Análise e Técnicas de Algoritmos

Jorge Figueiredo

Análise de Algoritmos Recursivos

Agenda

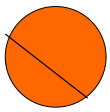
- Relação de Recorrência
  - Derivando recorrência
  - Resolvendo recorrência
- Análise de algoritmos recursivos

Introdução

- A análise de um algoritmo recursivo requer a resolução de uma recorrência.
- Uma recorrência é um algoritmo recursivo que calcula o valor de uma função em um ponto dado.
- Uma recorrência define T(n) em termos de T(n-1), T(n-2), etc.
- Exemplo:
  - T(1) = 1
  - T(n) = T(n – 1) + 3n + 2 , para n≥2

Introdução

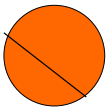
- Exemplo 2: Quantos pedaços com n cortes?



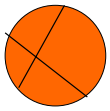
Cortes: 1
Pedaços: 2

Introdução

- Exemplo 2: Quantos pedaços com n cortes?



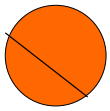
Cortes: 1
Pedaços: 2



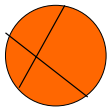
Cortes: 2
Pedaços: 4

Introdução

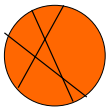
- Exemplo 2: Quantos pedaços com n cortes?



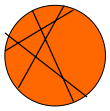
Cortes: 1
Pedaços: 2



Cortes: 2
Pedaços: 4



Cortes: 3
Pedaços: 7



Cortes: 4
Pedaços: 11

Introdução

- É possível observar que o n-ésimo corte cria n novos pedaços.
- Logo, o número total de pedaços obtido com n cortes, denotado por P(n), é dado pela seguinte relação de recorrência:
  - P(1) = 2
  - P(n) = P(n – 1) + n, para n≥2

Derivando Relações de Recorrências

Como proceder para derivar uma relação de recorrência para a análise do tempo de execução de um algoritmo:

- Determinar qual o tamanho n do problema.
- Verificar que valor de n é usado como base da recursão. Em geral é um valor único (n=1, por exemplo), mas pode ser valores múltiplos. Vamos considerar esse valor como n_0.
- Determinar $T(n_0)$. Pode-se usar uma constante c, mas, em muitos, casos um número específico é necessário.

Derivando Relações de Recorrências

- T(n) é definido como uma soma de várias ocorrências de T(m) (chamadas recursivas), mais a soma de outras instruções efetuadas. Em geral, as chamadas recursivas estão relacionadas com a subproblemas do mesmo tamanho f(n), definindo um termo a.T(f(n)) na relação de recorrência.
- A relação de recorrência é definida por:
  - T(n) = c, se $n = n_0$
  - T(n) = a.T(f(n)) + g(n), caso contrário

Derivando Relações de Recorrências

- Exemplo: Torre de Hanoi
  - Objetivo: transferir os n discos de A para C
  - Regras:
    - Mover um disco por vez.
    - Nunca colocar um disco maior em cima de um menor.
  - Solução Recursiva:
    - Transferir n-1 discos de A para B
    - Mover o maior disco de A para C
    - Transferir n-1 discos de B para C

Derivando Relações de Recorrências

```
Hanoi(A, C, B, n)
if n > 1
      Hanoi(A, B, C, n-1)
Move(A, C)
if n > 1
      Hanoi(B, C, A, n-1)
```

Relação de Recorrência

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

Derivando Relações de Recorrências

```
MergeSort(A, n)
if n ≤ 1
      return A
return merge(MergeSort(A1, n/2), MergeSort(A2, n/2))
```

Relação de Recorrência

T(1) = c
T(n) = 2.T(n/2) + d.n

Resolvendo Relações de Recorrência

- Resolver uma relação de recorrência nem sempre é fácil.
- Resolvendo uma relação de recorrência, determina-se o tempo de execução do algoritmo recursivo correspondente.
- Relação de recorrência: $T(n) = T(n_1) + T(n_2) +...+ T(n_a) + f(n)$
- É mais fácil quando temos a subproblemas de mesmo tamanho que é uma fração de n (por exemplo, n/b):
  - $T(n) = a.T(n/b) + f(n)$
- Como resolver:
  - Método do chute
  - Método da árvore de recursão
  - Método do desdobramento
  - Método master

Método do Chute e Prova por Indução

- Seja a seguinte relação de recorrência:
  - $T(1) = 1$
  - $T(n) = T(n – 1) + 3n + 2$, para $n \geq 2$
- A relação de recorrência é resolvida em duas partes:
  1. Chute: $T(n) = 3n^2/2 + 7n/2 – 4$
  2. Prova:
     1. Caso base é para n=1
     2. H.I.: assumir que é válido para n-1
     3. Provar T(n)

  Se a prova for confirmada, $T(n)$ é $O(n^2)$

Método do Chute e Prova por Indução

- Seja a seguinte relação de recorrência:
  - $T(1) = 1$
  - $T(n) = 2.T(n/2) + n$, para $n \geq 2$
- A relação de recorrência é resolvida em duas partes:
  1. Chute: $T(n) = n + n.\log n$
  2. Prova:
     1. Caso base: $1 + 1.\log 1 = 1$
     2. H.I.: assumir que é válido para valores até n-1
     3. Provar T(n):
        - $=2.(n/2 + n/2.\log n/2) + n$
        - $=n + n.(\log n -1) + n$
        - $=n + n.\log n$

  Logo, $T(n)$ é $O(n.\log n)$

Método da Árvore de Recursão

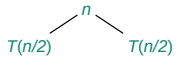
- Talvez o método mais intuitivo.
- Consiste em desenhar uma árvore cujos nós representam os tamanhos dos correspondentes problemas.
- Cada nível i contém todos os subproblemas de profundidade i.
- Dois aspectos importantes:
  - A altura da árvore.
  - O número de passos executados de cada nível.
- A solução da recorrência (tempo de execução do algoritmo) é a soma de todos os passos de todos os níveis.

Método da Árvore de Recursão

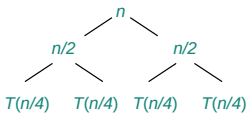
- Resolver $T(n) = 2.T(n/2) + n$

$T(n)$

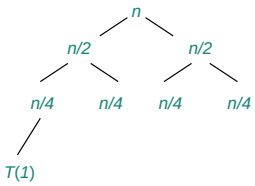
Método da Árvore de Recursão



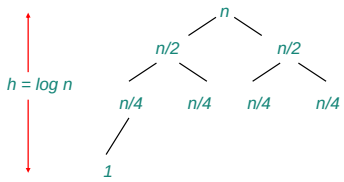
Método da Árvore de Recursão



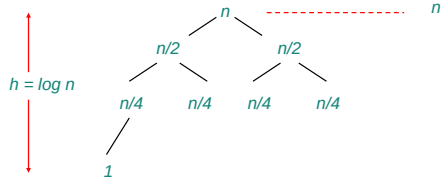
Método da Árvore de Recursão



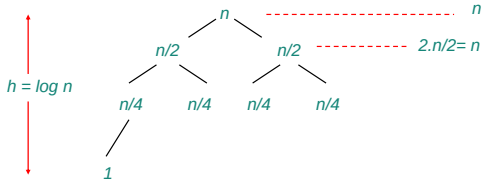
Método da Árvore de Recursão



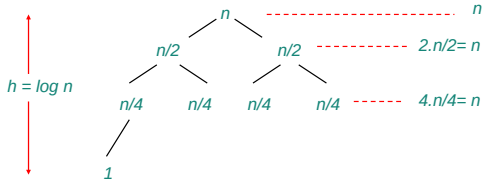
Método da Árvore de Recursão



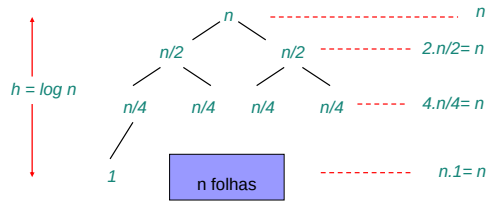
Método da Árvore de Recursão



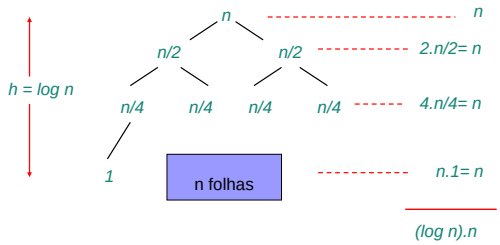
Método da Árvore de Recursão



Método da Árvore de Recursão



Método da Árvore de Recursão



Método do Desdobramento

- Esse método é o da árvore de recursão, representado de forma algébrica.
- Consiste em:
  - Usar (algumas poucas) substituições repetidamente até encontrar um padrão.
  - Escrever uma fórmula em termos de *n* e o número de substituições *i*.
  - Escolher *i* de tal forma que todas as referências a *T()* sejam referências ao caso base.
  - Resolver a fórmula.

Método do Desdobramento – Exemplo 1

- Solução para o problema da pizza:
  - T(1) = 2
  - T(n) = T(n – 1) + n , para n≥2
- Desdobrando a relação de recorrência:
  T(n) = T(n-1) + n
  T(n) = T(n-2) + (n-1) + n
  T(n) = T(n-3) + (n-2) + (n-1) + n
  ...
  T(n) = T(n-i) + (n-i+1) + ... + (n-1) + n
- Caso base alcançado quando i=n-1
- T(n) = 2 + 2 + 3 + ... + (n – 1) + n
- T(n) = 1 + n.(n-1)/2
- Logo, T(n) = $O(n^2)$

Método do Desdobramento – Exemplo 2

- Solução para o problema da Torre de Hanoi:
  - T(1) = 1
  - T(n) = 2.T(n – 1) + 1 , para n≥2
- Desdobrando a relação de recorrência:
  T(n) = 2.T(n – 1) + 1
  T(n) = 2.(2.T(n-2) + 1) + 1 = 4.T(n-2) + 2 + 1
  T(n) = 4.(2.T(n-3) + 1) + 2 + 1 = 8.T(n-3) + 4 + 2 + 1
  ...
  T(n) = 2^i.T(n-i) + 2^{i-1} + 2^{i-2} ... + 2^1 + 1
- Caso base alcançado quando i=n-1
- T(n) = 2^{n-1} + 2^{n-2} + 2^{n-3} + ... + 2^1 + 1
- Isso é uma soma geométrica
- Logo, T(n) = 2^n – 1 = $O(2^n)$

Método Master

- Teorema que resolve quase todas as recorrências.
- T(n) da forma *a.T(n/b) + f(n), a,b > 1*
- Casos:
  1. Se f(n) ∈ $O(n^{\log_a b - \varepsilon})$, para algum ε > 0, temos que:
     - T(n) ∈ $\Theta(n^{\log_a b})$.
  2. Se f(n) ∈ $O(n^{\log_a b})$, temos que:
     - T(n) ∈ $\Theta(n^{\log_a b}.\log n)$.
  3. Se f(n) ∈ $O(n^{\log_b + \varepsilon})$, para algum ε > 0 e se a.f(n/b)≤c.f(n) para algum c > 0 e n suficientemente grande, temos que:
     - T(n) ∈ Θ(f(n)).

Método Master – Exemplo 1

- MergeSort:
  - T(n) = 2.T(n/2) + n
  - a = b = 2
  - f(n) = n
- $\log^a_b = 1$. Cai no caso 2.
- Logo, T(n) = $\Theta(n.\log n)$

Método Master – Exemplo 1

- T(n) = 9.T(n/3) + n
  - a = 9, b = 3
  - f(n) = n
- $\log^a_b = 2$. Se $\varepsilon = 1$, Cai no caso 1.
- Logo, T(n) = $\Theta(n^2)$