Análise de Algoritmos

Algoritmos e Estuturas de Dados
Inverno 2006

Cátia Vaz

Análise de Algoritmos

- Para avaliar e comparar o desempenho de 2 algoritmos podemos executar ambos várias vezes para ver qual é o mais rápido.
- Este método empírico pode fornecer indicações sobre o desempenho mas
  - consome demasiado tempo.
  - continua a ser necessário uma análise mais detalhada para validar os resultados.
- Existem bases científicas irredutíveis para caracterizar, descrever e comparar algoritmos.

Análise de Algoritmos

- Que dados usar?
  - **dados reais**: verdadeira medida do custo de execução
  - **dados aleatórios**: assegura-nos que as experiências testam o algoritmo e não apenas os dados específicos
    - **Caso médio**
  - **dados perversos**: mostram que o algoritmo funciona com qualquer tipo de dados
    - **Pior caso**
  - **dados benéficos**:
    - **Melhor caso**

Análise de Algoritmos

- Melhorar algoritmos:
  - analisando o seu desempenho
  - fazendo pequenas alterações para produzir um novo algoritmo
  - identificar as abstracções essenciais do problema.
  - comparar algoritmos com base no seu uso dessas abstracções.
- É fundamental para percebermos um algoritmo de forma a tomarmos partido dele de forma eficiente:
  - compararmos com outros
  - prevermos o desempenho
  - escolher correctamente os seus parâmetros

Análise de Algoritmos

- É fundamental separar a análise da implementação, i.e. identificar as operações de forma abstracta:
  - ex:quantas vezes ***id[p]*** é acedido
    - Uma propriedade (**imutável**) do algoritmo
  - não é tão importante saber quantos nanosegundos essa instrução demora no meu computador!
    - Uma propriedade do computador
- O número de operações abstractas pode ser grande, mas
  - normalmente o desempenho depende apenas de um pequeno número de parâmetros
  - procurar determinar a frequência de execução de cada um desses operadores (estabelecer estimativas).

Análise de Algoritmos

- Dependência nos dados de entrada
  - dados reais geralmente não disponíveis:
  - assumir que são aleatórios: **caso médio**
    - podem não ser representativos da realidade
  - perversos: **pior caso**
    - por vezes difícil de determinar
    - podem nunca acontecer na realidade
  - benéficos: **melhor caso**
- Normalmente os dados são boas indicações quanto ao desempenho de um algoritmo

Análise de Algoritmos

- O tempo de execução geralmente depende de um único parâmetro ***N***
  - tamanho de um ficheiro a ser processado, ordenado, etc
  - usualmente relacionado com o número de dados a processar
- Pode existir mais do que um parâmetro!

Análise de Algoritmos

- Os algoritmos têm tempo de execução proporcional a:
  - ***1***
    - muitas instruções são executadas uma só vez ou poucas vezes
    - se isto for verdade para todo o programa diz-se que o seu tempo de execução é constante
  - ***log N***
    - tempo de execução é logarítmico
    - cresce ligeiramente à medida que ***N*** cresce
    - quando ***N*** duplica ***log N*** aumenta mas muito pouco; apenas duplica quando ***N*** aumenta para N^2
  - ***N***
    - tempo de execução é linear
    - situação óptima quando é necessário processar ***N*** dados de entrada (ou produzir ***N*** dados na saída)

Análise de Algoritmos

- $N \log N$
  - típico quando se reduz um problema em sub-problemas, se resolve estes separadamente e se combinam as soluções
- N^2
  - tempo de execução quadrático
  - típico quando é preciso processar todos os pares de dados de entrada
  - prático apenas em pequenos problemas (ex: produto matriz - vector)

Análise de Algoritmos

- N^3
  - tempo de execução cúbico
  - ex: produto de matrizes
- 2^N
  - tempo de execução exponencial
  - provavelmente de pouca aplicação prática
  - típico em soluções de força bruta
  - ex: cálculo da saída de um circuito lógico de N entradas

Valores típicos de várias funções

$lg\ N$	$\sqrt{N}$	N	$N\ lg\ N$	$N\ (lg\ N)^2$	$N^{3/2}$	N^2
3	3	10	33	110	32	100
7	10	100	664	4414	1000	10000
10	32	1000	9966	99317	31623	1000000
13	100	10000	132877	1765633	1000000	100000000
17	316	100000	1660964	27588016	31622777	10000000000
20	1000	1000000	19931569	397267426	1000000000	1000000000000

segundos	
10^2	1.7 minutos
10^4	2.8 horas
10^5	1.1 dias
10^6	1.6 semanas
10^7	3.8 meses
10^8	3.1 anos
10^9	3.1 décadas
10^{10}	3.1 séculos
10^{11}	*nunca*

Conversão de Segundos

Resoluções de grandes problemas

operações por segundo	tamanho do problema 1 milhão			tamanho do problema 1 bilião		
	N	NlgN	N^2	N	NlgN	N^2
10^6	segundos	segundos	semanas	horas	horas	nunca
10^9	instantes	instantes	horas	segundos	segundos	décadas
10^{12}	instantes	instantes	segundos	instantes	instantes	semanas

Análise: funções relevantes

função	nome	valor típico	aproximação
x	*floor function*	$\lfloor 3.14 \rfloor = 3$	x
x	*ceiling function*	$\lceil 3.14 \rceil = 4$	x
lg N	*binary logarithm*	lg 1024 = 10	1.44 ln N
F_N	*Fibonacci numbers*	$F_{10} = 55$	
H_N	*harmonic numbers*	H_{10} 2.9	ln N + *g*
N!	*factorial function*	10! = 3628800	$(N/e)^N$
lg(N!)		lg(100!) 520	N lg N - 1.44N
	e = 2.71828 ... *g* = 0.57721 ... ln 2 = 0.693147 ... lg e = 1/ ln2 = 1.44269 ...		

Análise de Algoritmos

- **Números Harmónicos**
  - $H_N = \sum_{i=1}^{N} 1/i$
    - **ln N** - área debaixo da curva de ***1/x*** entre ***1*** e ***N*** (integração)
    - $H_N \approx \ln N + \gamma + 1/(12N)$
    - $\gamma = 0.57721$ (constante de Euler)
- **Números de Fibonacci**
  - $F_N = F_{N-1} + F_{N-2}$, ***para*** $N \geq 2$ ***com*** $F0 = 0$ ***e*** $F1 = 1$
- **Fórmula de Stirling**
  - $\lg N! \approx N \lg N - N \lg e + \lg\sqrt{2\pi N}$

Progressão Aritmética

- É uma sequência numérica em que cada termo, a partir do segundo, é igual à soma do termo anterior com uma constante r. O número r é chamado de **razão da progressão aritmética**.

$$\begin{cases} a_1 = a \\ a_i = a_{i-1} + r \text{ , } i>1 \end{cases} \longrightarrow a_n = a_1 + r(n-1) \longrightarrow S_n = \sum_{i=1}^{n} a_i = n(a_1 + a_n)/2$$

- S_n é a soma de todos os termos de uma progressão aritmética.

$$S_n = \sum_{i=1}^{n} a_i = n(a_1 + a_n)/2$$

  - Exemplo: a=0 e r=1

$$S_n = \sum_{i=1}^{n} i-1 = n(n-1)/2 \xrightarrow[k=i-1]{} = \sum_{k=0}^{n-1} k = n(n-1)/2$$

Cátia Vaz

Progressão Geométrica

- é uma sequência numérica em que cada termo, a partir do segundo, é igual ao produto do termo anterior por uma constante r. Assim, a progressão fica totalmente definida pelo valor de seu termo inicial a e sua razão r.

$$\begin{cases} a_1 = a \\ a_i = a_{i-1} \times r \text{ , } i>1 \end{cases} \longrightarrow a_n = a \times r^{n-1}$$

- S_n é a soma de todos os termos de uma progressão geométrica.

$$S_n = \sum_{i=1}^{n} a_i = a \times (r^n - 1)/(r - 1)$$

Cátia Vaz

Insertion Sort – análise do custo

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*[A]	c_1	n
do *key* $\leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Insertion Sort - custo

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

- **Melhor caso**: *array* ordenado

  → $t_j = 1$ para $j = 2, 3, \ldots, n$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).$$

- **Pior caso**: *array* inversamente ordenado

Sabendo que:

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

e que:

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

Notação O

- **Definição:** Seja $f, g: \mathbb{N}_0 \longrightarrow \mathbb{R}^+$. Diz-se que $f = O(g)$ se existirem $c > 0$ ($c \in \mathbb{R}^+$) e $n_0 \in \mathbb{N}_0$ tais que
  $f(n) \leq c.g(n)$, para todo o $n > n_0$.

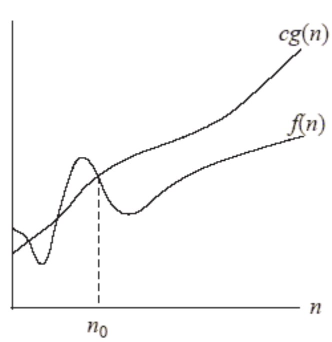
Nota:

- Por exemplo, $O(n^2)$ denota o conjunto de funções $\{ n^2, 17n^2, n^2 + 17n^{1.5} + 3n, n^{1.5}, 100n, \ldots \}$
- Logo, **$f = O(g)$** significa que $f \in O(g)$, i.e., f pertence ao conjunto de funções limitadas por g a partir de certa ordem.
- esta notação permite classificar algoritmos de acordo com **limites superiores** no seu tempo de execução.

Notação O

O-**notation**

$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.



$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

If $f(n) \in O(g(n))$, we write $f(n) = O(g(n))$

Notação O - Exemplos

Considere o seguinte código:

```
for (i = 0; i < N; i++) { instruções; }
```

- número de instruções: **N** iterações e em cada uma são executadas um conjunto de instruções em tempo constante - **O(N)**

Considere o seguinte código:

```
for (i = 0; i < N; i++) {
   for (j = 0; j < N; j++) {
        instruções; // executadas em O(1)
   }
}
```

- número de instruções: ciclo interno é **O(N)** e é executado **N** vezes - **O(N^2)**

Notação O - Exemplos

Considere o seguinte código:

```
for (i = 0; i < N; i++) {
   for (j = i; j < N; j++) {
        instruções; // executadas em O(1)
   }
}
```

número de instruções: ciclo interno é executado **N** iterações e em cada uma são executado

$N + (N-1) + (N-2) + \ldots + 3 + 2 + 1 = N(N+1)/2 = O(N^2)$

Notação O

- Exemplos de manipulações com a notação O:
  - $f = O(f)$
  - $c \,.\, O(f) = O(c \,.\, f) = O(f)$ (em que c>0)
  - $O(f) + O(g) = O(f+g) = O(max(f,g))$ (f,g assintoticamente não negativas)
  - $O(f) \,.\, O(g) = O(f \,.\, g)$
  - $O(f) + O(g) = O(f)$ se $g(N) \leq f(N)$ para $\forall N > N_0$

- Fórmula com termo contendo $O(...)$ diz-se **expressão assimptótica**.

Notação Ω

- **Definição:** Seja $f,g: \mathbb{N}_0 \longrightarrow \mathbb{R}^+$. Diz-se que $f = \Omega(g)$ se existirem $c > 0$ ($c \in \mathbb{R}^+$) e $n_0 \in \mathbb{N}_0$ tais que $0 \leq c.g(n) \leq f(n)$, para todo o $n > n_0$.

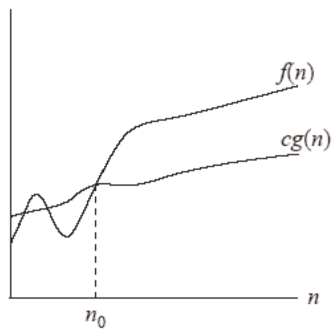
Nota:

- Por exemplo, $\Omega(n^2)$ denota o conjunto de funções $\{ n^2, 17n^2, n^2 + 17n^{1.5} + 3n, \ldots \}$
- Logo, **$f = \Omega(g)$** significa que $f \in \Omega(g)$, i.e., f pertence ao conjunto de funções limitadas por g a partir de certa ordem.
- esta notação permite classificar algoritmos de acordo com limites inferiores no seu tempo de execução.

Notação Ω

Ω-notation

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.



$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Notação Θ

- **Definição:** Seja f,g: $\mathbb{N}_0 \longrightarrow \mathbb{R}^+$. Diz-se que f = Θ(g) se existirem c1,c2 > 0 (c1,c2 ∈ $\mathbb{R}^+$) e n_0 ∈ $\mathbb{N}_0$ tais que $0 \le c1.g(n) \le f(n) \le c2.g(n)$, para todo o $n > n_0$.

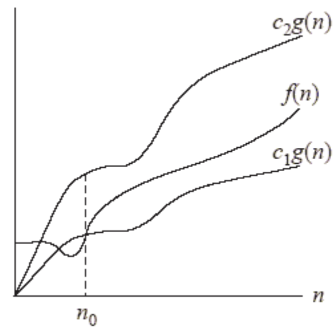
Nota:

- Logo, **f = Θ(g)** significa que f ∈ Θ(g).
- Esta notação permite classificar algoritmos de acordo com limites superiores e inferiores no seu tempo de execução.

Notação Θ

Θ-notation

$\Theta(g(n)) = \{f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.



$g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

Notações

- **Teorema:** Seja f,g: $\mathbb{N}_0 \longrightarrow \mathbb{R}^+$. f = Θ(g) se e só se f= Ω(g) e f=O(g).