INF 02779 Análise de Algoritmos - 2002/2	12/09/02
Aula 5: De volta para Fibonacci	
Instrutor: Berilhes Borges Garcia	Escriba: Humberto K. Sasaki

DRAFT

1 De volta para Fibonacci

Nós podemos fazer melhor ?

Sim, os algoritmos 2 e 3 não são os melhores !
Aqui está um truque matemático com matrizes:

Teorema 1.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} \tag{1}$$

Proof. (por Indução)

Observação 2. *Você não tem de recordar muita álgebra linear para provar isto, basta lembrar da fórmula de multiplicação de matrizes 2x2.*

1. Caso base:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F(2) & F(1) \\ F(1) & F(0) \end{pmatrix} = A \tag{2}$$

2. Assuma por indução que a equação é verdade para algum n. Multiplique ambos os lados por outra potência de A utilizando a fórmula para multiplicação de matriz:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} = \begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \tag{3}$$

Os termos obtidos são exatamente os mesmo obtidos através da definição de números de Fibonacci:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} = \begin{pmatrix} F(n+2) & F(n+1) \\ F(n+1) & F(n) \end{pmatrix} \tag{4}$$

$\square$

2 Outro Algoritmo para Números de Fibonacci

Agora utilizamos o truque anterior para definir outro algoritmo iterativo, utilizando multiplicação de matrizes.
Inicialize a matriz M para a matriz identidade (A^0). Repetidamente multiplique M por A de modo a formar a $(n-1)-$ésima potência de A.
Pela fórmula anterior, o elemento do canto superior esquerdo contém $F(n)$.

Algoritmo 3. *Algoritmo 4*
Fib(n)

1. $M[2][2] = \{\{1,0\},\{0,1\}\}$
2. *for* $i \leftarrow 1$ *to* n
3. $\quad M \leftarrow M * \{\{1,1\},\{1,0\}\}$
4. *return* $M[0][0]$

2.1 Análise do Algoritmo 4

Este demanda tempo $O(n)$, melhor que o Algoritmo 1, e provavelmente um pouco mais lento que o Algoritmo 2 ou o Algoritmo 3.
Note que a notação *big O* oculta as diferenças entre estes algoritmos; dessa forma necessitamos ser cuidadosos ao afirmar qual é o melhor.
Com relação a demanda de espaço, o Algoritmo 3 utiliza somente $O(1)$ bytes de memória.
O que ganhamos com este novo algoritmo ?

3 Mais um algoritmo

Podemos computar A^{n-1} mais rapidamente por meio de quadrados repetidos. Por exemplo, para computar 3^8 podemos fazer:

$$3^2 = 9, 9^2 = 3^4 = 81, 81^2 = 3^8 = 6561$$

Algoritmo 4. *Algoritmo 5*
$M[2][2] = \{\{1,1\},\{1,0\}\}$
Fib(n)

1. *Matpow*$(n-1)$
2. *return* $M[0][0]$

Matpow(n)

1. *if*$(n > 1)$
2. $\quad$ *Matpow*$(\lfloor \frac{n}{2} \rfloor)$
3. $\quad M \leftarrow M * M$
4. $\quad if(n\ \acute{e}\ \acute{i}mpar) M \leftarrow M * \{\{1,1\},\{1,0\}\}$

3.1 Análise do Algoritmo 5

Podemos observar que praticamente toda a demanda de tempo está situada na sub-rotina $Matpow$, que é recursiva: compute a n-ésima potência de A por meio de elevar-se ao quadrado a $\lfloor \frac{n}{2} \rfloor$-ésima potência de A.

Se n é ímpar, arredonde $\frac{n}{2}$ para baixo e eleve ao quadrado o resultado, calculando dessa forma a $(n-1)$-ésima potência de A. A seguir corrigimos a distorção multiplicando por A.

Relembre que algoritmos recursivos podem ter a sua demanda de tempo expressa por meio de uma equação de recursão. Neste caso específico, a demanda de tempo total será a demanda de tempo de $Matpow$ mais a demanda de tempo na chamada recursiva com argumentos $\lfloor \frac{n}{2} \rfloor$.

Note que a demanda de tempo em $Matpow$ é uma constante e que, portanto, de acordo com a notação *big O* é $O(1)$. Logo, a equação de recursão do algoritmo 5 é:

$$T(n) = T\left(\frac{n}{2}\right) + O(1) \tag{5}$$