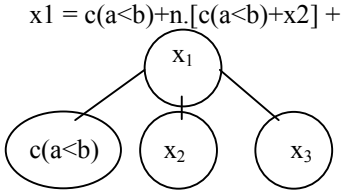
while a < b do
S_1;
endwhile;
S_2;
	x1 = C(while a < b do x2) + x3
	x1 = C (a<b) + f(n).[C (a<b) + x2] + x3
	x2 = C (S_1) + x4 = 1 + x4
	x4 = C (endwhile)
		

Tabela 5 – Estrutura de Iteração Condicional

No exemplo da Tabela 5, a complexidade do algoritmo é definida por c(a < b), mais a complexidade da estrutura S1, que será executada neste comando identificada por x2, mais a estrutura S2 seguinte do algoritmo, identificada por x3. O procedimento de cálculo é o mesmo aplicado à iteração não condicional. No cálculo desta estrutura, o usuário deverá fornecer o valor de f(n), avaliação do número de vezes que a iteração será executada.

Exemplo de Aplicação:

A Tabela 7 ilustra o exemplo de um algoritmo onde a primeira estrutura encontrada pelo sistema ANAC é a estrutura condicional, logo, a complexidade deste algoritmo será obtida pela soma da complexidade da avaliação da condição, mais a estrutura x2, que são as estruturas existentes no seu

bloco de comando, mais a próxima estrutura, x3. Com o prosseguimento do sistema é calculada a variável x2 como a complexidade da parte **then** mais a complexidade da parte **else**. Falta ainda definir a variável x3 para que a complexidade do algoritmo possa ser definida. a seguir é identificada uma de estrutura de iteração **for** e uma estrutura de continuação cuja complexidade é representada pela variável x7, sendo mais tarde identificada como a estrutura **end**. Uma vez definido seu valor de complexidade o sistema pode retornar na árvore e realizar a soma destas estruturas para obter o valor de x1 que irá representar a a complexidade do algoritmo. Pode-se observar na árvore gerada para o cálculo desta estrutura que, durante o processo de cálculo do programa são criadas as variáveis x',x" e x"' que atuam de maneira implícita na resolução da equação, não aparecendo durante a análise do programa que é visualizada pelo usuário do sistema.

Programa	Análise do programa	Árvore da Seqüência

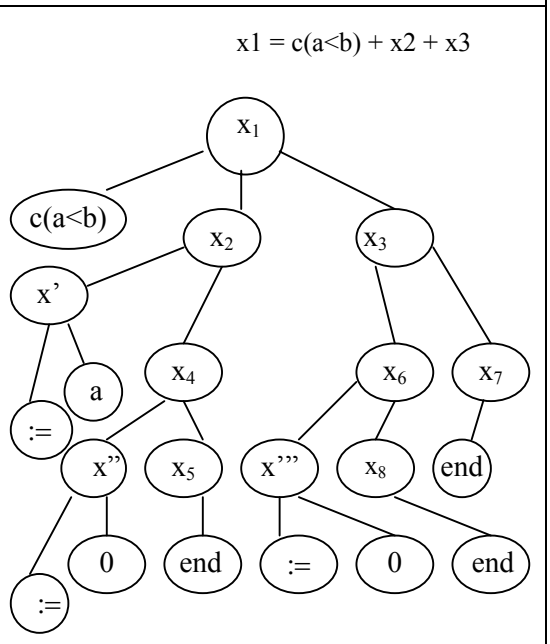
If a < b then
 	
vet[i] := a;
else
vet[i] := b;
endif;
for i:=1 to n do
vet[i] := 0;
 	
endfor;
end.
	x1 = c(a<b) + x2 + x3 =
	x1 = 1 + x2 + x3
	x2 = c(vet[i] := a) + x4 = 1 + x4
	x4 = c(vet[i] := b) + x5 = 1 + x5
	x5 = c (endif) = 0
	x4 = 1 + 0 = 1
	x2 = 1 + 1 = 2
	x3 = c (for i:= 1 to n do x6) + x7
	x3 = [SUM(i=1 , n) x6] + x7
	x6 = c(vet[i] := 0) + x8 = 1 + x8
	x8 = c (endfor) = 0
	x6 = 1 + 0 = 1
	x7 = c (end) = 0
	x3 = [SUM(i=1 , n) 1] + 0 =
	x3 = n + 0 = n
	x1 = 1 + 2 + n = n
	O (n)
		

Tabela 7 – Exemplo de aplicação

7 Conclusão

Neste artigo foi apresentado um método de cálculo da complexidade de algoritmos, e como ele foi utilizado na construção do protótipo de sistema ANAC. Os exemplos demonstrando a análise realizada pelo programa, mostrando claramente que o método pode ser aplicado à maioria das linguagens imperativas, que possuam estruturas do tipo: atribuição, condicional, iteração e iteração condicional.

O objetivo do método e do sistema é tornar a análise de algoritmos uma prática freqüente no projeto e desenvolvimento de algoritmos eficientes. A utilização do sistema ou método irá criar no projetista (usuário do ANAC) uma cultura de desenvolvimento em que esta prática acabe se tornando uma tarefa bem menos árdua do que pode-se ver atualmente em equipes ou profissionais que desenvolvem algoritmos e buscam eficiência e, no entanto, fogem à prática da análise da complexidade de algoritmos.

O sistema ANAC apresenta algumas vantagens em relação aos sistemas citados na seção 3. O fato da linguagem de especificação do algoritmo ser Pascal-like torna o ANAC um sistema mais utilizável, por ser o Pascal uma linguagem amplamente difundida no meio acadêmico, ao contrário de um sistema como METRIC que trabalha com a linguagem Lisp, cuja utilização é mais restrita. O ANAC possui um ambiente de trabalho mais simples de operar. Em sistemas como o Lambda-Upsilon-Omega o usuário necessita ter conhecimento do sistema operacional Unix, conhecer a linguagem de programação Lisp e ainda ter conhecimento do software matemático Maple. O

ANAC é um sistema interativo que vai guiando o usuário durante o processo de análise, isto faz com que o usuário vá adquirindo perícia no processo de análise e venha a tornar o processo de análise um hábito durante o processo de desenvolvimento de um algoritmo. Gera, sempre que possível, um resultado simplificado o que não é visto nos demais sistemas que podem gerar extensas e complicadas equações, que podem ser desestimulantes para quem não esteja habituado com o processo de cálculo de complexidade ou para quem está se iniciando no estudo da Complexidade de Algoritmos.

8 Referências

[1] Toscani, L. V. & Veloso, Paulo A. S. "Uma metodologia para cálculo da complexidade de algoritmos". *Simpósio Brasileiro de Engenharia de Software*, 4. Águas de São Pedro/SP, out. 24-26. Anais, SBC 1990.

[2] Barbosa, Marco A. C.; Toscani, Laira; Ribeiro, Leila. "Ferramenta para a Automatização da Análise da Complexidade de Algoritmos". In: *SBIE2000–XI Simpósio Brasileiro de Informática na Educação/SBC*. Anais..., Maceió, 2000.

[3] Aho, A.; Hopcroft, J.; Ullman, J. "Data Structures and Algorithms". Addison-Wesley. 1982.

[4] Knuth, D. E. "Algorithms and Program: Information and Data". Communication of ACM, New York, 26(1):56, janeiro. 1983. pp56.

[5] Veloso, P. A. S. & Veloso, S. R. M. "Problem Decomposition and Reduction: Applicability, Soundness, Completeness". In: Progress in Cybernetics and Systems Research. Washington, Hemisphere, 1981. v.8.

[6] Toscani, L. V. & Veloso, Paulo A. S. "Complexidade de Algoritmos". Sagra-Luzzatto. Porto Alegre, 2001.

[7] Wegbreit, B. "Mechanical program analysis". Communications of the ACM, 18(9):528-539, 1975.

[8] Metayer, D. Le. "Ace: An automatic complexity evaluator". ACM Transactions on Programming Languages and Systems, 10(2):248-266,1988.

[9] Flajolet, Philippe; Salvy, Bruno & Zimmermann, Paul. "ΛΥΩ: An Assistante Algorithms Analyser". In Proceedings AAECC'6, Lecture Notes in Computer Science 357, pg. 201-212, 1988. Also available as INRIA Reserch Report 876, 1988.

[10] Knuth, D. E. "The Art of Computer Programming". Addison-Wesley, Menlo Park, California, 1968.