Relações de Recorrência

Relações de Recorrência

Uma **relação de recorrência** (ou apenas **recorrência**) é uma maneira de definir uma função através de uma expressão envolvendo a mesma função.

Exemplo: A sequência de Fibonacci.

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

Relações de Recorrência

É fácil ver que pela recorrência definida anteriormente é possível calcular o número de Fibonacci para qualquer número *k* fazendo *k-2* computações.

É mais conveniente ter uma expressão explícita para *F(n)* e assim rapidamente computar seu valor, e até para comparar com outras funções conhecidas.

Recorrências aparecem sempre em Análise de Algoritmos.

Solucionando a Relação de Recorrência

Encontrar uma função f(n) através de uma recorrência nos ajuda a calcular seu valor e comparar com outra recorrências, isto é chamado **solucionando a relação de recorrência**.

Algumas técnicas para solucionar recorrências são:

- Palpites Inteligentes
- Método de Substituição
- Relações de Divisão e Conquista
- Recorrências com histórico completo

Palpites Inteligentes

Parece não muito científico, mas funciona bem quando estamos tentando **achar um limite superior ao invés da solução exata**.

Exemplo: Considere a recorrência abaixo definida para *n, considerando n* potências de 2.

$T(2) = 1$ se n=2; e

$T(2n) \leq 2T(n) + 2n - 1$ se n>2

Palpites Inteligentes

Queremos encontrar um limite superior (na forma da notação O) para *T*; ou seja, nosso objetivo é encontrar uma função *f(n)* tal que T(n) = O(f(n)).

Mas temos que ter o cuidado para que esse limite superior não seja muito distante de do valor exato de T(n).

Palpites Inteligentes

(Chute) Vamos supor então que $f(n) = n^{2.}$

Agora é preciso Provar que $T(n) = O (f(n))$.

Usando Indução matemática temos que:

Passo 1: Verificar a base da indução: $T(2) = 1$.

$$T(n) \leq f(n)$$
$$T(2) \leq f(2)$$
$$1 \leq 2^2$$
$$1 \leq 4$$

Palpites Inteligentes

Passo 2: Assumindo como hipótese de indução:

$$T(n) \leq f(n) \Rightarrow T(2n) \leq (2n)^2$$

Passo3: Prova:

$T(2n) \leq 2T(n) + 2n + 1$ (pela definição da recorrência)

$\leq 2f(n) + 2n + 1$

$\leq 2n^2 + 2n + 1$ (pela hipótese de indução)

$\leq 2n^2 + 2n + 1 < (2n)^2$

$\leq 2n^2 + 2n + 1 < 4n^2$

Verdadeiro p/todo $n>2$.

Palpites Inteligentes

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

É um bom palpite ???

Palpites Inteligentes

Vamos tentar $f(n) = cn$, para uma constante c.

1. Base da Indução:

$$T(2) \leq f(2)$$

$$T(2) = 1 \leq f(2) = 2c$$

2. Hipótese de Indução:

$$T(n) \leq f(n) \Rightarrow T(2n) \leq f(2n)$$

3. Prova:

$$T(2n) \leq 2T(n) + 2n - 1$$

$$\leq 2cn + 2n - 1$$

$$\leq 2cn + 2n - 1 \leq f(2n)$$

$$2cn + 2n - 1 > 2cn \quad \text{falso.}$$

logo $O(n) < T(n) < O(n^2)$

Palpites Inteligentes

Vamos tentar $f(n) = n.\log_2 n$.

1. Base da Indução:

$$T(2) \leq f(2)$$

$$T(2) = 1 \leq f(2) = 2(\log_2 2)$$

$$T(2) = 1 \leq f(2) = 2$$

2. Hipótese de Indução:

$$T(n) \leq f(n) \Rightarrow T(2n) \leq f(2n)$$

3. Prova:

$$T(2n) \leq 2T(n) + 2n - 1$$

$$\leq 2n \log_2 n + 2n - 1$$

$$\leq 2n \log_2 n + 2n - 1 \leq f(2n)$$

$$\leq 2n \log_2 n + 2n - 1 \leq 2n(\log_2 2n) \text{ (prop. logaritmo)}$$

$$\leq 2n \log_2 n + 2n - 1 \leq 2n(\log_2 n + \log_2 2)$$

$$\leq 2n \log_2 n + 2n - 1 \leq 2n.\log_2 n + 2n \quad \text{verdadeiro}$$

logo temos que $T(n) = O(n \log_2 n)$

Árvore de Recursão

Ao usar uma árvore de recursão para gerar um bom palpite, muitas vezes você pode tolerar uma pequena quantidade de “desleixo”, já que verificaremos o palpite mais tarde.

Se você for muito cuidadoso ao desenhar uma árvore de recursão e somar os custos, você pode usar uma árvore de recursão como uma prova direta de uma solução para uma recorrência. Faremos isso na prova do teorema mestre.

Árvore de Recursão

- Exemplo: Seja a recorrência

$$T(n) = 3T(n/4)+\Theta(n^2)$$

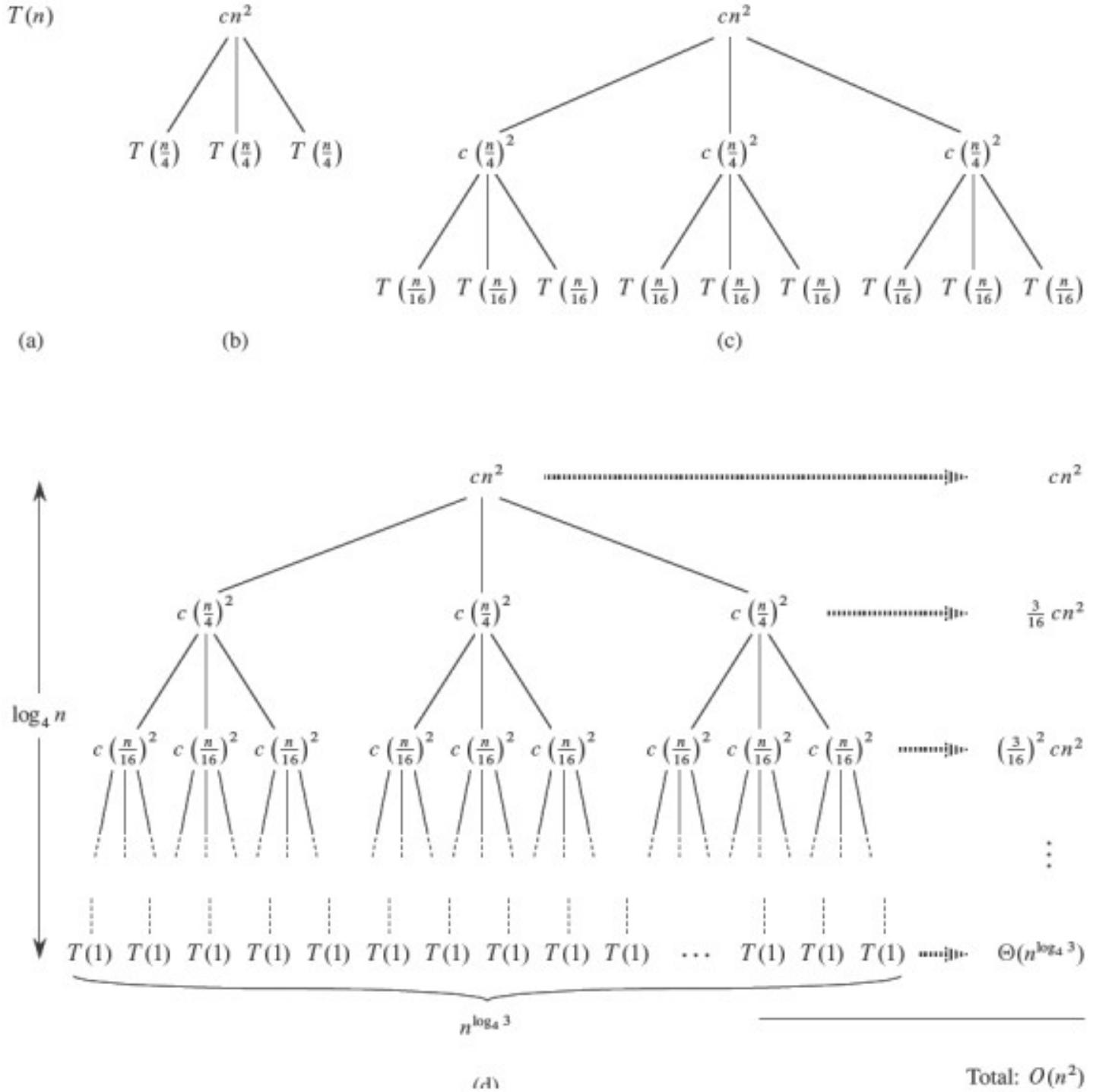
Buscamos um limite superior para essa recorrência, então criamos uma árvore de recursão para a recorrência

$$T(n) = 3T(n/4)+cn^2, \text{ para } c>0$$

Por conveniência, vamos assumir que n é uma potência de 4 exata(exemplo de desleixo permitido), para tornar os tamanhos subproblemas inteiros.

Árvore de Recursão

- A Figura 1 mostra como derivamos a árvore de recursão da recorrência.


T(n)
cn²
T(n/4) T(n/4) T(n/4)
c(n/4)²
T(n/16)
(a)
(b)
(c)
log₄ n
c(n/16)²
3/16 cn²
(3/16)² cn²
T(1)
Θ(n^log₄ 3)
n^log₄ 3
Total: O(n²)

Palpites Inteligentes
Árvore de Recursão

Para tentar definir um palpite mais próximo da complexidade da recorrência pode-se utilizar uma árvore de recursão a partir da recorrência.

- Em uma árvore de recursão cada nó representa o custo de um único subproblema em algum lugar no conjunto de invocações de funções recursivas.
- Somamos os custos dentro de cada nível da árvore para obter um conjunto de custos por nível e, em seguida, somamos todos os custos por nível para determinar o custo total de todos os níveis da recursão.