Sobre a análise de algoritmos...

Fevereiro 2007 **Armando B. Matos**

Índice

Sobre a eficiência dos algoritmos 2
Ordens de grandeza 9
Métodos de solução de recorrências 16
– Tabelar / suspeitar / demonstrar 21
– Diferenças finitas constantes 27
– Mudança de variável 32
– Equação característica homogénea 33
– Equação característica não homogénea 36
– Tempo médio do "quicksort" 38
Pequena bibiografia 40

Nota. Esta versão (2007) contém algumas correcções e pequenas alterações.

Eficiência dos algoritmos

Eficiência dos algoritmos: duas análises

1. Pormenorizada: mais exacta e directa mas em geral menos útil
   - Expressa em segundos.
   - Resultado da avaliação da eficiência (por exemplo: tempo gasto): único e dependente da velocidade e características do processador.
2. Através de ordens de grandeza: ignora as constantes multiplicativas e é uma análise assintótica ←
   - Expressa em *ordens de grandeza*
   - Resultado da avaliação da eficiência: paramétrico (uma função do comprimento dos dados) e independente da velocidade e características do processador.

Nesta disciplina usaremos a análise 2, "através de ordens de grandeza"!

O que é a "eficiência de um algoritmo"?
Resposta: a quantidade de recursos gastos durante a sua execução como função do *comprimento dos dados*.

Recursos? O que são?
Resposta: tempo (o mais ususal) e espaço (memória)

Modelos de computação mais usados:

1. Máquinas de Turing (MT)
2. Computadores de registos ("Random Access Machines")

Tempo e espaço, o que são?
Nas MT

1. Tempo: número de transições durante a computação
2. Espaço: número de células visitadas pela cabeça da MT durante a computação

Eficiência do algoritmo $\boxed{\text{A}}$ como função de quê?
Resposta: Do comprimento dos dados!

$$x \longrightarrow \boxed{\text{A}} \longrightarrow y$$

Seja $n = |x|$
Tempo de execução: depende dos dados, $t(x)$
Simplifica-se: como função do comprimento dos dados

$$t = f(n) \text{ onde } n = |x|$$

Mas... o tempo de execução é função de x e não de $n = |x|$!
Temos que escolher um tempo de entre todos os tempos $t(x)$
com $n = |x|$

Hipóteses mais frequentes:

1. Pior caso: $t(n) = \max\{t(x) : |x| = n\}$
2. Caso médio: $t(n) = E\{t(x) : |x| = n\}$ onde se assume uma determinada distribuição probabilística dos dados x de comprimento n; x e $t(x)$ são variáveis aleatórias. Significado de E: "valor médio".

Exemplo!

Seja o programa

```
//-- Dado: vector v[] com n elementos, v[0,1,...,n-1]
1    int i, m
2    m=0;
3    for i=1 to n-1
4      if v[i] > v[m] then
5         m=i
6    print m
```

É fácil ver que o tempo de execução é majorado por

$$\alpha n + \beta$$

onde α e β são constantes, ver por exemplo Knuth, "The Art of Computer Programming", vol. 1.

Concentremo-nos na seguinte questão:
Quantas vezes é executada a atribuição da linha 5?
Seja a esse número de vezes.

- Mínimo: $a = 0$ vezes
- Máximo: $a = n - 1$ vezes
- Médio: $E(a)$ =???

Hipóteses sobre a distribuição probabilística dos dados:

1. Todos os elementos de `v[]` são diferentes. Note que os valores do vector são irrelevantes, só é importante a sua "ordem", por exemplo, o comportamento para `v=[4,1,2,5,3]` e para `v=[41,2,5,55,30]` é idêntico.
2. Qualquer das $n!$ permutações é igualmente provável.

Calculemos alguns valores de $\mathsf{prob}\{a = i\}$

- $\mathsf{prob}\{a = 0\}$, probabilidade de ser $a = 0$ é $1/n = (n-1)!/n!$ = probabilidade de `v[0]` ser o maior de todos.
- $\mathsf{prob}\{a = n-1\}$ = é $1/n!$ pois só numa das $n!$ permutações isso acontece.

Teremos que calcular a média

$$\mathsf{E}(a) = 0 \times \mathsf{prob}\{a = 0\} + 1 \times \mathsf{prob}\{a = 1\} + \cdots + (n-1) \times \mathsf{prob}\{a = n-1\}$$

Vamos calcular $E(a)$ como uma função de n, seja $E(n)$. Temos uma recorrência:

1. $E(1) = 0$
2. Imaginemos que o vector tem $n+1$ elementos e calculemos como é que $E(n+1)$ depende de $E(n)$. Temos

$$E(n+1) = E(n) + 1 \times \mathsf{prob}\{\text{o último elemento ser o maior}\}$$

Ou seja

$$\begin{cases} E(1) = 0 \\ E(n+1) = E(n) + 1/n \end{cases}$$

A solução desta recorrência é fácil

$$E(n) = 0 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} = \left(\sum_{i=1}^{n-1} \frac{1}{i}\right) - 1$$

Por exemplo, para $n = 1000$ temos

- Mínimo = 0
- Máximo = 999
- Média $E(n) \approx 6.49$

Mas isto não é uma "forma fechada"!
Podemos determinar uma forma fechada?
Ou... talvez uma boa aproximação?

Exercício. *Confirme o resultado anterior para um vector de 3 elementos distintos, estudando as* $6 = 3!$ *situações possíveis:* $\diamond$

`v[0]<v[1]<v[2]`,... `v[2]<v[1]<v[0]`

Fim do exemplo

Investigar o caso geral! $E(n) = \ldots$?

Ordens de grandeza

Como "ignorar constantes e comportamento inicial"?
Resposta: **usando ordens de grandeza**.
Ordens de grandeza (ideia)

1. Majoração de $f(n)$ por $g(n)$:
   $f(n)$ é de ordem $O(g(n))$ – ou $f(n) \in O(g(n))$.
   Para valores suficientemente grandes de n é $f(n) \leq kg(n)$ onde k é uma constante real positiva.

2. Minoração de $f(n)$ por $g(n)$:
   $f(n)$ é de ordem $\Omega(g(n))$ – ou $f(n) \in \Omega(g(n))$.
   Para valores suficientemente grandes de n é $f(n) \geq kg(n)$ onde k é uma constante real positiva.

3. Ordem de grandeza exacta:
   $f(n)$ é de ordem $\Theta(g(n))$ – ou $f(n) \in \Theta(g(n))$.
   Para valores suficientemente grandes de n é $k_1 g(n) \leq f(n) \leq k_2 g(n)$ onde k_1 e k_2 são constantes reais positivas.

→ Definições matemáticas de $O()$, $\Omega()$ e Θ nas páginas seguintes!

Definições (como conjuntos):
Sejam f e g funções de $\mathbb{N}$ em $\mathbb{R}$. Seja $\mathbb{R}^+$ o conjunto dos reais positivos.

- Majoração: $g(n)$ é $O(f(n))$, f é majorante ("upper bound") de g

$$O(f(n)) = \{g(n) : \exists n_0 \in \mathbb{N}, \exists k \in \mathbb{R}^+, \forall n \geq n_0 : \ g(n) \leq kf(n)$$

- Minoração: $g(n)$ é $\Omega(f(n))$, f é minorante ("lower bound") de g

$$\Omega(f(n)) = \{g(n) : \exists n_0 \in \mathbb{N}, \exists k \in \mathbb{R}^+, \forall n \geq n_0 : \ g(n) \geq kf(n)$$

- $g(n)$ é $\Theta(f(n))$, f é da ordem de grandeza exacta de g

$$\Theta(f(n)) = \{g(n) : \exists n_0 \in \mathbb{N}, \exists k_1, k_2 \in \mathbb{R}^+, \forall n \geq n_0 : \ k_1 f(n) \leq g(n) \leq k_2 f(n)\}$$

Exercício. *Compare estas definições com as da página anterior; dê um significado a "suficientemente grande" por forma a que as definições coincidam.* ⋄

- $O(f(n))$, $\Omega(f(n))$ e $\Theta(f(n))$ são conjuntos! Quando dizemos n^2+3 é de ordem $O(n^3)$ queremos dizer $n^2 + 3 \in O(n^3)$.
- Para que serve o n_0, porquê apenas para valores de n suficientemente grandes?
  - O domínio de $f(n)$ pode não ser $\mathbb{N}$. Por exemplo, diz-se que $\log n^2$ é de ordem $O(n^2)$ embora a função não esteja definida para $n = 0$.
  - $f(n)$ pode tomar valores nulos ou negativos...
- A noção de ordem pode ser estendida para funções $\mathbb{R}_0^+ \to \mathbb{R}^+$.
- Gráficos ilustrativos (no quadro!)

Exercícios!

1. Verdadeiro ou falso? Justifique

   (a) $\log n \in O(n)$

   (b) $n \in O(\log n)$

   (c) $2n^2$ é de ordem $O(n^2)$

   (d) $2n^2$ é de ordem $O(n^3)$

   (e) $\Omega(n) = \Omega(3n)$ (igualdade de conjuntos)

   (f) 4 é $O(1)$

   (g) $O(n^2) \subseteq \Omega(n^2)$

2. Mostre que a seguinte função

$$f(n) = \begin{cases} 2n & \text{se } n \text{ é par} \\ 2 & \text{se } n \text{ é ímpar} \end{cases}$$

   é de ordem $O(n)$ mas não de ordem $\Theta(n)$.

3. Porque é que usualmente não se diz que uma determinada função é de ordem $O(2n)$ ou $\Omega(3n)$ ou $\Theta(4n)$? O coeficiente que se usa é sempre 1; assim, diz-se ordens $O(n)$, $\Omega(n)$ e $\Theta(n)$ respectivamente.

4. Considere a seguinte relação binária entre funções totais de $\mathbb{N}$ em $\mathbb{R}^+$: $f R g$ sse $f(n)$ é de ordem $O(g(n))$. Averigue se a relação é simétrica. Repita o exercício para a ordem de grandeza Θ.

5. Diz-se que $f(n)$ é de ordem $o(g(n))$, quando

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

   Por exemplo, $1/(n^2 \log n)$ é $o(1/n^2)$. Consegue encontrar alguma relação matemática entre esta definição e as "nossas" ordens de grandeza?

"Tempo" em algoritmos de ordenação e pesquisa

Para exprimir o tempo de execução usa-se muitas vezes

- o número de comparações envolvendo elementos do vector em questão

Exemplo

```
    //-- ordena v[0..n-1] pelo método da selecção do mínimo
    int i,j,m,t;
1   for i=0 to n-2
2   | m=i;
3   | for j=i+1 to n-1
4   |   if v[j]<v[m]
5   |    m=j
6   | t=v[i]; v[i]=v[j]; v[j]=m;
    // v está ordenado!
```

Exercício. *Quantas comparações* `v[j]<v[m]` *são efectuadas (linha* `5`*)? Exprima a sua resposta como uma função de* `n`. ◇

Exemplo

```
    //-- "merge" de a[0..m-1] e b[0..n-1] (já ordenados)
    //   em v[]
    //-- Ex: a=[1,4,5,8], b=[2,3,4] --> v=[1,2,3,4,4,5,8]
1   int i=0, j=0, k=0;
2   while i<m && j<n
3     if a[i]<b[j]
4       v[k]=a[i]; i=i+1; k=k+1;
5     else
6       v[k]=b[j]; j=j+1; k=k+1;
7   while i<m: v[k]=a[i]; i=i+1; k=k+1; // só um destes ciclos
8   while j<n: v[k]=b[j]; j=j+1; k=k+1; // é executado. Porquê?
```

(Note que um dos ciclos finais (linhas `7` e `8` é "executado" 0 vezes)

Exercício. *Determine o número de comparações* `a[i]<b[j]` *efectuadas (na linha* `3`*)*

1. *Valor mínimo (pode supor* `m<n`*).*
2. *Valor máximo.* Resposta: $m + n - 1$.

◇

Exercício. *Dê um exemplo de um algoritmo de ordenação em que o número de comparações* $c(n)$ *envolvendo elementos do vector não é da mesma ordem que o tempo de execução do algoritmo. Assim, neste caso,* $c(n)$ *não "exprime" correctamente o tempo de execução do algoritmo.* ◇

Solução de recorrências

Exemplos de funções definidas através de recorrências (definições indutivas)

Factorial

$$n! = \begin{cases} 1 & (\text{se } n = 0) \\ n(n-1)! & (\text{se } n \geq 1) \end{cases} \qquad \text{ou...} \qquad \begin{cases} f(0) = 1 & (\text{se } n = 0) \\ f(n) = nf(n-1) & (\text{se } n \geq 1) \end{cases}$$

Fibonacci

$$\begin{cases} f_0 = 0 \\ f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} & (\text{se } n \geq 2) \end{cases}$$

Que será?

$$\begin{cases} f(0) = 0 \\ f(n+1) = f(n) + 3n^2 + 3n + 1 & (\text{se } n \geq 0) \end{cases}$$

Que será?

$$\begin{cases} f(1) = 0 \\ f(2n) = 2f(n) + 2n & (\text{para } n \geq 1) \end{cases}$$

Nota: só está definido quando n é potência de 2.

O que é uma recorrência?
Resposta: um método de definir sucessões.
Uma sucessão $f : \mathbb{N} \to \mathbb{R}$ pode ser definida por um conjunto finito de equações dos seguintes tipos:

- Equações fronteira: $f(k) = c$ onde k é uma constante inteira não negativa e c é uma constante real.
  Por exemplo: $f(2) = 1$.
- Equações gerais:
  $$f(n) = \exp(\cdots) \text{ para } n\ldots$$
  onde n é uma variável inteira e $\exp(\cdots)$ é uma expressão que pode envolver valores de $f(i)$ para $i < n$.
  Por exemplo: $f(n) = f(n-1) + f(n-2)$ para $n \geq 2$.

As equações gerais e fronteira devem definir univocamente todos os valores de $f(i)$. Normalmente existe apenas uma equação geral.

Há uma relação muito próxima entre

recorrências | definições indutivas
demonstração por indução finita

Há uma relação próxima entre

recorrências $\leftrightarrow$ eq. diferenciais

Por vezes a solução é directa...

Um exemplo:

$$\begin{cases} f(0) = 1 \\ f(n+1) = f(n) + n \end{cases}$$

Nota. "$f(n+1) = f(n) + n$ para $n \geq 0$" é equivalente a "$f(n) = (n-1) + f(n-1)$ para $n \geq 1$" (mudança de variável).
Temos

$$\begin{aligned} f(n) &= (n-1) + f(n-1) \\ &= (n-1) + (n-2) + f(n-2) \\ &= \cdots \\ &= ((n-1) + (n-2) + \cdots + 1 + 0) + 1 \\ &= n(n-1)/2 + 1 \end{aligned}$$

Portanto a solução é

$$f(n) = \frac{n^2 - n + 2}{2}$$

Neste caso a solução directa foi possível porque sabemos somar progressões aritméticas.
Nota. O aluno deve conhecer as fórmulas das somas de progressões aritméticas e geométricas (ou saber deduzi-las...)

Um exercício

Exercício. *Determine directamente a solução da seguinte recorrência*

$$\begin{cases} f_1 = 2 \\ f_{n+1} = 3f_n \end{cases}$$

◇

Observações

- *Domínio.* Nem sempre uma recorrência define uma sucessão para todos os valores de n. Por exemplo, em algoritmos de ordenação pode simplificar-se a análise se se supuser que o número de elementos é uma potência de 2.
- *Existência.* para muitas recorrências não é conhecida – e possivelmente não existe – uma forma fechada para a sua solução.
- *Majoração.* Por vezes a solução de uma recorrência é muito difícil ou impossível, mas pode por vezes a partir dela definir outra mais simples cuja solução majora a função caracterizada pela recorrência; isso pode ser suficiente para, por exemplo, conhecer uma "ordem de grandeza" da solução.
  Isso pode acontecer, por exemplo, quando a função definida pela recorrência é o tempo de execução de um algoritmo.

Solução de recorrências

Método "tabelar → suspeitar → demonstrar"

O "método"

1. Tabelar: usando a recorrência tabelamos os primeiros valores da função; para ajudar no passo seguinte podem tabelar-se outras funções como n^2, 2^n, $\log n$, etc.
2. Suspeitar: eventualmente a tabela construída no passo anterior pode levar-nos a suspeitar que a solução é uma determinada função de n, seja $f(n)$.
3. Demonstrar: provamos – usualmente usando o método da indução finita – que $f(n)$ é de facto (se for!) a solução da recorrência.

Claro que se não conseguirmos avançar no passo 3. pode a suspeita estar errada e há que retroceder ao passo 2.

Um exemplo simples

$$\begin{cases} a_0 = 0 \\ a_{n+1} = a_n + 2n \end{cases}$$

Tabelar

n	a_n
0	0
1	0
2	2
3	6
4	12
5	20

Qual parece ser a solução?

Suspeitar

. . . coloquemos a coluna n^2

n	n^2	a_n
0	0	0
1	1	0
2	4	2
3	9	6
4	16	12
5	25	20

Agora, a suspeita é fácil!

$$f(n) = n^2 - n? \qquad \text{(suspeita)}$$

Demonstrar

O quê?

Teorema 1 *A solução da recorrência é* $n^2 - n$.

Dem. Por indução em n.
I. Caso $n = 0$: $0^2 - 0 = 0$. E da recorrência é $a_0 = 0$. $\surd$

II. Demonstrar que $\boxed{a_n = n^2 - n \;\Rightarrow a_{n+1} = (n+1)^2 - (n+1)}$.
Temos

$$\begin{array}{rcll} a_{n+1} & = & a_n + 2n & \text{(da recorrência)} \\ & = & (n^2 - n) + 2n & \text{(hipótese indutiva)} \\ & = & (n+1)^2 - (n+1) & \text{(contas simples!)} \end{array}$$

. . . e está demonstrada a implicação. $\diamond$

Exercício. *O "mergesort" é um método de ordenação muito eficiente, mesmo no pior caso. Descrição, supondo que* n *é uma potência de 2:*

`mergesort(v[0...n-1],n]:`

1. *Se* $n = 1$*: nada se faz*
2. *Se* $n \geq 2$*:*
   (a) `mergesort`$(v[0 \cdots n/2-1], n/2)$
   (b) `mergesort`$(v[n/2 \cdots n-1], n/2)$
   (c) `merge`$(v[0 \cdots n/2-1], v[n/2 \cdots n-1]) \longrightarrow v[0 \cdots n-1]$

Já definimos o que é o `merge`*, ver página 15.*
1. *Ilustre a execução do mergesort para o vector*

$$v[\,] = \boxed{9}\boxed{7}\boxed{8}\boxed{5}\boxed{1}\boxed{3}\boxed{6}\boxed{2}$$

2. *Da definição resulta a seguinte recorrência para um majorante do número de comparações efectuadas (são todas efectuadas nas chamadas de* `merge`*); use o máximo indicado na página 15 com* $m \to n/2$*,* $n \to n/2$*:*
Resposta: _____
3. *Resolva a recorrência pelo método "tabelar / suspeitar / demonstrar".* $\diamond$

Solução de recorrências

Método das diferenças finitas constantes

Definição. Seja a_n, $n \in \mathbb{N}$, uma sucessão. A diferença finita (ou simplesmente diferença) de 1ª ordem de a_n é a sucessão

$$D^1_n = a_{n+1} - a_n \qquad (\text{para } n \geq 0)$$

Para $i \geq 2$ a diferença de ordem i de a_n é a sucessão

$$D^i_n = D^{i-1}_{n+1} - D^{i-1}_n$$

Exemplo. Para a sucessão a_n definida pela recorrência

$$\begin{cases} a_0 = 1 \\ a_n = a_{n-1} + n^2 \quad \text{para } n \geq 1 \end{cases}$$

as diferenças de ordem 1 e 2 são

n	a_n	D^1_n	D^2_n
0	1	1	3
1	2	4	5
2	6	9	$\cdots$
3	15	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Teorema 2 *Se as diferenças finitas de ordem* m *da sucessão* a_n *são uma constante não nula, a solução da recorrência é um polinómio de grau* m*, isto é*

$$a_n = c_m n^m + c_{m-1} n^{m-1} + \cdots + c_1 n + c_0$$

com $c_m \neq 0$.

Nota. A *diferença* de ordem m é como que a imagem discreta, isto é, em termos das funções de $\mathbb{N}$ em $\mathbb{R}$, da *derivada* de ordem m. No domínio "contínuo" o Teorema anterior tem a seguinte "versão": se a derivada de ordem m de uma função $f(x)$ é uma constante não nula, então $f(x)$ é um polinómio de grau m em x.

Método 1 da solução da recorrência

1. Provar que, para um determinado inteiro m, as diferenças finitas de ordem m são uma constante não nula.
2. Usar o método dos coeficientes indeterminados ($m+1$ valores de a_n) para determinar os coeficientes do polinómio.

Método 2 da solução da recorrência

1. Tabelar os primeiros valores de: a_n, D^1_n,..., D^m_n.
2. Verificar que (se) D^m_n aparenta ser uma constante.
3. Usar o método dos coeficientes indeterminados ($m+1$ valores de a_n) para determinar os coeficientes do polinómio.
4. Provar que o polinómio obtido é solução da recorrência.

O seguinte resultado permite em alguns casos a aplicação directa do método das diferenças finitas constantes.

Teorema 3 *A solução de uma recorrência da forma* $t_0 = a$, $t_{n+1} = t_n + p(n)$ *onde* a *é uma constante e* $p(n)$ *um polinómio de grau* d *é um polinómio de grau* $d+1$.

Exercício. *(i) Continuar a tabela do exemplo anterior (pág. 28), mostrando que* D^3_n *aparenta ser uma constante positiva. Determinar a solução da recorrência (método 2). (ii) Resolver directamente a recorrência usando o Teorema 3.* $\diamond$

Exercício. *Determinar uma fórmula fechada para a soma dos primeiros* n *quadrados,*

$$S_n = \sum_{i=1}^{n} i^2$$

Use o exercício anterior. $\diamond$

Solução de recorrências
Método da mudança de variável

Vamos usar um exemplo.
Seja a recorrência

$$\begin{cases} a(1) = 1 \\ a(2n) = a(n) + 1 \end{cases}$$

Note que $a(n)$ só fica definido quando n é potência de 2, $n = 2^p$, $p \geq 0$.
É natural efectuar a mudança de variável $n = 2^p$ (ou $p = \log n$). Isto é vamos representar $a(b)$ por uma outra recorrência $b(p)$ sendo

$$a(n) \equiv b(p)$$

Fica

$$\begin{cases} b(0) = 1 \\ a(p+1) = b(p) + 1 \end{cases}$$

cuja solução é muito simples, $b(p) = p + 1$, ou seja, em termos de n:

$$a(n) = \log n + 1$$

Solução de recorrências
Método da equação característica homogénea

Estudamos agora a solução de recorrências em f_n com uma única equação geral da forma

$$a_0 f_n + a_1 f_{n-1} + \cdots + a_k f_{n-k} = 0$$

onde k é uma constante e $a_0 \neq 0$.
Exercício. *Mostre que as equação geral da sequência de Fibonacci é da forma indicada.* $\diamond$
Se experimentarmos soluções da forma $f_n = r^n$ vemos que r deve satisfazer a equação (dita característica)

$$a_0 r^k + a_1 r^{k-1} + \cdots + a_r = 0$$

Teorema 4 *Se $f_n = f_1(n)$ e $f_n = f_2(n)$ são soluções da equação $a_0 f_n + a_1 f_{n-1} + \cdots + a_k f_{n-k} = 0$, então, sendo α e β quaisquer constantes, $\alpha f_1(n) + \beta f_2(n)$ também é solução dessa equação.*

Exercício. *Demonstre este resultado.* $\diamond$

Teorema 5 *Se as raízes da equação característica são todas distintas (reais ou complexas), sejam $r_1, \ldots, r_k$, a solução da equação geral é exactamente constituída pelas sucessões da forma*

$$f_n = \alpha_1 {r_1}^n + \alpha_2 {r_2}^n + \cdots + \alpha_k {r_k}^n$$

onde α_1, $\alpha_2, \ldots$, e α_k são constantes arbitrárias.

Nota. Entre as soluções possíveis está $f(n) = 0$ (solução identicamente nula) que se obtém com $\alpha_1 = \alpha_2 = \ldots = \alpha_k = 0$.

Método da equação característica homogénea (raízes distintas)

Método:

1. Determine a equação característica e as suas raízes, $r_1, \ldots, r_k$.
2. Determine os coeficientes α_i, para $i = 1, 2, \cdots, k$, através de um sistema de k equações lineares da forma

   $$\alpha_1 {r_1}^n + \alpha_2 {r_2}^n + \cdots + \alpha_k {r_k}^n = f_n$$

   para k valores de f_n distintos que calculou separadamente, por exemplo, para $n = 0, 1, \ldots, k-1$.

Exercício. *Aplique o método descrito à solução geral da sequência de Fibonacci (ver página 17). Confirme no final 2 valores da solução obtida.*
Resposta (a preencher pelo aluno!):

- *A equação geral é...*
- *A equação característica é...*
- *As raízes da equação característica são...*
- *A solução geral é da forma...*
- *A solução é...*

◇

Equação característica homogénea
Caso geral: existência de raízes múltiplas

Teorema 6 *Se* m *é a multiplicidade de uma raíz* r *da equação característica, as seguintes sucessões, bem como todas as suas combinações lineares, são soluções da equação geral*

$$r^n,\ nr^n,\ n^2r^n, \cdots,\ n^{m-1}r^n$$

Mais geralmente se as raízes da equação característica forem $r_1, \ldots,\ r_p$ *de multiplicidades respectivamente* $m_1, \ldots,\ m_p$*, a solução geral é uma qualquer combinação linear da forma*

$$\sum_{i=1}^{i=p} \sum_{j=0}^{m_i-1} \alpha_{ij} n^j {r_i}^n$$

Não é difícil mostrar-se que uma combinação linear da forma indicada é sempre solução da equação geral. Também é verdade o recíproco: qualquer solução da equação geral é da forma indicada!.

Exercício. *Resolva a recorrência* $t_n = 2t_{n-1} - t_{n-2}$*, sendo*

$$\begin{cases} t_0 = 0 \\ t_1 = 1 \end{cases}$$

◇

Solução de recorrências
Método da equação característica não homogénea
Casos em que se conhece a solução

Suponhamos que a equação geral da recorrência tem a forma

$$a_0 f_n + a_1 f_{n-1} + \cdots + a_k f_{n-k} = b^n p(n)$$

onde

- b é uma constante
- $p(n)$ é um polinómio em n; seja d o seu grau

Teorema 7 *As soluções da recorrência com a equação geral indicada podem obter-se a partir das raízes da equação*

$$(a_0 r^k + a_1 r^{k-1} + \cdots + a_k)(r - b)^{d+1} = 0$$

usando o método que foi explicado para o caso das equações homogéneas.

Exercício. *Determine a forma geral da solução da recorrência*

$$\begin{cases} t_0 = 2 \\ t_n = 2t_{n-1} + 3^n \quad (n \geq 1) \end{cases}$$

Resposta (a preencher pelo aluno!):

- *A equação geral é...*
- *A equação característica é...*
- *Portanto* $b = \cdots$, $p(n) = \cdots$, $d = \cdots$.
- *A solução geral é da forma...*
- *A solução da recorrência é...*

◇

Uma generalização do Teorema 7
(equação característica não homogénea)

Teorema 8 *Se a equação geral da recorrência tem a forma*

$$a_0 f_n + a_1 f_{n-1} + \cdots + a_k f_{n-k} = b_1^n p_1(n) + b_2^n p_2(n) + \cdots + b_m^n p_m(n)$$

onde os b_i *são constantes e os* $p_i(n)$ *são polinómios em* n *de grau* d_i*, então as soluções da recorrência correspondente podem ser obtidas a partir das raízes da equação*

$$(a_0 r^k + a_1 r^{k-1} + \cdots + a_k)(r - b_1)^{d_1+1}(r - b_2)^{d_2+1} \cdots (r - b_m)^{d_m+1} = 0$$

usando o método que foi explicado para o caso das equações homogéneas.

Exercício. *Determine a forma geral da solução da recorrência*

$$\begin{cases} t_0 = 2 \\ t_n = 2t_{n-1} + n + 2^n \quad (n \geq 1) \end{cases}$$

◇

Nota. Será fornecido o enunciado do Teorema 8 nas provas em que isso for necessário.

Tempo médio do "quicksort"

Supõe-se que o leitor conhece o algoritmo de ordenação "quicksort" que se esquematiza de seguida

```
// ordena a secção v[a..b] do vector v[ ]
void quicksort(v,a,b)
  if a>=b  return
  else
    m = split(v,a,b)
    quicksort(v,a,m-1)
    quicksort(v,m+1,b)
```

O efeito da execução de `m=split(v,a,b)` é alterar o vector `v[]`, definindo um índice `m` tal que todos os elementos da secção do vector `v[a..m-1]` são menores ou iguais a `v[m]` e todos os elementos da secção do vector `v[m+1..b]` são maiores que `v[m]`.

Seja $n = b - a + 1$ o número de elementos a ordenar. Na execução de `m=split(v,a,b)` há $n - 1$ comparações envolvendo elementos do vector.

Pretendemos mostrar que o número de comparações $c(n)$ é de ordem $O(n \log n)$; mais precisamente que é sempre $c(n) \leq kn \log n$ para uma constante k apropriada.

Admitimos que os elementos do vector são todos distintos e que todas as $n!$ permutações de elementos são igualmente prováveis. Assim, o valor de `m` dado por `m=split(v,a,b)` tem a mesma probabilidade se ser `a`, de ser $a + 1$,... e de ser `b`; essa probabilidade é $1/n$ (relembra-se que $n = b - a + 1$).

Obtemos a recorrência que determina o valor médio do número de comparações

$$\begin{cases} c(n) = 0 & \text{para } n \leq 1 \\ c(n) = \sum_{i=1}^{n} \frac{1}{n}(n - 1 + c(i-1) + c(n-i)) & \text{para } n \geq 2 \end{cases}$$

Note-se que para cada $i \in \{1, 2, \ldots, n\}$ o valor $c(i)$ ocorre 2 vezes na soma anterior; podemos então escrever a recorrência da forma seguinte

$$\begin{cases} c(n) = 0 & \text{para } n \leq 1 \\ c(n) = \frac{2}{n}\sum_{i=1}^{n-1}(n-1+c(i)) & \text{para } n \geq 2 \end{cases}$$

Pré-teorema *A solução* $c(n)$ *da recorrência é majorada por* $kn\log n$ *em que o valor de* k *é encontrado na demonstração.*
Caso base, $n = 1$: temos (da recorrência) $c(1) = 0 \leq k(1\log 1)$ para qualquer $k > 0$; analogamente se trata o caso $n = 0$.
Passo de indução: suponhamos que $c(i) \leq ki\log i$ para todo o i com $0 \leq i \leq n-1$. Vamos mostrar que $c(n) \leq k(n\log n)$.

$$\begin{aligned} c(n) &= n-1+\frac{2}{n}\sum_{i=1}^{n-1}c(i) \\ &\leq n-1+\frac{2k}{n}\sum_{i=2}^{n-1}(i\log i) \\ &\leq n-1+\frac{2k}{n}\int_{i=2}^{n}(i\log i)\,di \\ &= n-1+\frac{2k(n^2\log n/2 - n^2/4 - 2\ln 2+1)}{n} \\ &= n-1+kn\log n - kn/2+\alpha \end{aligned}$$

onde $\alpha = 2k(-2\ln 2+1)/n$ tem valor negativo. O teorema fica então demonstrado se for $n-1 \leq kn/2$, pois fica então $k(n\log n)$. Em particular podemos tomar $k = 2$ e enunciar a versão final do "pré-teorema" anterior.

Teorema 9 *O número médio de comparações efectuado pelo "quicksort" não excede* $2n\log n$

Vemos pois que o número de comparações efectuadas pelo "quicksort" é, no caso médio, majorado por $2n\log n$. Admitimos que o leitor conhece o comportamento no pior caso do "quicksort" e as situações em que esse comportamento se verifica. O número de comparações efectuado é quadrático. Para mais pormenores ver por exemplo o livro de Cormen, Liederson, Rivest e Stein.

Pequena bibliografia

⋆ Brassard and Bratley. *Algorithmics, Theory and Practice*, Prentice-Hall International Editions, 1988, em especial:

- Capítulo I: (generalidades)
- Capítulo II, páginas: 37–43, 65–75

– R. Sedgewick, P. Flajolet, *An Introduction to the Analysis of Algorithms*. Pearson, 1998.

– Udi Manber, *Introduction to Algorithms : A Creative Approach*, Addison-Wesley, 1989.

– Thomas H. Cormen, Ronald L. Rivest, Clifford Stein, Charles Eric Leiserson, *Introduction to Algorithms*, Second edition, MIT Press, 2001.