

Complexidade de Algoritmos

Introdução à Ciência da Computação II

2.semestre/2006

Prof. Aneu de Andrade Lopes
Apresentação com material gentilmente cedido pelas
profas. Renata Pontin Mattos Fortes
<http://www.icmc.usp.br/~renata> e Graça Nunes :
<http://www.icmc.usp.br/~mdgvnune>

Complexidade de Algoritmos

- Complexidade de Algoritmos
- Ordens de Magnitude
- Definição de Ordem
- Relação entre as Ordens
- Quem é o parâmetro N?
- Upper e Lower Bounds
- Análise de Algoritmos X Análise de Classes de Algoritmos
- Classes de Problemas P e NP

Fonte: <http://www.icmc.sc.usp.br/~mdgvnune/download/sce5832/Teoria.html>

Complexidade de Algoritmos

Critérios para o julgamento de um programa:

- Ele faz exatamente o que queremos que faça?
- Ele o faz corretamente, de acordo com as especificações originais da tarefa?
- Há uma documentação que descreva como usá-lo e como ele trabalha?
- As sub-rotinas são criadas de tal modo que realizem sub-funções lógicas?
- O código é legível?
- Ele é executado em tempo ótimo de execução?
- Ele usa espaço de memória mínimo necessário para sua execução?

Eficiência

Complexidade de Algoritmos

Critérios vitais quando se constrói software

Os critérios:

- Ele é executado em tempo ótimo de execução?
- Ele usa espaço de memória mínimo necessário para sua execução?

têm relação com a performance do programa.

A avaliação da performance pode ser dividida em duas fases:

- (a) estimativas prévias e
- (b) testes posteriores.

Complexidade de Algoritmos

Estimativas Prévias:

Suponha a existência de um comando $x = x + 1$ em algum lugar do programa. Queremos estimar: (1) o tempo que tomará uma única execução sua, e (2) o número de vezes que ele será executado. O produto desses valores nos dará o tempo total gasto pelo comando.

A estatística (2) é chamada de Contagem de Freqüência, e ela pode variar de um conjunto de dados para outro. Assim, é necessário escolher amostras adequadas de dados para se estimar a freqüência.

Complexidade de Algoritmos

Quanto a (1), é impossível determinar exatamente quanto tempo leva para executar qualquer comando, a menos que tenhamos informações como:

- a máquina em que o programa está sendo executado;
- o conjunto de instruções da linguagem de máquina;
- o tempo gasto por cada instrução de máquina;
- a tradução que o compilador fará da L.Fonte para a L.Máquina.

Complexidade de Algoritmos

E mesmo assim, isso é insuficiente, pois:

- compiladores podem variar em diferentes máquinas;
- imprevisibilidade de ambientes de multiprogramação, multiprocessamento ou processamento distribuído.

Assim, vamos nos limitar a considerar apenas a contagem de freqüência de todos os comandos de um programa.

Além disso, vamos considerar um tempo constante para todo comando.

Complexidade de Algoritmos

Exemplos

<pre>x = x + 1</pre>	<pre>for (i = 1; i <= n; i++) x = x + 1</pre>	<pre>for (i = 1; i <= n; i++) for (j = 1; j <= n; j++) x = x + 1</pre>
(a)	(b)	(c)
Freq.(a) = 1	Freq.(b) = n	Freq.(c) = n ²

Complexidade de Algoritmos

Ordem de Magnitude

• 1, n e n^2 são ordens de magnitude diferentes e crescentes, assim como 1, 10, 100 seriam, se fizéssemos $n=10$.

• Em nossa análise, estaremos interessados principalmente em determinar a ordem de magnitude de um algoritmo, o que significa que estaremos preocupados com os comandos mais freqüentes.

Complexidade de Algoritmos

Ordem de Magnitude

Fórmulas Úteis

$$\sum_{1 \leq i \leq n} (1) = ?$$

$$\sum_{1 \leq i \leq n} (i) = ?$$

$$\sum_{1 \leq i \leq n} (i^2) = \frac{n(n+1)(2n+1)}{6} \Rightarrow \text{ordem } n^3$$

$$\text{Em geral, } \sum_{1 \leq i \leq n} (i^k) = \frac{n^{k+1}}{k+1} + \text{termos de menor grau,} \quad k \geq 0$$

Complexidade de Algoritmos

Ordem de Magnitude

Fórmulas Úteis

$$\sum_{1 \leq i \leq n} (1) = n$$

$$\sum_{1 \leq i \leq n} (i) = \frac{n(n+1)}{2} \Rightarrow \text{ordem } n^2$$

$$\sum_{1 \leq i \leq n} (i^2) = \frac{n(n+1)(2n+1)}{6} \Rightarrow \text{ordem } n^3$$

$$\text{Em geral, } \sum_{1 \leq i \leq n} (i^k) = \frac{n^{k+1}}{k+1} + \text{termos de menor grau,} \quad k \geq 0$$

Complexidade de Algoritmos

Ordem de Magnitude

Exemplo

No Exemplo de Fibonacci:

fib(n):

início

$x, y, k \leftarrow 0, 1, 0$ (3)

enquanto $k < n$ ($n+1$)

faça $x, y, k \leftarrow y, x + y, k + 1$; ($5 \cdot n$)

devolva x

fim

Este procedimento executa apenas $6n + 4$ operações para calcular $\text{Fib}(n)$. Expressamos esta frequência como $O(n)$, ignorando as constantes 6 e 4, significando que a ordem de magnitude é proporcional a n . Ou seja, conforme n cresce, o tempo gasto pelo algoritmo cresce linearmente em função de n .

Complexidade de Algoritmos

Definição de Ordem

Ordem

Ex. a função $0.01n^2 + 10n$ é dita ser *da ordem* da função n^2 , $O(n^2)$, pois quando ela cresce torna-se mais proximamente proporcional a n^2 .

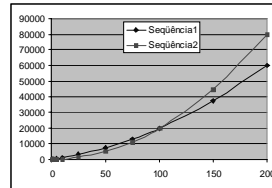
Definição: $f(n) = O(g(n))$ se e só se existem duas constantes c e n_0 tal que $|f(n)| \leq c |g(n)|$, para todo $n \geq n_0$.

n é um parâmetro que caracteriza o tamanho da entrada e/ou a saída do algoritmo.

Complexidade de Algoritmos

- 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 $g(n)$.

- Ex. $f(n) = n^2 + 100n$, $f(n)$ é $O(n^2) \Rightarrow c = 2, n_0 = 100$



Seq1: $n^2 + 100n$

Seq2: $2n^2$

Complexidade de Algoritmos

Definição de Ordem

Obs.:

$O(1)$ - tempo constante

$O(\log_2 n)$ - ordem logarítmica

$O(n)$ - ordem linear

$O(n^2)$ - ordem quadrática

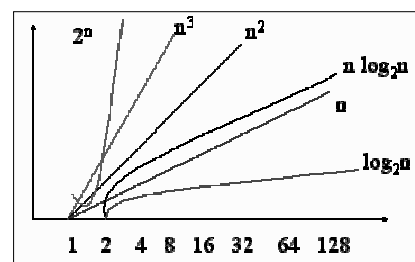
$O(n^3)$ - ordem cúbica

$O(2^n)$ - ordem exponencial

Complexidade de Algoritmos

Relação entre as Ordens

Relação entre as ordens



Complexidade de Algoritmos

Relação entre as Ordens

Função de Complexidade	Tamanho do problema n			
	10	10^2	10^3	10^4
$\log_2 n$	3.3	6.6	10	13.3
n	10	10^2	10^3	10^4
$n \log_2 n$	$0.33 \cdot 10^2$	$0.17 \cdot 10^3$	10^4	$1.3 \cdot 10^5$
n^2	10^2	10^4	10^6	10^8
n^3	10^3	10^6	10^9	10^{12}
2^n	1024	$1.3 \cdot 10^{30}$	$> 10^{100}$	$> 10^{100}$
$n!$	$3 \cdot 10^6$	$> 10^{100}$	$> 10^{100}$	$> 10^{100}$

Complexidade de Algoritmos

Método

Dado um algoritmo, calculamos a frequência de seus comandos e fazemos a soma de todas elas. A ordem de magnitude do algoritmo é dada pela ordem do(s) comando(s) mais frequente(s). Assim, se a soma é dada por um polinômio:

$$P(n) = c_k \cdot n^k + c_{k-1} \cdot n^{k-1} + \dots + c_1 \cdot n + c_0, \text{ onde } c_k \neq 0,$$

$$\text{então } P(n) = O(n^k)$$

Mas, se algum passo for executado, por exemplo, 2^n vezes, a expressão passa a ser $P(n) = O(2^n)$.

Complexidade de Algoritmos

Quem é o Parâmetro N?

Considere o papel do parâmetro n nas seguintes classes de problemas:

- Encontrar o maior número de uma seqüência de n inteiros.
- Resolver um conjunto de equações algébricas lineares $Ax = b$, onde A é uma matriz real $n \times n$ e b é um vetor real de tamanho n .
- Seja W um array unidimensional de n inteiros distintos. Ordenar as entradas de W em ordem crescente.
- Avaliar o polinômio $P_n(x) = \sum_{k=0}^n a_{n-k} x^k$ quando $x = x_0$.

Complexidade de Algoritmos

Quem é o Parâmetro N?

Em cada um desses problemas, o parâmetro n provê uma medida do tamanho do problema no sentido de que o tempo requerido para solucioná-lo, ou o espaço de armazenamento necessário, ou ambos, serão incrementados conforme n aumenta.

A ordem de magnitude dará exatamente a proporção em que ocorre esse incremento. Na verdade, $O(\)$ nos dá o limitante superior do recurso (tempo ou espaço) necessário para o algoritmo.

Complexidade de Algoritmos

Upper e Lower Bounds

•A análise do algoritmo nos fornece um *limite superior* (*upper bound*) para a quantidade de recursos que é *suficiente* para resolver um problema.

•Para saber se e quanto podemos melhorar este algoritmo, no entanto, precisamos estabelecer um *limite inferior* (*lower bound*) na quantidade de recursos necessários, ou seja, qual a quantidade mínima *necessária* de recursos para a resolução do problema.

•Idealmente, os dois limites, inferior e superior, deveriam ser iguais, pois neste caso conheceríamos exatamente a quantidade de recursos que é tanto *necessária* quanto *suficiente* para resolver um problema.

Complexidade de Algoritmos

Upper e Lower Bounds

•Se dispusermos de um algoritmo que utilize exatamente esta quantidade de recursos, então, teremos um *algoritmo ótimo* para a tarefa, no sentido de que a quantidade de recursos utilizada por qualquer outro algoritmo para a tarefa será maior, ou no melhor caso igual à do algoritmo que temos.

•A diferença entre o limite inferior e o superior nos dá uma medida de quanto um algoritmo pode ser melhorado. Nem sempre, no entanto, é possível se construir algoritmos ótimos.

Complexidade de Algoritmos

Análise de Algoritmos X Análise de Classes de Algoritmos

Objetivo: Identificar o "melhor algoritmo possível" da família de algoritmos que solucionam um determinado problema.

Exemplo1: Ordenação de n elementos (memória interna)

Critério: tempo de execução medido pelo número C de comparações entre chaves.

Lower Bound da Classe: $C \approx n \log_2 n - O(n \cdot \log_2 n)$

Upper Bound da Classe: $C \approx n^2 - O(n^2)$

Complexidade de Algoritmos

Análise de Algoritmos X Análise de Classes de Algoritmos

Métodos Diretos (inserção, seleção e troca): $O(n^2)$ no pior caso.

Métodos Avançados:

Shellsort (inserção melhorada): $O(n^{1.25})$ no pior caso.

Heapsort (seleção melhorada): $O(n \log_2 n)$ no pior caso.

Quicksort (troca melhorada): $O(n \log_2 n)$ no melhor caso e caso médio; $O(n^2)$ no pior caso.

n pequeno \rightarrow qualquer método direto.

Complexidade de Algoritmos

Análise de Algoritmos X Análise de Classes de Algoritmos

Atenção: algoritmos podem não alcançar o limite mínimo.

Exemplo3: Multiplicação de Matrizes $n \times n$

Lower Bound da classe: $O(n^2)$
 Algoritmo mais rápido: $O(n^2.8)$
 Algoritmo usual: $O(n^3)$

Complexidade de Algoritmos

Exemplo - Busca Sequencial

- Chaves → Campos usados para identificação de um registro no arquivo.
- Busca através de exames das chaves. Se um arquivo tem n registros, com K_i como o valor da chave do registro R_i , a recuperação é feita via exame das chaves K_n, \dots, k_1 .

- Pior caso → $n + 1$ comparações
 - Busca com êxito
 - Número de comparações em média
 $(n + 1) / 2$
 $N \uparrow$ eficiência \downarrow

Complexidade de Algoritmos

Exemplo - Busca Binária

- Arquivo ordenado seqüencialmente.
- Busca começa examinando o registro do meio do arquivo, ao invés de um por um, iniciando em uma das extremidades, como na busca seqüencial.
- Supondo que o arquivo está ordenada por valores de chave crescente, dependendo do resultado da comparação com a chave do meio:
 - $K < K_m$, registro procurado encontra-se na metade do arquivo com chaves de valores mais baixos;
 - $K = K_m$, registro encontrado;
 - $K > K_m$, registro procurado encontra-se na metade do arquivo com chaves de valores mais altos.

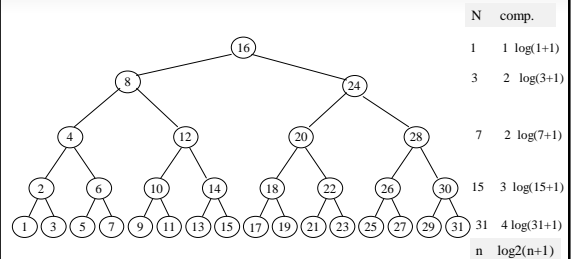
A cada comparação ou a busca termina ou o tamanho do arquivo a ser pesquisado é reduzido a metade do tamanho original

Comp. 1 2 3 ... n
 Tam. $n/2$ $n/4$ $n/8$... $n/2^k$

Estratégia: Dividir para conquistar!!

Complexidade de Algoritmos

Busca Binária



Considere que o valor de um módulo é o índice da chave a ser testada.
 $N=31$ registros; 1a chave a ser testada é K_{16} , pois $\lfloor (1+31)/2 \rfloor = 16$.