PANC: Projeto e Análise de Algoritmos

Aula 07a: Resolução de Recorrências

Breno Lisi Romano

Semana: 16 a 22 de Agosto de 2020

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Sumário

- Revisão de Conteúdo
- Introdução a Resolução de Recorrências
- Resolução de Recorrências:
  - Método de Substituição
  - Método da Iteração
  - Método da Árvore de Recursão
  - Teorema Mestre (Próxima Aula)
- Exercícios Práticos

Recapitulando... (1)

- **Algoritmos Recursivos:**
  - Quando uma **função invoca a si própria**
  - A ideia é **aproveitar a solução** de um ou mais subproblemas com **estrutura semelhante para resolver o problema original**
  - Geralmente adota-se a **recursividade** para **auxiliar na aplicação do paradigma de Divisão e Conquista**
  - **Princípio:** Parte-se da hipótese de que a solução para um problema de tamanho t pode ser obtida a partir da solução para o mesmo problema, porém, de tamanho t−1 (ou outra fração do problema)
  - **Projeto:** Um algoritmo recursivo é composto, em sua forma mais simples, de uma **condição de parada** e de um **passo recursivo**

Recapitulando... (2)

- **Recorrências:**
  - **Uma recorrência é uma equação ou desigualdade que descreve uma função em termos de seu próprio valor em entradas menores**
  - **Aplicação e Resolução:**
    - A **complexidade de algoritmos recursivos** pode ser frequentemente descrita através de **recorrências**
    - Geralmente, recorremos ao **Teorema Mestre** para **resolver** estas **recorrências**
    - Em **casos** em que o **Teorema Mestre não se aplica**, a **recorrência** deve ser **resolvida** de **outras maneiras**
    - **Resolver** uma **recorrência significa eliminar** as **referências** que **ela faz a si mesma**
    - **Três dos métodos mais comuns** para resolução de recorrências são o método de **substituição**, o método de **árvore de recursão** e o **teorema mestre**

Resolução de Recorrências (1)

- Por exemplo, o T(n) do **pior caso do Algoritmo do MergeSort()** pela **recorrência** cuja solução afirmamos ser **T(n) = Θ(n lg n)** é:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n) & \text{se } n > 1 \end{cases}$$

- **Queremos resolver esta fórmula!!!**
- Porém, as **recorrências** podem tomar muitas formas:
  - Um **algoritmo** recursivo poderia **dividir problemas em tamanhos desiguais**, como uma subdivisão 2/3 para 1/3. Se as etapas de divisão e combinação levarem tempo linear, tal algoritmo dará origem à recorrência **T(n) = T (2n/3) + T(n/3) + Θ(n)**
  - Os **subproblemas não** estão necessariamente restritos a ser **uma fração constante do tamanho do problema original**
    - Por exemplo, uma versão recursiva da busca linear criaria apenas um subproblema contendo somente um elemento a menos do que o problema original. Cada chamada recursiva levaria tempo constante mais o tempo das chamadas recursivas que fizer, o que produz a recorrência **T(n) = T(n - 1) + Θ(1).**

Resolução de Recorrências (2)

- Existem alguns **métodos para resolver recorrências**, isto é, para **obter limites assintóticos "Θ" ou "O" para a solução**. Os principais são:
  - **Método de substituição:** arrisca-se um palpite para um limite e então utiliza-se da indução matemática para provar que nosso palpite estava correto
  - **Método da Iteração:** expandir (iterar) a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n e das condições iniciais
  - **Método da árvore de recursão:** converte a recorrência em uma árvore cujos nós representam os custos envolvidos em vários níveis da recursão. Usa-se técnicas para limitar somatórios, resolvendo-se a recorrência;
  - **Método do Teorema mestre**: dá limites para recorrências da forma **$T(n) = aT(n/b) + f(n)$**, onde $a \geq 1$, $b > 1$ e $f(n)$ é uma função dada. Tais recorrências ocorrem frequentemente.
    - Uma recorrência da forma da equação apresentada caracteriza um algoritmo de divisão e conquista que cria **a subproblemas**, cada um **com 1/b do tamanho do problema original** e no qual as etapas de divisão e conquista, juntas, levam o **tempo f(n)**

Método de Substituição

- O **método de substituição** para resolver recorrências envolve duas etapas:
  1. **Arriscar um palpite** para a forma da solução
  2. **Usar indução** para **determinar as constantes** e mostrar que a solução funciona

- **Substitui-se a função pela solução suposta na primeira etapa quando aplica-se a hipótese indutiva a valores menores**; daí o nome "método de substituição"

- **Método é poderoso**, mas tem que adivinhar a forma da resposta para aplicá-lo

- **Aplica-se** este método **em casos que é fácil pressupor** a forma de resposta

Método de Substituição – Exemplo 01

- **Recorrência do Pior Caso do Algoritmo do MergeSort():**

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n) & \text{se } n > 1 \end{cases}$$

- **Resolução da Recorrência:**
  - Vamos supor a recorrência tem limite superior igual a **n.lgn**, ou seja, **T(n) = O(n.lgn)**
  - Deve-se provar que **T(n) ≤ c.n.lgn** para uma escolha apropriada da constante c > 0, temos:

$$\mathrm{T(n)} \leq 2(c\left\lfloor \frac{n}{2} \right\rfloor . (\lg\left(\left\lfloor \frac{n}{2} \right\rfloor\right)) + n)$$

$$\mathrm{T(n)} \leq cn.\lg\left(\frac{n}{2}\right) + n) = cn.\lg(n) \boxed{-cn+n}$$

Resíduo

$$\mathrm{T(n)} \leq cn.\lg(n)$$

  - para c ≥ 1.
  - A expressão **−cn + n** é resíduo. A prova objetiva mostra que o resíduo é negativo.
- **Portanto, está provado!**

Método de Substituição – Exemplo 02

- **Recorrência:**

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lfloor n/2 \rfloor + 17) + n \quad \text{para } n \geq 2 \end{aligned}$$

- **Resolução da Recorrência:**
  - Ela parece bem mais difícil por causa do "17" no lado direito
  - Intuitivamente, porém, isto não deveria afetar a solução. Para n grande a diferença entre $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ e $T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right)$ não é tão grande (ambos cortam n quase uniformemente pela metade!
  - Chuta-se então que T(n) = O(n.lgn)
  - Pode-se verificar que é correto usando o método de substituição! **(Exercício para casa)**

Método da Iteração (1)

- Não é necessário adivinhar a resposta para a recorrência
- Precisa-se fazer mais conta → Mão na massa
- **Ideia**: expandir (iterar) a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n e das condições iniciais
  - Existem **alguns truques** que podem ser aplicados:
    - **Procurar por algum padrão ao expandir uma recorrência**, como alguma recorrência básica
    - **Realizar manipulações algébricas**, como troca de variáveis ou divisão da recorrência, que favoreçam a resolução
- Para tanto, é necessário ter **conhecimento algébrico**, de recorrências básicas e uma **dose de "maldade"**

Método da Iteração (2)

- **Alguns Somatórios Úteis:**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1} - 1}{a - 1} \quad (a \neq 1)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Método da Iteração (3)

- A ideia da resolução pelo **método da iteração (ou expansão telescópica)** é **expandir a relação de recorrência** até que possa ser **detectado seu comportamento no caso geral**
- **Passos para resolver:**
  1. Copie a fórmula original
  2. Descubra o passo (se T(n) estiver escrito em função de T(n/2), a cada passo o parâmetro é dividido por 2)
  3. Isole as equações para “os próximos passos”
  4. Substitua os valores isolados na formula original
  5. Identifique a fórmula do i-ésimo passo
  6. Descubra o valor de i de forma a igualar o parâmetro de T(x) ao parâmetro (valor de n) no caso base
  7. Substitua o valor de i na fórmula do i-ésimo caso
  8. Identifique a complexidade dessa fórmula
  9. Prove por indução que a equação foi corretamente encontrada

Método da Iteração – Exemplo 1 (1)

- **Recorrência:**
  - **$T(n) = 2T(n/2) + 2$**
  - **$T(1) = 1$**
- **Resolução da Recorrência:**
  1. $T(n) = 2\,T(n/2) + 2$ (Fórmula Original)
  2. $T(n)$ está escrito em função de $T(n/2)$
  3. Isole as equações para $T(n/2)$ e $T(n/4)$
     - $T(n/2) = 2(T(n/4)) + 2$
     - $T(n/4) = 2(T(n/8)) + 2$
  4. Substitua $T(n/2)$ pelo valor que foi isolado acima e, em seguida, faça o mesmo para $T(n/4)$
     - Substituindo valor isolado de $T(n/2)$
       – $T(n) = 2(2(T(n/4)) + 2) + 2 \rightarrow 2^2\,T(n/2^2) + 2^3 - 2$
     - Substituindo valor de $T(n/4)$
       – $T(n) = 2^2\,(2(T(n/8)) + 2) + 6 \rightarrow 2^3\,T(n/2^3) + 2^3 + 6 \rightarrow 2^3\,T(n/2^3) + 2^4 - 2$

Método da Iteração – Exemplo 1 (2)

- **Resolução da Recorrência:**

5. Identifique a fórmula do i-ésimo passo
   - $T(n) = 2^i T(n/2^i) + 2^{i+1} - 2$
6. Descubra o valor de i de forma a igualar o parâmetro de T(x) ao parâmetro (valor de n) no caso base
   - T(n/2i) = T(1)
   - n/2i = 1
   - n = 2i
   - i = lg (n)
7. Substitua o valor de i na fórmula do i-ésimo caso
   - $T(n) = 2^{lg(n)} T(1) + 2^{lg(n)+1} - 2$
   - T(n) = n + 2n – 2
   - T(n) = 3n – 2
8. Identifique a complexidade dessa fórmula
   - **$T(n) \in \Theta(n)$**

Método da Iteração – Exemplo 1 (3)

- **Resolução da Recorrência:**
  - Prova por Indução de **$T(n) = 3n – 2$**, lembrando-se que **$T(n) = 2T(n/2) +2$ e $T(1) = 1$**
    - **Passo base:** para $n = 1$, o resultado esperado é 1
      - $T(n) = 3n − 2 = 3 − 2 = 1$ **(correto)**
    - **Passo indutivo:**
      - Por Hipótese de Indução (H.I), assume-se que a fórmula está correta para $n/2$, isto é, $T(n/2) = 3n/2 − 2$
      - Então, tem-se que verificar se $T(n) = 3n-2$, sabendo-se que $T(n) = 2T(n/2) + 2$ e partindo da H.I que
        - » $T(n/2) = 3n/2 − 2$
        - » $T(n) = 2\ T(n/2) +2$
        - » $T(n) = 2\ (3n/2 − 2) +2$
        - » $T(n) = 2.\ 3n/2 – 2.2 + 2$
        - » $T(n) = 3n − 4 + 2$
        - » **$T(n) = 3n − 2$ (passo indutivo provado)**
    - **Demonstrado que $2T(n/2) + 2 = 3n − 2$ para $n \geq 1$**

Método da Iteração – Exemplo 2 (1)

- **Recorrência – Problema da Torre de Hanoi:**
  - **$T(n) = 2\,T(n - 1) + 1$**
  - **$T(1) = 1$**
- **Resolução da Recorrência:**
  1. $T(n) = 2\,T(n - 1) + 1$ (Fórmula Original)
  2. $T(n)$ está escrito em função de $T(n - 1)$
  3. Isole as equações para $T(n - 1)$ e $T(n - 2)$
     - $T(n - 1) = 2\,T(n - 2) + 1$
     - $T(n - 2) = 2\,T(n - 3) + 1$
  4. Substitua $T(n - 1)$ pelo valor que foi isolado acima e, em seguida, faça o mesmo para $T(n - 2)$
     - Substituindo valor isolado de $T(n - 1)$
       - $T(n) = 2(2\,T(n - 2) + 1) + 1$
     - Substituindo valor de $T(n - 2)$
       - $T(n) = 2^2\,T(n - 2) + 2 + 1 \rightarrow T(n) = 2^2\,(2\,T(n - 3) + 1) + 2 + 1 \rightarrow$
       - $T(n) = 2^3\,T(n - 3) + 2^2 + 2 + 1 \rightarrow T(n) = 2^3\,T(n - 3) + 2^3 - 1$

Método da Iteração – Exemplo 2 (2)

- **Resolução da Recorrência:**

5. Identifique a fórmula do i-ésimo passo
   - $T(n) = 2^i T(n - i) + 2^i - 1$
6. Descubra o valor de i de forma a igualar o parâmetro de T(x) ao parâmetro (valor de n) no caso base
   - $T(n - i) = T(1)$
   - $n - i = 1$
   - $i = n - 1$
7. Substitua o valor de i na fórmula do i-ésimo caso
   - $T(n) = 2^{n-1} T(1) + 2^{n-1} - 1$
   - $T(n) = 2^{n-1} + 2^{n-1} - 1$
   - $T(n) = 2.2^{n-1} – 1$
   - $T(n) = 2^n – 1$
8. Identifique a complexidade dessa fórmula
   - **$T(n) \in \Theta(2^n)$**

Método da Iteração – Exemplo 2 (3)

- **Resolução da Recorrência:**
  - Prova por Indução de **$T(n) = 2^n – 1$**, lembrando-se que **$T(n) = 2\ T(n - 1) + 1$ e $T(1) = 1$**
    - **Passo base:** para n = 1, o resultado esperado é 1
      - $T(n) = 2^n – 1 = 2 – 1 = 1$ **(correto)**
    - **Passo indutivo:**
      - Por Hipótese de Indução (H.I), assume-se que a fórmula está correta para n – 1, isto é, $T(n - 1) = 2^{n-1} – 1$
      - Então, tem-se que verificar se $T(n) = 2^n – 1$, sabendo-se que $T(n) = 2^n – 1$ e partindo da H.I que $T(n - 1) = 2^{n-1} – 1$
        - » $T(n) = 2\ T(n - 1) + 1$
        - » $T(n) = 2\ (2^{n-1} – 1) + 1$
        - » $T(n) = 2^n – 2 + 1$
        - » **$T(n) = 2^n - 1$ (passo indutivo provado)**
    - **Demonstrado que $2T(n - 1) + 1 = 2^n − 1$ para $n \geq 1$**

Método da Árvore de Recursão (1)

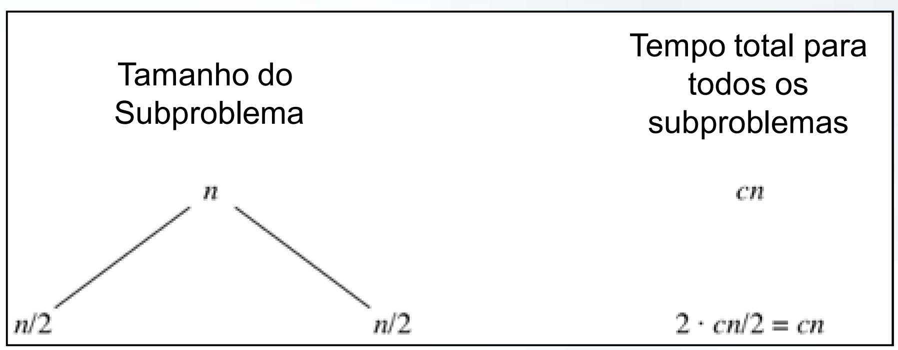
- Uma **árvore de recursão** apresenta uma **forma bem intuitiva** para a **análise de complexidade** de algoritmos **recursivos**
  - Numa árvore de recursão, **cada nó representa o custo de um único subproblema** da respectiva chamada recursiva
  - **Somam-se os custos de todos os nós** de um mesmo nível, para obter o custo daquele nível
  - **Somam-se os custos de todos os níveis** para obter o custo da árvore
- Permite visualizar melhor o que acontece quando a recorrência é iterada
- É mais fácil organizar as contas
- **Útil** para recorrências de **algoritmos de divisão e conquista** → Por isto, apresentamos no estudo do MergeSort ()

Método da Árvore de Recursão – Exemplo 01 (1)

- **Fórmula de Recorrência – Mergesort():**
  - **T(1) = c** se n=1
  - **T(n) = 2.T(n/2) + c.n** se n>1

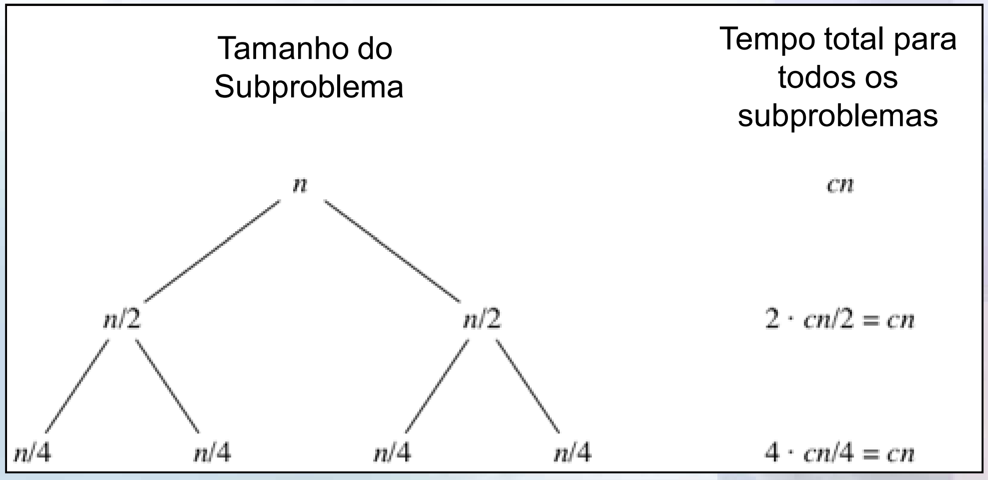
  onde c é um a constante para o tempo exigido em resolver problemas de tamanho 1, bem como o tempo por elemento do (sub)array para as etapas de dividir e combinar

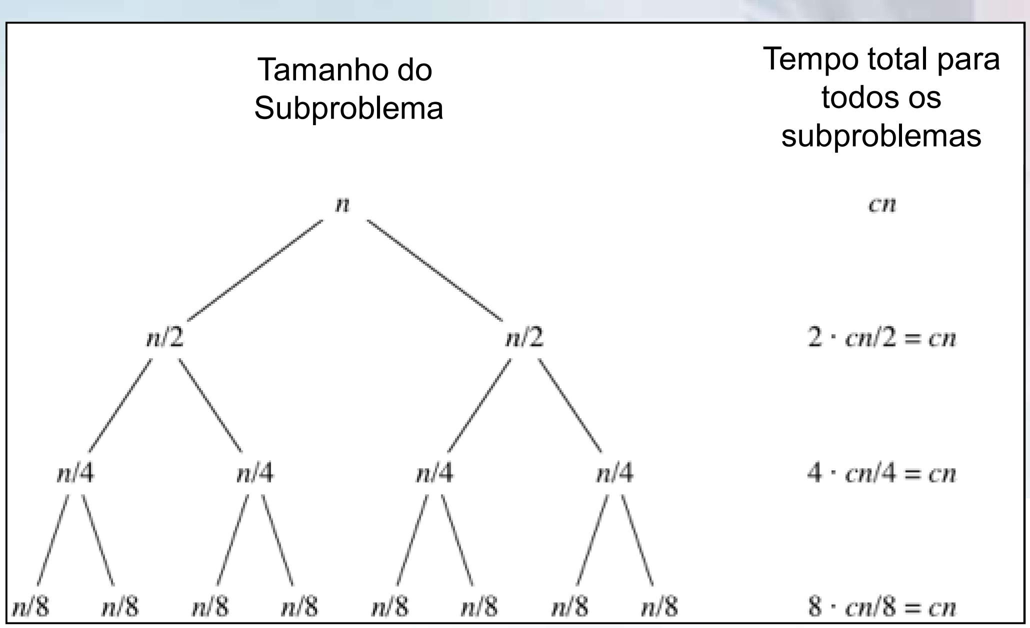
- **Resolução da Recorrência:**
  - Vamos resolver com uma "Árvore de Recorrência":



Método da Árvore de Recursão – Exemplo 01 (2)

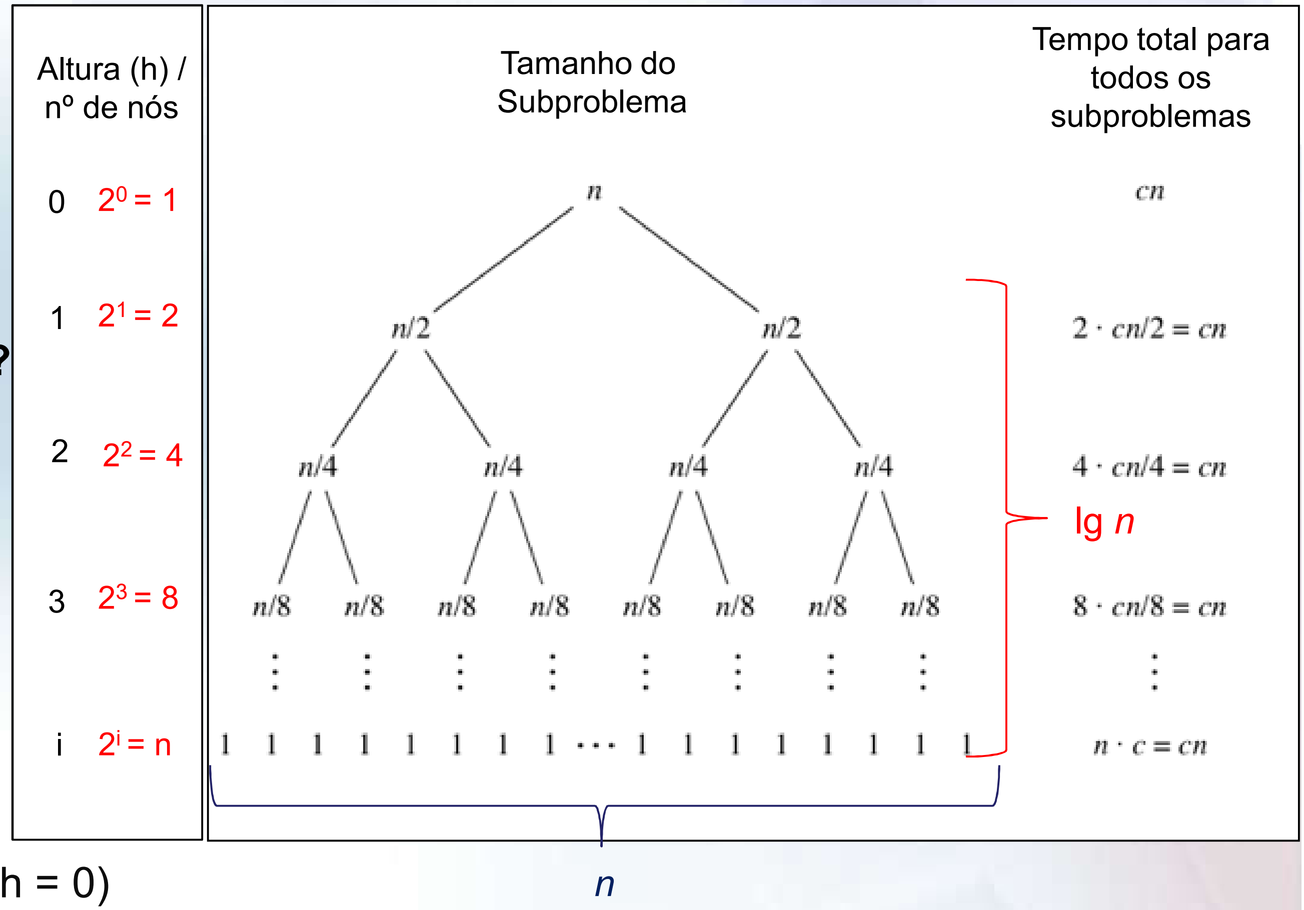
- **Fórmula de Recorrência:**
  - $T(1) = c$
  - $T(n) = 2.T(n/2) + c.n$





Método da Árvore de Recursão – Exemplo 01 (3)

- **Quantos nós tem na árvore com a altura i?**
  - n elementos
- **E qual a relação com a altura da árvore?**
  - $2^i = n$
  - $\lg 2^i = \lg n$
  - $i = \lg n$ ($\lg n = \log_2 n$)
- **Quantos níveis tem a árvore?**
  - lg n + 1 (Obs.: +1 → h = 0)



- **Complexidade T(n):**
  - c.n . (lg n + 1) → T(n) = c.n lg n + c.n → **T(n) = O(n.lg n)**

Método da Árvore de Recursão – Exemplo 02 (1)

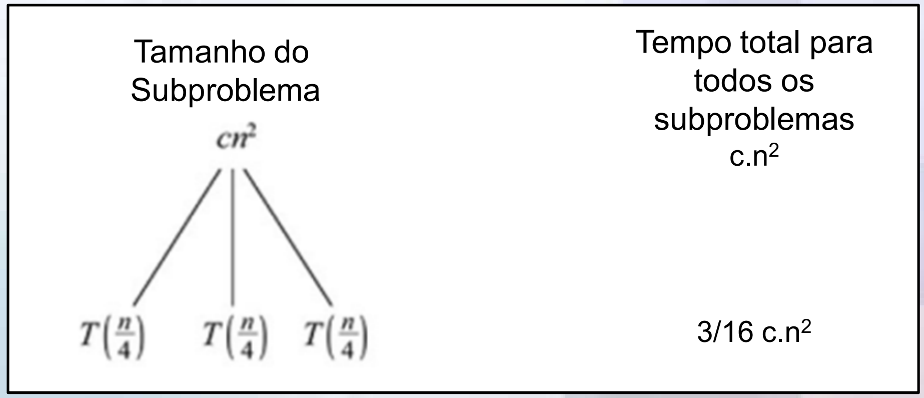
- **Fórmula de Recorrência:**
  - **$T(n) = \Theta(1)$** se n = 1, 2 e 3
  - **$T(n) = 3.T(n/4) + c.n^2$** se $n \geq 1$

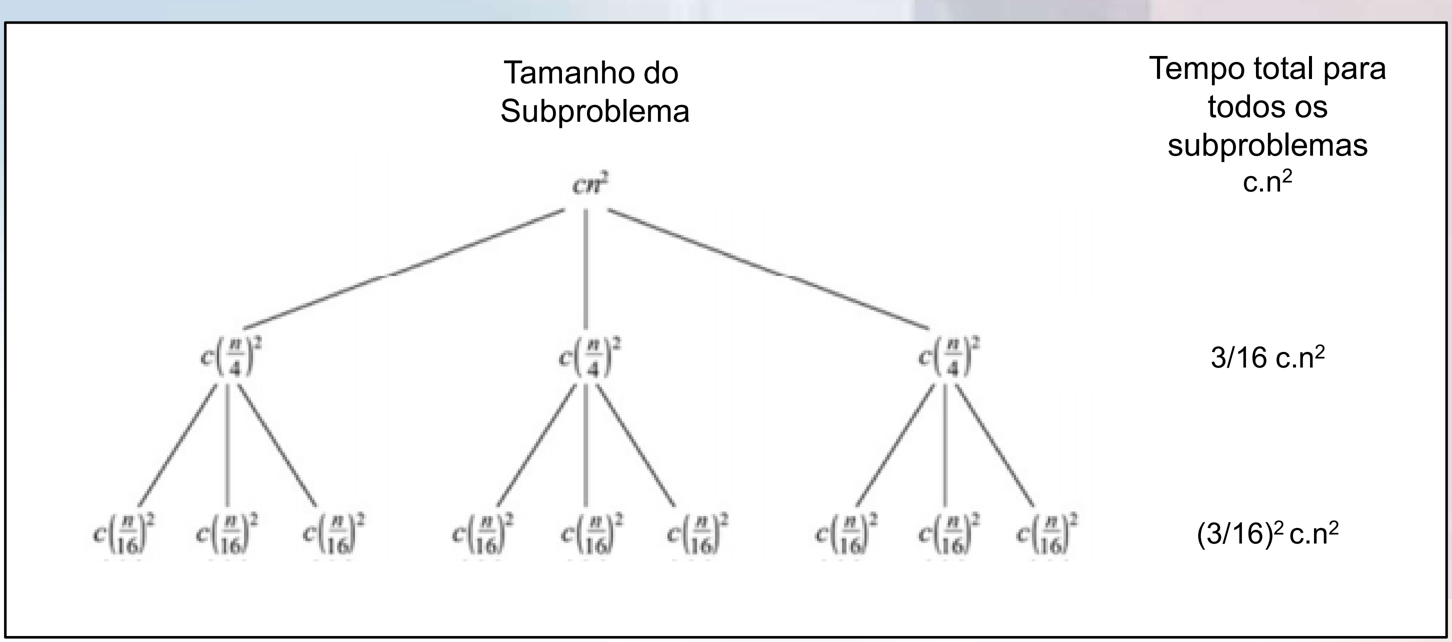
  onde c é um a constante e $T(n) = \Theta(1)$ para indicar uma constante.

- **Resolução da Recorrência:**
  - Vamos resolver com uma “Árvore de Recorrência”:
  - Por conveniência, supomos que n é uma potência exata de 4 (outro exemplo de desleixo tolerável) de modo que os tamanhos de todos os subproblemas são inteiros.

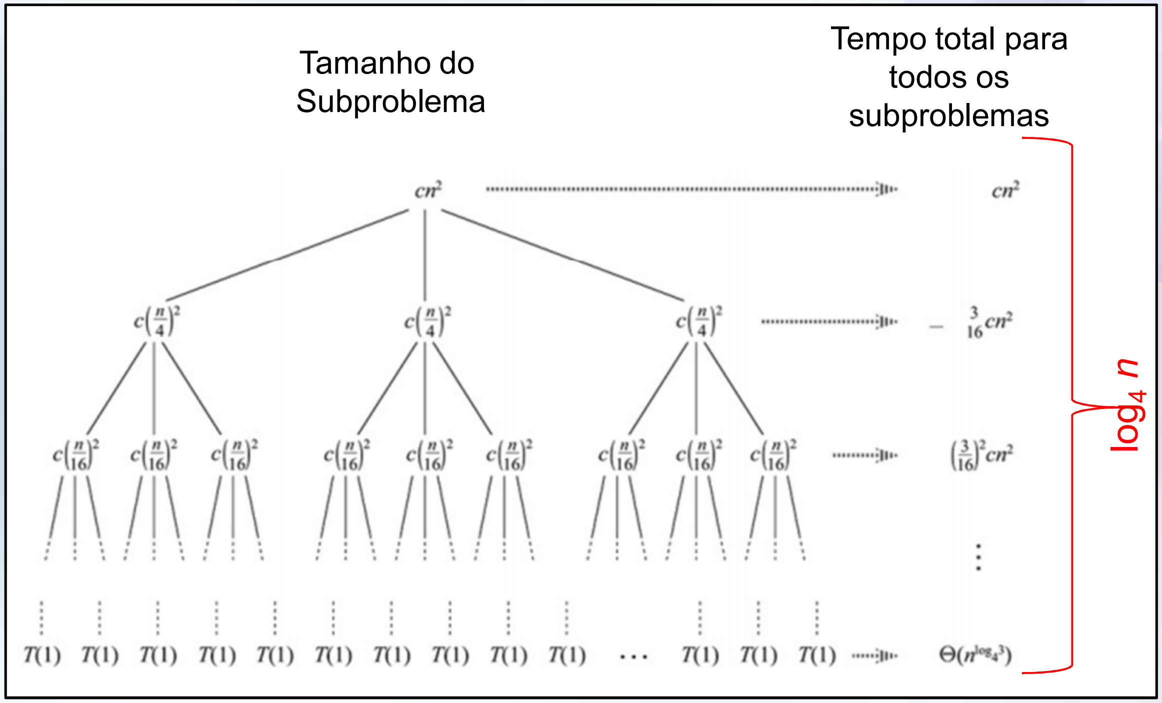
Método da Árvore de Recursão – Exemplo 02 (2)

- **Fórmula de Recorrência:**
  - **T(n) = Θ(1) ,** se n = 1, 2 e 3
  - **T(n) = 3.T(n/4) + c.n²,** se n≥1



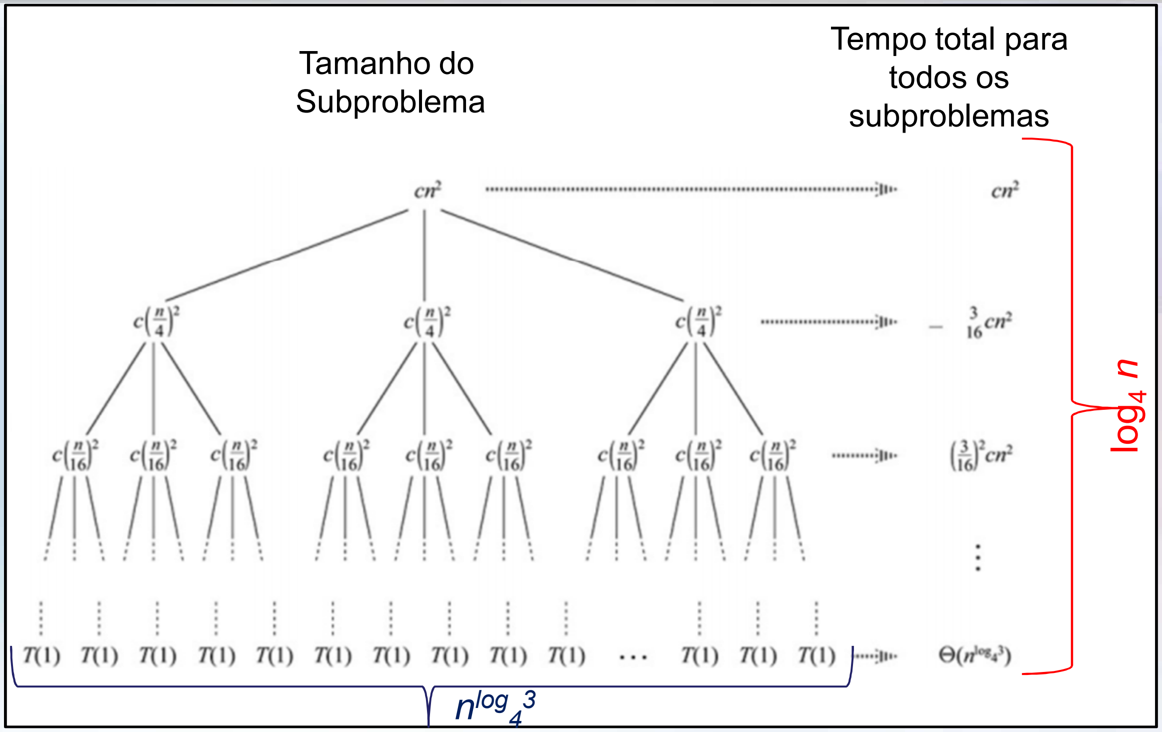


Método da Árvore de Recursão – Exemplo 02 (3)



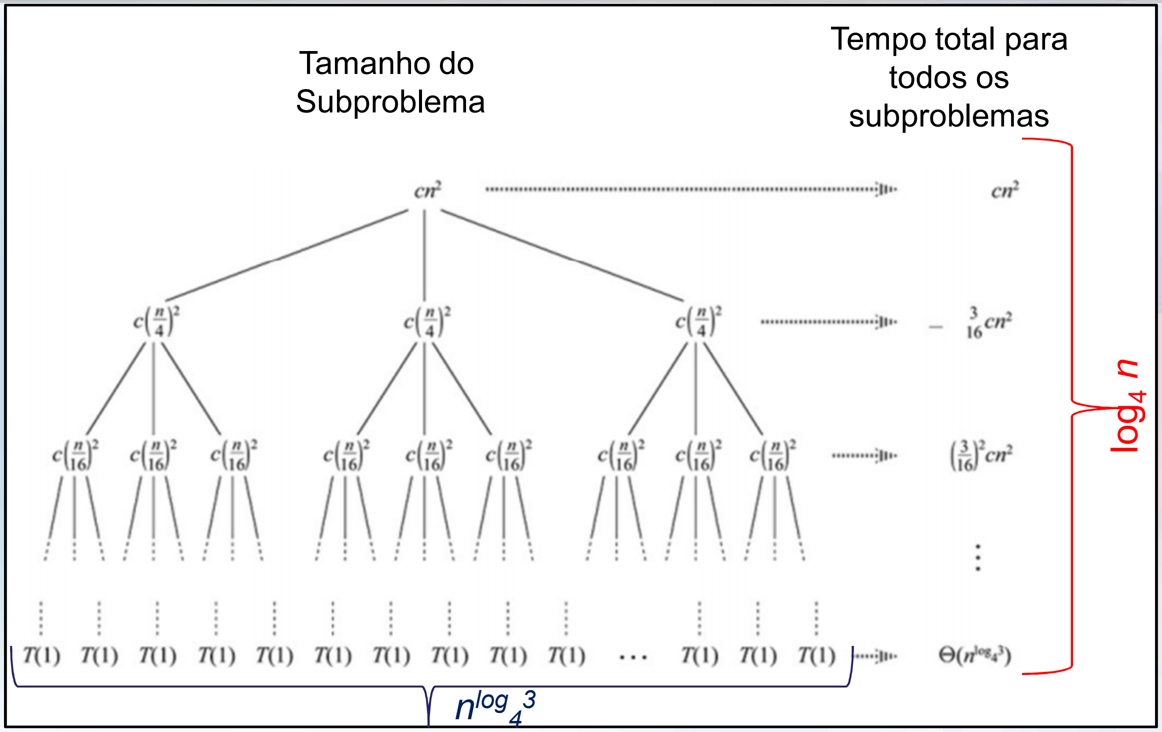
- Os tamanhos dos subproblemas diminuem por um fator de 4 toda vez que descemos um nível, a certa altura devemos alcançar uma condição de contorno
  - A que distância da raiz nós a encontramos? O tamanho do subproblema para um nó na profundidade i é n/4*i*. Desse modo, o tamanho do subproblema chega a n = 1 quando $n/4^i=1$ ou, o que é equivalente, quando $i = \log_4 n$
  - A árvore tem **$\log_4$ n+1 níveis (nas profundidades 0, 1, 2, ... , log4 n)**

Método da Árvore de Recursão – Exemplo 02 (4)



- Determinar o custo em cada nível da árvore
  - Cada nível tem três vezes mais nós que o nível acima dele, portanto o número de nós na profundidade i é 3^i
  - Como os tamanhos dos subproblemas se reduzem por um fator de 4 para cada nível que descemos a partir da raiz, cada nó na profundidade i, para i = 0, 1, 2, ... , $\log_4 n-1$, tem o custo de $c(n/4^i)^2$
  - Multiplicando, o custo total para todos os nós na profundidade i, para i = 0, 1, 2, ... , log4 n-1, é $3^i c(n/4^i)^2 = (3/16)^i c.n^2$
  - O nível inferior, na profundidade $\log_4 n$, tem $3^{\log_4 n} = n^{\log_4 3}$ nós, e cada um deles contribui com o custo T(1), para um custo total de $n^{\log_4 3}T(1)$, que é $\Theta(n^{\log_4 3})$, já que supomos que T(1) é uma constante

Método da Árvore de Recursão – Exemplo 02 (4)



- Agora somamos os custos em todos os níveis para determinar o custo da árvore inteira:
  - **Complexidade T(n): O(n²)**
    - Esta última fórmula parece um pouco confusa até percebermos que, mais uma vez, é possível tirar proveito de uma certo desleixo e usar uma série geométrica decrescente infinita como um limite superior.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)}cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13}cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2).
\end{aligned}
$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

série geométrica decrescente infinita

Método da Árvore de Recursão – Exemplo 02 (5)

- Pode-se utilizar **o método de substituição** para **verificar** que nosso está correto, isto é, **$T(n) = O(n^2)$** é um **limite** superior para a recorrência **$T(n) = 3T(n/4)+\Theta(n^2)$**
- Queremos mostrar que **$T(n) \leq dn^2$** para alguma constante $d > 0$. Usando a mesma constante $c > 0$ de antes, temos onde a última etapa é válida desde que $d \geq (16/13)c$

$$\begin{aligned} T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2 \end{aligned}$$

Provado com sucesso!

Método da Árvore de Recursão - Resumo

- **Passo a Passo:**
  - Desenha a árvore de recursão
  - Determine:
    - O número de níveis
    - O número de nós e o custo por nível
    - O número de folhas
  - Soma-se os custos dos níveis e o custo das folhas
  - (Eventualmente) Verifica utilizando-se o método de substituição
- **Resumo:**
  - O número de nós em cada nível da árvore é o número de chamadas recursivas
  - Em cada nó, indica-se o “tempo/trabalho” gasto naquele nó que não corresponde a chamadas recursivas
  - Na coluna mais á direita indica-se o tempo total naquele nível que não corresponde a chamadas recursivas
  - Somando ao longo da coluna, determina-se a solução da recorrência

Trabalhos para Casa (1)

- **Exercício 01: Refazer todas as resoluções de recorrências apresentadas nesta aula e entregar no Google Classroom (em um único arquivo):**
  a) 02 Recorrências de Substituição
  b) 02 Recorrências de Iteração
  c) 02 Recorrências da Árvore de Recursão

Trabalhos para Casa (2)

- **Exercício 02: Aplicar o Método de Substituição para Resolver as Recorrências a seguir e entregar no Google Classroom em um único arquivo:**

  a) $T(1) = 0$ / $T(n) = T(n/2) + 1$ (para $n \geq 2$)
     - Supunha-se que $n = 2^k \rightarrow k = \lg n$

  b) $T(1) = 0$ / $T(n) = 2T(n/2) + n$ (para $n > 1$)
     - Supunha-se que $n = 2k \rightarrow k = \lg n$

Trabalhos para Casa (3)

- **Exercício 03: Aplicar os 09 passos do Método Iterativo (Slide 9) para Resolver as Recorrências a seguir e entregar no Google Classroom em um único arquivo:**

a) $T(1) = 1$ / $T(n) = 3T(n - 1) + 1$

b) $T(1) = 1$ / $T(n) = 4T(n/2) + n$

Trabalhos para Casa (4)

- **Exercício 04: Aplicar o Método de Árvore de Recursão para Resolver a Recorrência a seguir e entregar no Google Classroom em um único arquivo:**

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

PANC: Projeto e Análise de Algoritmos

Aula 07: Resolução de Recorrências

Breno Lisi Romano

Dúvidas???

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista