Cursos de Verão 2010 - IME/USP
Tópicos de Programação
Recursão
Prof. Alexandre da Silva Freire

1 Elemento mínimo em um vetor: versão recursiva

Considere o seguinte problema problema: dado um vetor $v[0 \cdots n-1]$ de números inteiros, encontrar um inteiro min tal que $0 \leq min \leq n-1$ e $v[min] \leq v[j]$, para $j = 0, 1, \ldots, n-1$. Eis um algoritmo *recursivo* para o problema:

```
    int minimo(int v[], int n) {
       int min;

(2)    if(n == 1) {
(3)       return 0;
       } else {
(4)       min = minimo(v, n-1);

(5)       if(v[n-1] < v[min]) {
(6)          return n-1;
          } else {
(7)          return min;
          }
       }
    }
```

Provaremos, por indução em n, que o algoritmo acima está correto.

Lema 1.1 *O algoritmo* `minimo` *está correto.*

Prova. Provaremos por indução em n. Suponha que $n = 1$ (base da indução). Neste caso a linha (3) executada e o algoritmo devolve 0, que o nico ndice dentro vetor e, consequentemente, $v[0]$ o elemento mínimo de v. Agora suponha que $n > 1$ (passo da indução). Por hipótese de indução, temos que *a chamada função* `minimo(v, n-1)` *feita na linha (4) devolve o ndice de um elemento mínimo de* $v[0\,..\,n-2]$. Caso $v[n-1] < v[min]$, temos que $v[n-1]$ o elemento mínimo de $v[0\,..\,n-1]$, pois por hipótese de indução $v[min]$ um elemento mínimo de $v[0\,..\,n-2]$. Neste caso a linha (6) executada e o algoritmo devolve $n-1$, que o ndice de um elemento mínimo de $v[0\,..\,n-1]$. Caso $v[n-1] \geq v[min]$, temos que $v[min]$ o elemento mínimo de $v[0\,..\,n-1]$, pois por hipótese de indução $v[min]$ um elemento mínimo de $v[0\,..\,n-2]$. Neste caso a linha (7) executada e o algoritmo devolve min, que o ndice de um elemento mínimo de $v[0\,..\,n-1]$. Portanto, o algoritmo `minimo` está correto.

■

Qual o consumo de tempo do algoritmo `minimo`? Seja $T(n)$ o consumo de tempo do algoritmo `minimo` para uma entrada de tamanho n. Definiremos $T(n)$ através a seguinte recorrência:

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ T(n-1) + 1, & \text{se } n > 1. \end{cases}$$

Lema 1.2 $T(n) = n$.

Prova. Por indução em n. Suponha que $n = 1$. Neste caso $T(n) = 1 = n$. Agora suponha que $n > 1$. Neste caso $T(n) = T(n-1) + 1$. Por hipteste de indução, temos que $T(n-1) = n-1$. Logo, $T(n) = n-1+1 = n$. Portanto, $T(n) = n$ para $n \geq 1$. ■

2 Sequência de Fibonacci

O n-ésimo número da sequência de Fibonacci definido pela seguinte recorrência:

$$F(n) = \begin{cases} 1, & \text{se } 1 \leq n \leq 2 \\ F(n-2) + F(n-1), & \text{se } n > 2. \end{cases}$$

Utilizando a recorrência acima, temos que os oito primeiros números de Fibonacci so: $1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Eis um código em C para calcular o n-ésimo número de Fibonacci:

```
    int fibonacci(int n) {
       if(n <= 2) {
(2)       return 1;
       } else {
(3)       return fibonacci(n-2) + fibonacci(n-1);
       }
    }
```

Lema 2.1 *O algoritmo* `fibonacci` *está correto.*

Prova. Provaremos por indução em n. Suponha que $1 \leq n \leq 2$ (base da indução). Neste caso a linha (2) executada e o algoritmo devolve 1, que o n-ésimo número de Fibonacci. Agora suponha que $n > 2$ (passo da indução). Por hipótese de indução, temos que *as chamadas* `fibonacci(n-1)` e `fibonacci(n-2)` *feitas na linha (3) devolvem o* $(n-1)$*-ésimo e o* $(n-2)$*-ésimo números de Fibonacci, respectivamente*, que somados correspondem ao n-ésimo número de Fibonacci. Logo, quando $n > 2$, a linha (3) executada e, portanto, o algoritmo devolve o n-ésimo número de Fibonacci. ■

Seja $T(n)$ o consumo de tempo do algoritmo `fibonacci` para uma entrada de tamanho n. Definiremos $T(n)$ através a seguinte recorrência:

$$T(n) = \begin{cases} 1, & \text{se } n \leq 2 \\ T(n-2) + T(n-1) + 1, & \text{se } n > 2. \end{cases}$$

Limitaremos $T(n)$ através das duas seguintes recorrências:

$$I(n) = \begin{cases} 1, & \text{se } n \leq 2 \\ 2I(n-2) + 1, & \text{se } n > 2, \end{cases}$$

$$S(n) = \begin{cases} 1, & \text{se } n \leq 2 \\ 2S(n-1) + 1, & \text{se } n > 2. \end{cases}$$

Lema 2.2 *Para $n \geq 1$ vale que,*

$$I(n) = \begin{cases} 2^{\frac{n}{2}} - 1, & \text{se } n \text{ for par} \\ 2^{\frac{n+1}{2}} - 1, & \text{se } n \text{ for ímpar.} \end{cases}$$

Prova. Por indução em n. Temos dois casos como base da indução ($n = 1$ e $n = 2$). Suponha que $n = 1$. Temos que $I(n) = 1 = 2^{\frac{n+1}{2}} - 1$. Suponha que $n = 2$. Temos que $I(n) = 1 = 2^{\frac{n}{2}} - 1$. Agora suponha que $n > 2$. Sabemos que $I(n) = 2I(n-2)+1$. Caso n seja par, por hipótese de indução temos que $I(n-2) = 2^{\frac{n-2}{2}} - 1$, pois $n-2$ par. Logo, $I(n) = 2I(n-1)+1 = 2(2^{\frac{n-2}{2}} - 1)+1 = 2^{\frac{n}{2}} - 1$. Caso n seja ímpar, por hipótese de indução temos que $I(n-2) = 2^{\frac{n-1}{2}} - 1$, pois $n-2$ ímpar. Logo, $I(n) = 2I(n-1)+1 = 2(2^{\frac{n-1}{2}} - 1)+1 = 2^{\frac{n+1}{2}} - 1$. ∎

Lema 2.3 *Para $n \geq 2$, vale que $S(n) = 2^{n-1} - 1$.*

Prova. Por indução em n. Suponha que $n = 2$. Temos que $S(n) = 1 = 2^{n-1} - 1$. Agora suponha que $n > 2$. Neste caso $S(n) = 2S(n-1) + 1$. Por hipótese de indução vale que $S(n-1) = 2^{n-2} - 1$. Logo, $S(n) = 2S(n-1) + 1 = 2(2^{n-2} - 1) + 1 = 2^{n-1} - 1$. ∎

Lema 2.4 *Para $n \geq 2$, vale que $I(n) \leq T(n) \leq S(n)$.*

Prova. Por indução em n. Suponha que $n = 2$. Neste caso $I(n) = T(n) = S(n) = 2$. Suponha que $n = 3$. Neste caso $I(n) = T(n) = S(n) = 3$ Agora suponha que $n > 3$. Como a função T monotonicamente crescente, vale que $2T(n-2) \leq T(n-1) + T(n-2) \leq 2T(n-1)$. Por hipótese de indução, vale que $I(n-2) \leq T(n-2)$ e $S(n-1) \geq T(n-1)$. Logo, vale que $2T(n-2)+1 \leq T(n-1)+T(n-2)+1 \leq 2T(n-1)+1 \Rightarrow I(n) \leq T(n) \leq S(n)$.

■

Corolário 2.1 *Vale que* $T(n) = \Omega(2^{\frac{n}{2}})$ *e* $T(n) = O(2^n)$.

Prova. Segue imediatamente dos lemas 2.2, 2.3 e 2.4. ■

Conclusão, o algoritmo `fibonacci` é exponencial! Na verdade $T(n) = \Theta(\phi^n)$ (não provaremos isso – basta olhar a taxa de crescimento da função), sendo que $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618033989$ é a razão áurea.