Algoritmia Aplicada

Ano lectivo 2011-2012

Aula 2 – Análise de Algoritmos

Sumário

- Critérios de análise
- Determinação de T(n)
- Complexidade de um algoritmo
- Classificação de algoritmos
- Caso de algoritmos não determinísticos
- Classes de problemas

Critérios de análise

1. Tempo de execução (eficiência)
2. Memória a utilizar
   - **Contraditórios:** não é possível minimizar simultaneamente os dois
   - É determinante o critério da eficiência
   - **Tempo de execução** depende (para uma det. máquina) principalmente da **dimensão do input**

Critérios de análise

Exemplo:

- Tempo de ordenação de um conjunto de itens depende do nº de itens a ordenar, isto é, da **dimensão do input**.

Sendo:

n= dimensão do input
T= tempo de execução

T = T(n)

- Tempo de execução, enquanto medido em unidades de tempo (ex: segundos) é **variável de máquina para máquina**, para um **mesmo algoritmo**.

Então:

T(n) = Nº de instruções que são executadas (ou uma aproximação)

Critérios de análise

- Aferição da eficiência de um algoritmo **independente** da máquina em que ele corre.
- "Instrução executada" é diferente de "passo de algoritmo"

Exemplo:

```
1. A ← 2
2. DO FOR I=1 TO 10
3.       A ← A * A
4. PRINT (A)
```

4 passos / 23 instruções executadas

Critérios de análise

Exemplo:

Algoritmo de cálculo média de um conjunto de ***n*** valores

```
1. SOMA ← 0
2. I ← 1
3. DO WHILE I <= n
4.       SOMA ← SOMA + X[I]
5.       I ← I + 1
6. MEDIA ← SOMA /n
```

Algoritmo média conjunto valores

- **Determinação rigorosa de T (n)**

Instrução	Nº vezes executada
1	1
2	1
3	n + 1
4	n
5	n
6	1
Total	**3n+4**

- **T (n) = 3n + 4** ***(1)***

Algoritmo média conjunto valores

- **Determinação aproximada de T (n)**
  - **A determinação rigorosa de T(n) é difícil se o algoritmo tiver muitos passos**
  - **Aproximação: contabilizar apenas algumas instruções (as mais repetidas)**

Se contabilizamos apenas o ciclo do algoritmo (passos 3, 4 e 5), teremos:

- **T (n) ≈ 3n + 1 ≈ 3n** ***(2)***

Algoritmo média conjunto valores

Comparação das duas expressões de T(n)

- **(1) e (2) conduzem a valores idênticos quando n é elevado ($n \to \infty$)**

$$\lim \frac{(3n + 4) - (3n)}{(3n + 4)} = \lim \frac{4}{(3n + 4)} = 0$$

n	3n + 4	3n	Erro → 0
1	7	3	4/7
10	34	30	4/34
100	304	300	4/304
1000	3004	3000	4/3004

Complexidade de um algoritmo

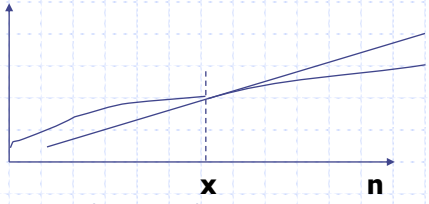
- **T(n) é de ordem de grandeza f(n)**

  T(n) = O (f(n))

  Se existir uma constante **c**, para a qual se tenha:

T(n) ≤ c. f(n) ; para todo o *n* suficientemente grande

- A "complexidade do algoritmo é O(f(n))" ou "O algoritmo é de ordem f(n)"



n ≥ x → T(n) ≤ c. f(n)

Complexidade de um algoritmo

Exemplo:

Algoritmo de cálculo média de um conjunto valores

Tem-se:

- T(n) = 3n + 4
- Fazendo c=4 e f(n) = n, teremos $3n + 4 \leq 4n, \forall n \geq 4$
- $3n + 4 \leq 4n, \forall n \geq 4$
- Então T(n)=O(f(n))
- O algoritmo tem complexidade O(n) ou
  O agoritmo é de ordem n

Determinação de T(n) - caso geral

Até agora:

- Só algoritmos em que o tempo de execução dependia exclusivamente da **dimensão do input (n)**
- Muitos problemas em que o tempo de execução é também influenciado pelo **modo como estão organizados** os itens que constituem o input

Exemplo: será mais rápido (em princípio) ordenar um conjunto de itens que se encontram quase ordenados do que ordenar esses mesmos itens quando se encontram por ordem inversa

Determinação de T(n) - caso geral

Dependência do modo como estão organizados os itens:

- Especificar qual o caso que se pretende considerar para a determinação de T(n):
  - **O melhor caso (o mais favorável)**
  - **O pior caso**
  - **O valor médio de T(n)**
  - **O valor de T(n) para uma entrada típica**
- Toma-se a expressão de T(n) para o caso médio

T(n) = Média aritmética dos vários T(n)

Determinação de T(n) – caso geral

Exemplo:

Algoritmo pesquisa linear de um item numa lista de valores

```
Procedure PLinear (LISTA, N, X, FOUND)
1.  FOUND ← false
2.  POS ← 1
3.  DO WHILE (NOT FOUND) and (POS <= n)
3.1     if LISTA [POS] = X
3.1.1   then FOUND ← true
3.1.2   else POS ← POS+1
```

n+1 casos dependendo da posição que item a pesquisar ocupa na lista

Algoritmo pesquisa linear

- **Determinação de T (n)**

n+1 casos

posição item a pesquisar ocupa na lista

(1)	**Melhor caso:**	**1ª posição**
(2)	**2º melhor caso:**	**2ª posição**
(3)	**3º melhor caso:**	**3ª posição**
(i)	**Iésimo melhor caso:**	**iésima posição**
(n)	**Enésimo melhor caso:**	**última posição**
(n+1)	**Pior caso:**	**não existe**

Algoritmo pesquisa linear

- **Determinação de T (n)**

Passo	**Nº vezes que é executado**				
1.	1	1	1	1	1
2.	1	1	1	1	1
3.	2	3	4	i+1	n+1
3.1	1	2	3	i	n
3.1.1	1	1	1	1	1
3.1.2	0	1	2	i-1	n-1
Total	**6**	**9**	**12**	**3i+3**	**3n+3**
CASO	**(1)**	**(2)**	**(3)**	**(i)**	**(n)**

Algoritmo pesquisa linear

- **Cálculo de T (n) para o caso médio**

$$T(n) = \left[\left(\sum_{i=1}^{n} (3i+3) \right) + (3n+3) \right] /(n+1)$$

***n* casos em que pesquisa tem sucesso**

um caso em que não tem sucesso

n+ 1 casos que podem ocorrer

$$T(n) = \left[3\sum_{i=1}^{n} i + (6n+3) \right] /(n+1)$$

Algoritmo pesquisa linear

- **Cálculo de T (n) para o caso médio**

$$T(n) = \left[3\sum_{i=1}^{n} i + (6n+3) \right] /(n+1)$$

$$\frac{1}{2}n(n+1)$$

$$= \frac{3}{2}n + \frac{6n+3}{n+1} = \frac{3}{2}n + 6 - \frac{3}{n+1}$$

Complexidade algoritmo é O(n)

f(n)=n e c=2:

$$\frac{3}{2}n + 6 \le 2n \Leftrightarrow 3n + 12 \le 4n \Leftrightarrow 12 \le n$$

Classificação de algoritmos

Casos mais freqs. f(n)	Complexidade O(f(n))	Algoritmo corre num tempo...
1	O(1)	constante
n	O(n)	linear
n^2	$O(n^2)$	quadrático
n^3	$O(n^3)$	cúbico
2^n	$O(2^n)$	exponencial
$\log_2 n$	$O(\log_2 n)$	logarítmico
$n\log_2 n$	$O(n\log_2 n)$	
$\log_2 \log_2 n$	$O(\log_2 \log_2 n)$	

Classificação de algoritmos

f(n)

$\log_2 \log_2 n$	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
---	0	1	0	1	1	2
0	1	2	2	4	8	4
1	2	4	8	16	64	16
1.58	3	8	24	64	512	256
2	4	16	64	256	4096	65536
2.32	5	32	160	1024	32768	4294967296
2.6	6	64	384	4096	2.6×10^5	1.85×10^{19}
3	8	256	2.05×10^3	6.55×10^4	1.68×10^7	1.16×10^{77}
3.32	10	1024	1.02×10^4	1.05×10^6	1.05×10^9	1.8×10^{308}
4.32	20	1048576	2.1×10^7	1.1×10^{12}	1.15×10^{18}	6.7×10^{315652}

Classificação de algoritmos

- **Algoritmos de complexidade exponencial**
  - ➢ Viáveis para resolver problemas em que a **dimensão do input é pequena**
  - Admitindo que:
    - Cada instrução de um algoritmo é executada em 1 microsegundo
    - n = 256 (dimensão do input)

Classificação de algoritmos

- **Algoritmos de complexidade exponencial**

Admitindo que:

- Cada instrução de um algoritmo é executada em 1 microsegundo
- n = 256 (dimensão do input)

f(n)	Tempo
$\log_2 \log_2 n$	3 microsegundos
$\log_2 n$	8 microsegundos
n	.25 milisegundos
$n \log_2 n$	2 milisegundos
n^2	65 milisegundos
n^3	17 segundos
2^n	3.7×10^{61} séculos !!!

Caso de algoritmos não determinísticos

- **Algoritmo determinístico:** para qualquer passo, a próxima instrução a ser executada está univocamente identificada e é descrita sem ambiguidades
- **Algoritmo não determinístico:** aquele em que surgem ambiguidades

Instrução (*Quando encontrar...*)

É permitida uma escolha feita de maneira não determinística

Quando encontrar o 3º semáforo,
Siga por uma das ruas que entroncam no cruzamento
Se...
Então...

Caso de algoritmos não determinísticos

- **Complexidade muito influenciada pela escolha que é feita sempre que surgem as ambiguidades**
- **Como calcular a complexidade?**
  - Em cada ponto de não determinismo é feita a opção **correcta;**
  - **"Opção correcta":** opção que conduz a um tempo óptimo de execução do algoritmo
- **É avaliado o desempenho (tempo de execução)**

Caso de algoritmos não determinísticos

- **É avaliado o desempenho (tempo de execução)**
  - Perante as ambiguidades, as opções são sempre as melhores.
  - O possível mau desempenho de um algoritmo em que foram feitas "más" opções, perante ambiguidades do mesmo, será atribuído, não ao algoritmo, mas à escolha que foi feita.

Classes de problemas

- **Problemas solúveis: têm uma solução algorítmica**
- **Problemas Insolúveis: não têm uma solução algorítmica**

Classes de problemas

- **Problemas Polinomiais**
  - **Algoritmo de resolução tem uma complexidade O(f(n));**
    f(n) é uma função limitada por um polinómio p(n)
  - **Todos os algoritmos de complexidade**
    O(n), O(n²), O(n³), O($\log_2 n$), O(n$\log_2 n$), O($\log_2 \log_2 n$)

f(n) p(n)

- **n ≤ n**
- **n² ≤ n²**
- **n³ ≤ n³**
- **$\log_2 n \leq n$**
- **N $\log_2 n \leq n^2$**
- **$\log_2 \log_2 n \leq n$**

f(n) <= p(n) ∀n

Classes de problemas

- **Problemas não Polinomiais**
  - **Algoritmos de complexidade O (2^n)**

"Listar todos os possíveis grupos de *1,2,3, ..., n* pessoas, de uma população de *n* pessoas"

- **2^n -1 grupos diferentes**
- **algoritmo tem de identificar 2^n -1 casos**
- **executadas, no mínimo, 2^n -1 instruções**

Próxima aula

- Processamento de cadeias de caracteres
  - Compressão