Sobre Recorrências – Método da Árvore de Recursão

Resolução Exercícios dos Slides – Equações de Recorrência

Prof. M.Sc. Rodrigo Hagstrom
São Paulo – janeiro de 2025

Exercício 1
Resolver a Recursão do Fibonacci (abaixo) usando árvore de recursão:

T(1) = 1
T(2) = 1
T(n) = T(n-1) + T(n-2)

```
           T(n)
          /     \
     T(n-1)   T(n-2)
      /    \     /    \
 T(n-2) T(n-3) T(n-3) T(n-4)
   /  \    /  \    /  \
...
```

Até um certo nível onde restam apenas T(1) e T(2)

O tempo total será a soma dos tempos de todos os níveis (**Chamadas da Função**)

Se chamarmos **nível 0** o nó raiz T(n), podemos observar que:

- **Nível 0**: 1 chamada (T(n))
- **Nível 1**: 2 chamadas (T(n−1),T(n−2))
- **Nível 2**: 4 chamadas (T(n−2),T(n−3),T(n−3),T(n−4))
- **Nível 3**: 8 chamadas
- **Nível k**: 2^k chamadas

A árvore continua se expandindo até que $n-k \le 2$, pois os casos base são T(1) = 1 e T(2) = 1.

Se chamarmos h a altura da árvore (quantidade de níveis), ela é determinada pelo número de subtrações sucessivas necessárias para reduzir n até 2:

$$h \approx n$$

Portanto, a árvore tem **aproximadamente n níveis.**

Sabemos que o número de nós em cada nível segue a progressão geométrica:

Chamadas no nível k = 2^k

O total de **chamadas C(n)** será a soma das chamadas em todos os níveis da árvore, ou seja:

C(n) = ∑ (de k=0 até h) 2^k

Essa é a soma de uma progressão geométrica:

$C(n) = 2^0 + 2^1 + 2^2 + ... + 2^h$

A soma de uma progressão geométrica de razão 2 é dada por:

$C(n) = (2^{(h+1)} - 1) / (2 - 1) = 2^{(h+1)} - 1$

Substituímos $h \approx n$:

$C(n) \approx 2^{(n+1)} - 1$

Para a ordem de complexidade, ignoramos constantes e obtemos:

$C(n) = O(2^n)$

****

Exercício 2

$$T(1)=2$$
$$T(n)=T(n-1)+n, \text{ para } n \geq 2$$

Expandindo os primeiros termos:

$$T(n)=T(n-1)+n$$

$$T(n-1)=T(n-2)+(n-1)$$

$$T(n-2)=T(n-3)+(n-2)$$

Expandindo até chegar em T(1):

$$T(n)=T(n-1)+n$$

$$=(T(n-2)+(n-1))+n$$

$$=((T(n-3)+(n-2))+(n-1))+n$$

$$=(((T(n-4)+(n-3))+(n-2))+(n-1))+n$$

Continuamos até chegar ao caso base T(1)=2.

Cada chamada de T(n) gera **apenas uma chamada recursiva** para T(n−1), diferente do caso de Fibonacci (que gera duas chamadas por nível).

A árvore recursiva tem a seguinte estrutura:

```
    T(n)
     |
 T(n-1) + n
     |
T(n-2) + (n-1)
     |
T(n-3) + (n-2)
     |
    ...
     |
  T(1) + 2
```

Podemos agora reescrever T(n) como uma soma:

$$T(n) = T(1) + 2 + 3 + \ldots + n$$

Sabemos que a soma dos primeiros n números naturais é dada por:

$$\sum(k \text{ de } n \text{ a } 1) = (n(n + 1))/2$$

Substituímos T(1)=2:

$$T(n) = 2 + [((n-1)n)/2]$$

O termo dominante é $n^2/2$, o que implica que a complexidade cresce quadraticamente:

$$T(n) = O(n^2)$$

****

Exercício 3

$$T(1) = 1$$

$$T(n) = T(n - 1) + 3n + 2,\ n \geq 2$$

Expandimos a equação recursiva substituindo valores anteriores:

$$T(n) = T(n-1) + 3n +2$$

$$=(T(n-2) + 3(n-1) + 2) + 3n + 2$$

$$=((T(n-3) + 3(n-2) + 2) + 3(n-1) + 2) + 3n + 2$$

Expandimos até chegar ao caso base T(1) = 1:

$$T(n) = T(1) + k = \sum(\text{com } k \text{ de } 2 \text{ a } n)\ (3k + 2)$$

A árvore recursiva tem uma única chamada descendente por nível:

```
           T(n)
            |
    T(n-1) + (3n + 2)
            |
  T(n-2) + (3(n-1) + 2)
            |
  T(n-3) + (3(n-2) + 2)
            |
           ...
            |
    T(1) + (3(2) + 2)
```

A **altura da árvore** é n−1, pois reduzimos n até alcançar T(1).

Somando os tempos:

Sabemos que:

$$T(n) = T(1) + \sum(\text{com } k \text{ de } 2 \text{ a } n)(3k+2)$$

$$T(n) = 1 + \sum(\text{com } k \text{ de } 2 \text{ a } n)(3k+2)$$

$$T(n) = 1 + \sum(\text{com } k \text{ de } 2 \text{ a } n)3k + \sum(\text{com } k \text{ de } 2 \text{ a } n)2$$

Sabemos que a soma dos primeiros n números naturais é dada por:

$$\sum(\text{com } k \text{ de } 1 \text{ a } n)k = (n(n+1))/2$$

Logo:

$$\sum(\text{com } k \text{ de } 2 \text{ a } n)k = [(n(n+1))/2] - 1$$

Agora o somatório da constante:

$$\sum(\text{com } k \text{ de } 2 \text{ a } n)2 = 2(n-1)$$

Assim:

$$T(n)= 1 + 3\left(\left[\frac{n(n+1)}{2}\right] - 1\right) + 2(n-1)$$

Distribuímos os termos:

$$T(n)= 1 + \frac{3n(n+1)}{2} - 3 + 2n - 2$$
$$T(n) = \frac{3n(n+1)}{2} + 2n - 4$$

O termo dominante é $3n^2/2$ o que implica que a complexidade cresce quadraticamente:

$$T(n) = O(n^2)$$

****

Exercício 4

$$T(1) = 1$$

$$T(n) = 2T(n/2) + n, \text{ para } n \geq 2$$

Expandimos T(n) aplicando a definição recursiva repetidamente:

$$T(n) = 2T(n/2) + n$$

$$=2(2T(n/4) + n/2) + n$$

$$=4T(n/4) + 2(n/2) + n$$

$$=8T(n/8) + 4(n/4) + 2(n/2) + n$$

Expandimos até que o tamanho do problema seja reduzido para T(1), ou seja, até $n/2^k = 1$, o que significa que $k = \log_2 n$.

A recursão forma uma **árvore binária**, onde cada nó gera **duas chamadas recursivas** reduzindo n pela metade.

Níveis da árvore:

- **Nível 0** (raiz): T(n)
- **Nível 1**: 2T(n/2)
- **Nível 2**: 4T(n/4)
- **Nível k**: $2^k T(n/2^k)$

A recursão termina quando $n/2^k=1$, ou seja:

$$2^k = n \Rightarrow k = \log_2 n$$

Portanto, a **altura da árvore** é O(logn).

Agora podemos somar os tempos das chamadas:

$$n + 2(n/2) + 4(n/4) + 8(n/8) + \ldots + 2^{\log 2\ n}\, T(1)$$

Como T(1)=1, a soma total dos custos nos nós internos é:

$$\sum(\text{com k de 0 a } \log_2 n)\ 2^k \cdot (n/2^k)$$

Cada termo dessa soma vale **n**, pois 2^k e $1/2^k$ se cancelam, resultando em:

$$n + n + n + \ldots + n = (\log_2 n + 1) \cdot n$$

A complexidade dominante é:

$$T(n) = O(n \log n)$$