Projeto e Análise de Algoritmos

Algoritmo Mergesort e Relações de Recorrência

Atílio G. Luiz

Primeiro Semestre de 2024

Projetando algoritmos

Projetando algoritmos

- Divisão e conquista

Entendendo e melhorando

Até agora:

- ordenamos incrementalmente com o INSERTION-SORT
- vimos que sua complexidade de pior caso é $\Theta(n^2)$

Vamos estudar uma maneira alternativa de ordenar números

- vamos utilizar uma técnica recursiva chamada de divisão e conquista
- muitas vezes, obtemos algoritmos mais rápidos do que os incrementais

Algoritmos recursivos

> *"To understand recursion, we must first understand recursion."*
>
> (autor desconhecido)

- Um algoritmo recursivo resolve um problema
  - diretamente, se a instância for pequena
  - executando a si mesmo, se a instância não for pequena
- A chamada recursiva deve receber uma instância menor

Divisão e conquista

Um algoritmo de divisão e conquista tem três etapas:

1. Divisão: dividir o problema em subproblemas semelhantes, mas com instâncias menores
2. Conquista: cada subproblema é resolvido recursivamente, ou diretamente se os subproblemas forem pequenos
3. Combinação: as soluções dos subproblemas são combinadas para obter uma solução da instância original

Exemplo: ordenando usando divisão e conquista

MERGE-SORT é um exemplo clássico de divisão e conquista.

Ideia:

1. Divisão: divida um vetor de tamanho n em dois subvetores de tamanhos $\lceil n/2 \rceil$ e $\lfloor n/2 \rfloor$
2. Conquista: ordene os dois subvetores recursivamente
3. Combinação: intercale os dois subvetores obtendo um vetor ordenado

Mergesort

O vetor de entrada é representado como $A[p \dots r]$, com $p \leq r$.

```
Merge-Sort(A, p, r)
  se p < r
    então q = ⌊(p + r)/2⌋
          Merge-Sort(A, p, q)
          Merge-Sort(A, q + 1, r)
          Intercala(A, p, q, r)
```

	p				q				r
A	66	33	55	44	99	11	77	22	88

Mergesort

O vetor de entrada é representado como $A[p \dots r]$, com $p \leq r$.

Merge-Sort(A, p, r)

```
se p < r
    então q = ⌊(p + r)/2⌋
          Merge-Sort(A, p, q)
          Merge-Sort(A, q + 1, r)
          Intercala(A, p, q, r)
```

	p				q				r
A	33	44	55	66	99	11	77	22	88

Mergesort

O vetor de entrada é representado como $A[p \dots r]$, com $p \leq r$.

Merge-Sort(A, p, r)
```
se p < r
    então q = ⌊(p + r)/2⌋
            MERGE-SORT(A, p, q)
            MERGE-SORT(A, q + 1, r)
            INTERCALA(A, p, q, r)
```

	p				q				r
A	33	44	55	66	99	11	22	77	88

Mergesort

O vetor de entrada é representado como $A[p \dots r]$, com $p \leq r$.

Merge-Sort(A, p, r)

    se $p < r$
        então $q = \lfloor (p + r)/2 \rfloor$
                Merge-Sort(A, p, q)
                Merge-Sort$(A, q + 1, r)$
                Intercala(A, p, q, r)

	p				q				r
A	11	22	33	44	55	66	77	88	99

Combinando soluções dos subproblemas

Problema: Intercalar dois subvetores

Entrada: vetor $A[p \dots r]$ tal que

1. subvetor $A[p \dots q]$ está ordenado
2. subvetor $A[q+1 \dots r]$ está ordenado

Saída: rearranjo $A[p \dots r]$ ordenado

Entrada:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

	p				q				r
A	11	22	33	44	55	66	77	88	99

Intercalando

Como intercalar os dois subvetores?

Intercalando

	p				q				r
A	22	33	55	77	99	11	44	66	88
B									

Intercalando

	k								
A									
	i								j
B	22	33	55	77	99	88	66	44	11

Intercalando

		k							
A	11								

	i							j	
B	22	33	55	77	99	88	66	44	11

Intercalando

			k						
A	11	22							

		i						j	
B	22	33	55	77	99	88	66	44	11

Intercalando

				k					
A	11	22	33						

			i					j	
B	22	33	55	77	99	88	66	44	11

Intercalando

					k				
A	11	22	33	44					

			i				j		
B	22	33	55	77	99	88	66	44	11

Intercalando

					k				
A	11	22	33	44	55				

				i			j		
B	22	33	55	77	99	88	66	44	11

Intercalando

							k		
A	11	22	33	44	55	66			

				i		j			
B	22	33	55	77	99	88	66	44	11

Intercalando

								k	
A	11	22	33	44	55	66	77		

					i	j			
B	22	33	55	77	99	88	66	44	11

Intercalando

									k
A	11	22	33	44	55	66	77	88	

					$i{=}j$				
B	22	33	55	77	99	88	66	44	11

Intercalando

A	11	22	33	44	55	66	77	88	99

					j	i			
B	22	33	55	77	99	88	66	44	11

Pseudocódigo de INTERCALA

```
Intercala(A, p, q, r)
    n = r − p + 1
    crie array B[1...n] vazio
    para i = p até q faça
        B[i − p + 1] = A[i]
    para j = q + 1 até r faça
        B[n + q + 1 − j] = A[j]
    i = 1
    j = n
    para k = p até r faça
        se B[i] ≤ B[j]
            então A[k] = B[i]
                  i = i + 1
            senão A[k] = B[j]
                  j = j − 1
```

Complexidade de INTERCALA

Entrada:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

	p				q				r
A	11	22	33	44	55	66	77	88	99

Tamanho da entrada: $n = r - p + 1$

Consumo de tempo: $\Theta(n)$

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

	p	r							
A	66	33							

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

	p	r							
A	66	33							

	$p = r$								
A	66								

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

	p	r							
A	66	33							

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

	p	r							
A	66	33							

		$p = r$							
A		33							

Ordenando por intercalação

	p				q				r
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

	p	r							
A	66	33							

Ordenando por intercalação

	p				q				r
A	33	66	55	44	99	11	77	22	88

	p		q		r				
A	33	66	55	44	99				

	p	q	r						
A	33	66	55						

	p	r							
A	33	66							

	p				q				r
A	33	66	55	44	99	11	77	22	88

	p		q		r				
A	33	66	55	44	99				

	p	q	r						
A	33	66	55						

Ordenando por intercalação

	p				q				r
A	33	66	55	44	99	11	77	22	88

	p		q		r				
A	33	66	55	44	99				

	p	q	r						
A	33	66	55						

			$p = r$						
A			55						

Ordenando por intercalação

	p				q				r
A	33	66	55	44	99	11	77	22	88

	p		q		r				
A	33	66	55	44	99				

	p	q	r						
A	33	66	55						

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

	p	q	r						
A	33	55	66						

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

				$p = r$					
A				44					

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

					$p = r$				
A					99				

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

Ordenando por intercalação

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

	p		q		r				
A	33	44	55	66	99				

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

						$p = r$			
A						11			

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

							$p = r$		
A							77		

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

								$p = r$	
A								22	

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

									$p = r$
A									88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	22	77	88

						p			r
A						11	22	77	88

Ordenando por intercalação

	p				q				r
A	33	44	55	66	99	11	22	77	88

Ordenando por intercalação

	p				q				r
A	11	22	33	44	55	66	77	88	99

Ordenando por intercalação

	p				q				r
A	11	22	33	44	55	66	77	88	99

Complexidade de MERGE-SORT

Merge-Sort(A, p, r)

```
se p < r
    então q = ⌊(p + r)/2⌋
          MERGE-SORT(A, p, q)
          MERGE-SORT(A, q + 1, r)
          INTERCALA(A, p, q, r)
```

- Tamanho da entrada: $n = r - p + 1$
- Seja $T(n)$ o número de instruções executadas no pior caso

Complexidade de Merge-Sort

Merge-Sort(A, p, r)

1 **se** $p < r$

2 **então** $q = \lfloor (p+r)/2 \rfloor$

3 Merge-Sort(A, p, q)

4 Merge-Sort$(A, q+1, r)$

5 Intercala(A, p, q, r)

Linha	Tempo
1	$\Theta(1)$
2	$\Theta(1)$
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	$\Theta(n)$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$$

Recorrência

O tempo de MERGE-SORT é dado pela fórmula

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{se } n = 2, 3, 4, \dots \end{cases}$$

Obtemos uma fórmula de recorrência

- é a descrição de uma função em termos de si mesma.
- o tempo de um algoritmo recursivo costuma ser descrito por uma recorrência

Mas queremos uma fórmula fechada

- Nesse caso, $T(n) = \Theta(n \lg n)$
- Aprenderemos a resolver recorrências depois

Correção de algoritmos recursivos

Correção de algoritmos recursivos

- Algoritmo MERGE-SORT

Correção de Merge-Sort

```
Merge-Sort(A, p, r)
  se p < r
     então q = ⌊(p + r)/2⌋
           Merge-Sort(A, p, q)
           Merge-Sort(A, q + 1, r)
           Intercala(A, p, q, r)
```

Como demonstrar a correção de um algoritmo recursivo?

- podemos usar indução diretamente
- precisamos verificar as demais sub-rotinas

Correção de Merge-Sort

- queremos mostrar que Merge-Sort ordena $A[p \dots r]$
- basta usar indução em $n = r - p + 1$

Revendo INTERCALA

```
Intercala(A, p, q, r)
1   n = r − p + 1
2   crie array B[1...n] vazio
3   para i = p até q faça
4       B[i − p + 1] = A[i]
5   para j = q + 1 até r faça
6       B[n + q + 1 − j] = A[j]
7   i = 1
8   j = n
9   para k = p até r faça
10      se B[i] ≤ B[j]
11          então A[k] = B[i]
12                i = i + 1
13          senão A[k] = B[j]
14                j = j − 1
```

Invariante

Em cada iteração da linha 9, vale:

1. $A[p \dots k-1]$ está ordenado,
2. $A[p \dots k-1]$ contém itens de $B[p \dots i-1]$ e de $B[j+1 \dots r]$,
3. $B[i] \geq A[k-1]$ e $B[j] \geq A[k-1]$.

Exercício: demonstre

- a invariante
- a correção de INTERCALA

Correção de MERGE-SORT

Base da indução:

- para um vetor de tamanho 0 ou 1, o vetor não é alterado
- assim MERGE-SORT está correto nesses casos

Caso geral:

- considere um vetor de tamanho n
- suponha que MERGE-SORT ordena vetores menores
- daí, após as chamadas recursivas, os subvetores $A[p \dots q]$ e $A[q+1 \dots r]$ estão ordenados
- como INTERCALA está correto, o vetor $A[p \dots r]$ estará ordenado no final do algoritmo

Recorrências

Complexidade de MERGE-SORT

```
Merge-Sort(A, p, r)
  se p < r
    então q = ⌊(p + r)/2⌋
          MERGE-SORT(A, p, q)
          MERGE-SORT(A, q + 1, r)
          INTERCALA(A, p, q, r)
```

- Vamos relembrar a complexidade do MERGE-SORT
- Tamanho da entrada: $n = r - p + 1$
- Seja $T(n)$ o número de instruções executadas no pior caso

Complexidade de Merge-Sort

```
Merge-Sort(A, p, r)
1  se p < r
2     então q = ⌊(p + r)/2⌋
3           Merge-Sort(A, p, q)
4           Merge-Sort(A, q + 1, r)
5           Intercala(A, p, q, r)
```

Linha	Tempo
1	c_1
2	c_2
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	$c_5 n + d_5$

$$
\begin{aligned}
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + c_5 n + d_5 + c_1 + c_2 \\
&= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + b
\end{aligned}
$$

Resolução de recorrências

Obtemos a recorrência:

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + b & \text{se } n \geq 2, \end{cases}$$

- Queremos uma fórmula fechada para $T(n)$.
- Não é necessária a solução exata.
- Basta encontrar uma função $f(n)$ tal que $T(n) = \Theta(f(n))$.

Resolução de recorrências

Existem alguns métodos comuns para resolver recorrências:

- substituição
- árvore de recorrência

Veremos também o chamado Teorema Mestre

- aplicável a uma família comum de recorrências
- fornece uma fórmula fechada diretamente

Recorrências

- Método da substituição

Método da substituição

Ideia:

1. adivinhar a forma da solução
2. usar indução para achar as constantes e demonstrar que a solução funciona.

Nem sempre é fácil chutar a solução

- é necessário ter experiência
- mas vamos obter sugestões com os métodos estudados

Exemplo

Exemplo

Encontre uma fórmula fechada para

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{se } n \geq 2 \end{cases}$$

Esta recorrência é o tempo de execução do mergesort.

- Chutamos que $T(n) = O(n \lg n)$.
- Observe que há uma constante escondida na notação O.
- Para usar indução, podemos escolher essa constante ou trabalhar com uma constante c arbitrária.

Passos para o método da substituição

1. Chutamos que $T(n) = O(n \lg n)$.
2. Queremos provar por indução que $T(n) \leq cn \lg n$ para todo $n \geq n_0$.
3. Suponha $T(k) \leq ck \lg k$ para algum c genérico e para todo $n_0 \leq k < n$ (indução forte).
4. Substitua e desenvolva a expressão para $T(n)$.
5. Determine c e n_0 de forma a provar o passo da indução e satisfazer o caso básico.

Passo indutivo

Queremos provar que $T(n) \leq cn\lg n$.

$$
\begin{aligned}
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \\
&\leq c\left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + c\left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \qquad \text{(hipótese indutiva)} \\
&\leq c\left\lceil \frac{n}{2} \right\rceil \lg n + c\left\lfloor \frac{n}{2} \right\rfloor (\lg n - 1) + n \\
&= c\left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor\right) \lg n - c\left\lfloor \frac{n}{2} \right\rfloor + n \\
&= cn\lg n - c\left\lfloor \frac{n}{2} \right\rfloor + n \\
&\leq cn\lg n. \qquad \text{(escolhendo } c \text{ e } n \geq n_0 \text{ adequado)}
\end{aligned}
$$

- Para a última desigualdade, queremos $-c\lfloor n/2 \rfloor + n \leq 0$
- Basta que $c = 3$ e $n_0 \geq 2$.

Base da indução

Ainda falta provar a base

- Temos $T(1) = 1$, mas $3 \cdot 1 \cdot \lg 1 = 0$
- A desigualdade não vale para $n = 1$
- Não temos uma base da indução

Ok, queremos mostrar apenas $T(n) = O(n \lg n)$.

- Basta mostrar que a desigualdade vale para $n \geq n_0$
- Vamos escolher $n_0 = 2$

Base da indução:

- Para $n = 2$ e $n = 3$, temos

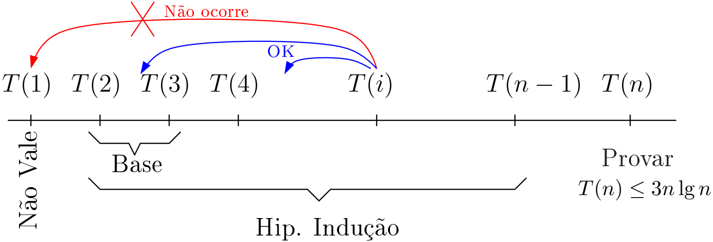
$$T(2) = T(1) + T(1) + 2 = 4 \leq 3 \cdot 2 \cdot \lg 2 = 6$$
$$T(3) = T(2) + T(1) + 3 = 8 \leq 3 \cdot 3 \cdot \lg 3 \approx 14{,}26$$

- Por que precisamos de dois casos básicos?

Exemplo: verificando a base

Não podemos ter $k = 1$ quando usamos a hipótese da indução:

$$T(1) = 1$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$$



Modificando um pouco o exemplo

Mas se tivéssemos $T(1) = 8$?

- A desigualdade não vale para $n = 2$
- Temos $T(2) = 8 + 8 + 2 = 18$, mas $3 \cdot 2 \cdot \lg 2 = 6$
- Podemos escolher uma constante multiplicativa maior.
- Tentando $T(n) \leq 10 n \lg n$, obtemos

$$T(2) = 18 \leq 10 \cdot 2 \cdot \lg 2 = 20$$
$$T(3) = 29 \leq 10 \cdot 3 \cdot \lg 3 \approx 47{,}55$$

Conclusão:

- Se o passo da indução vale, então podemos escolher valores adequados para as constantes c e n_0.

Completando o exemplo

Exemplo

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{se } n \geq 2 \end{cases}$$

- Já mostramos que $T(n) = O(n \lg n)$.
- Mas queremos mostrar que $T(n) = \Theta(n \lg n)$.
  - Basta mostrar $T(n) = \Omega(n \lg n)$.
  - Faça como exercício.

Sutilezas

Algumas vezes é necessário fortalecer a hipótese de indução.

Exemplo

Resolva a recorrência

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 & \text{se } n \geq 2 \end{cases}$$

Chutamos que $T(n) \leq cn$ para alguma constante c.

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leq c\lceil n/2 \rceil + c\lfloor n/2 \rfloor + 1 \\ &= cn + 1. \end{aligned}$$

- Não deu certo
- Mas de fato $T(n) \in \Theta(n)$!

Fortalecendo a hipótese

- Vamos fazer uma hipótese de indução um pouco diferente.
- Vamos mostrar que $T(n) \leq cn - b$ para $c > 0$ e $b > 0$

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leq c\lceil n/2 \rceil - b + c\lfloor n/2 \rfloor - b + 1 \\ &= cn - 2b + 1 \\ &\leq cn - b \end{aligned}$$

- Para a última desigualdade, basta escolher $b \geq 1$.
- Isso mostra o passo indutivo.

Exercício – Método da substituição

Exercício 4.3-6 CRLS

Mostre que a solução para a recorrência $T(n) = 2T(\lfloor n/2 \rfloor + 13) + n$ é $\Theta(n \lg n)$.

- Ela parece mais difícil por causa do termo 13.
- Mostre isso como exercício.

Recorrências

- Método da árvore de recorrência

Árvore de recorrência

- Uma árvore de recorrência é geralmente usada para obter uma bos estimativa sobre a solução de uma relação de recorrência. Essa estimativa é depois verificada usando o método da substituição.
- Ao usar uma árvore de recorrência para gerar uma boa estimativa, muitas vezes você pode tolerar uma pequena quantidade de "desleixo", já que estará verificando seu palpite mais tarde.
- Tipos de "desleixo" comuns: desconsiderar pisos e tetos, e ainda supor que a entrada tem tamanho específico.

Árvore de recorrência

Ideia:

1. crie uma árvore de recorrência:
   - os nós representam os termos independentes
   - os filhos representam as subfunções recorrentes
2. somamos os termos de cada nível da árvore
3. depois somamos todos os níveis

Vantagens

- útil quando há vários termos recorrentes
- é mais fácil organizar as contas

Exemplo

Exemplo

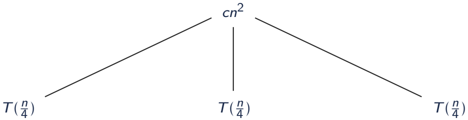
Encontre um limitante superior assintótico para

$$T(n) = \begin{cases} 1 & \text{se } n \leq 3 \\ 3T(\lfloor n/4 \rfloor) + cn^2 & \text{se } n \geq 4 \end{cases}$$

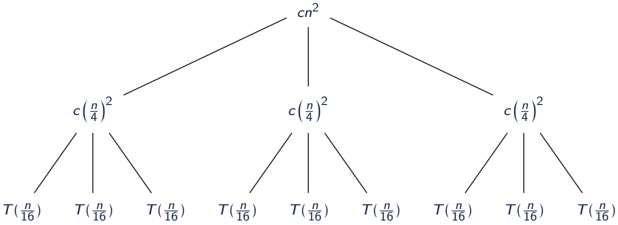
Simplificando

- Desconsideramos pisos e tetos
- Vamos supor que $n = 4^k$
- Depois, verificamos com o método da substituição

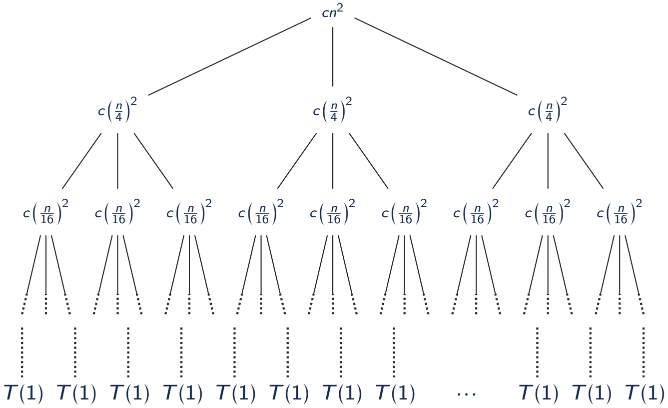
Árvore de recorrência



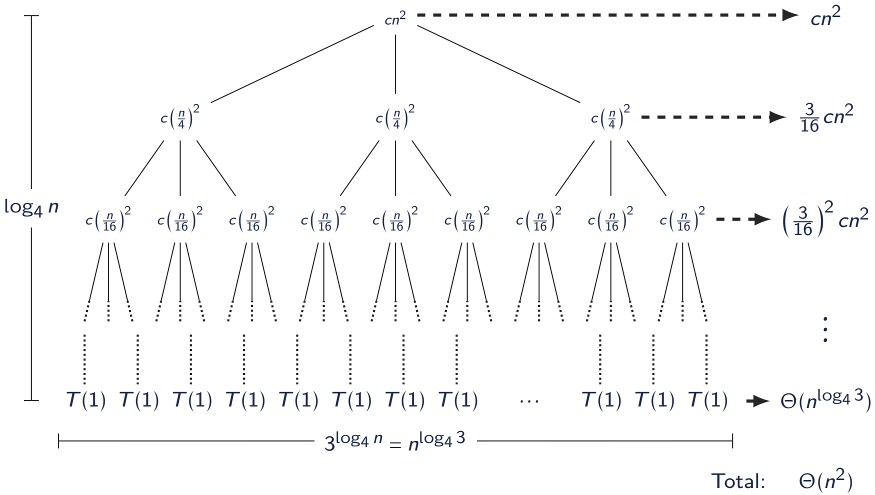
Árvore de recorrência



Árvore de recorrência



Árvore de recorrência



Somando os termos

- A árvore tem altura $\log_4 n$
- A soma de cada nível interno é $\left(\frac{3}{16}\right)^i cn^2$
- O número de folhas é $3^{\log_4 n} = n^{\log_4 3}$

$$
\begin{aligned}
T(n) &= \left(cn^2 + \frac{3}{16} cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 \right) + \Theta(n^{\log_4 3}) \\
&= \left(\sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i \right) cn^2 + \Theta(n^{\log_4 3}) \\
&\le \left(\sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i \right) cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}), \qquad \text{pela Equação } A.6
\end{aligned}
$$

Concluímos que $T(n) \in O(n^2)$.

Verificando com o método da substituição

- Vamos provar que $T(n) = O(n^2)$ é um limitante superior para a recorrência $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.
- Queremos mostrar que $T(n) \leq dn^2$ para alguma constante $d > 0$.

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

onde o último passo vale sempre que $d \geq (16/13)c$.
Logo, $T(n) = O(n^2)$.

- De fato, também temos que $T(n) = \Omega(n^2)$ dado que a primeira chamada recursiva contribui com o custo de $\Theta(n^2)$. Logo, $T(n) = \Theta(n^2)$.

Passo a passo

Se a árvore de recorrência que você obteve for do tipo cheia, então os seguintes passos podem te ajudar a calcular o limitante:

(1) Determine a altura da árvore de recorrência.
   - qual o tamanho do subproblema para um nó na profundidade i da árvore?

(2) Determine o custo das chamadas recursivas para cada nível da árvore.
   - qual a taxa de crescimento do número de subproblemas a cada chamada recursiva?
   - qual o custo de um nó na profundidade i da árvore?

(3) Por fim, some os custos de cada nível a fim de obter o custo total.

Exercícios

- Use uma árvore de recorrência para determinar um limitante superior assintótico para a recorrência $T(n) = T(n/2) + n^2$. Depois, use o método da substituição para verificar o seu limitante.
- Use uma árvore de recorrência para determinar um limitante superior assintótico para a recorrência $T(n) = 3T(\lfloor n/2 \rfloor) + n$. Depois, use o método da substituição para verificar o seu limitante.

Outro exemplo

Exemplo

Encontre uma fórmula fechada para

$$T(n) = \begin{cases} 1 & \text{se } n \leq 2 \\ T(n/3) + T(2n/3) + n & \text{se } n \geq 3 \end{cases}$$

Esse exemplo é um pouco mais complicado

- Os termos recorrentes são diferentes
- A árvore não será completa
- Vamos mostrar juntos que $T(n) \in O(n \lg n)$

Recorrências

- Recorrências e notação assintótica

Recorrências com notação assintótica

- Escrevemos

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 2 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{se } n \geq 3 \end{cases}$$

- Para representar a família de recorrências

$$T(n) = \begin{cases} a & \text{se } n \leq 2 \\ 3T(\lfloor n/4 \rfloor) + f(n) & \text{se } n \geq 3 \end{cases}$$

onde a é uma constante e $f(n)$ está em $\Theta(n^2)$

- Todas elas têm a mesma solução assintótica
- A mesma coisa vale para as outras classes de função

Cuidados com a notação assintótica

Exemplo

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n & \text{se } n \geq 2, \end{cases}$$

- O uso descuidado da notação assintótica leva a erros
- Já sabemos que $T(n) = \Theta(n \log n)$
- Mas vamos "provar" que $T(n) = O(n)$!

Cuidados com a notação assintótica (cont)

Vamos mostrar que $T(n) \leq cn$ para alguma constante $c > 0$.

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2c\lfloor n/2 \rfloor + n \\ &\leq cn + n \\ &= O(n) \qquad \longleftarrow \text{ERRADO!} \end{aligned}$$

Qual é o erro?

- A nossa afirmação é que $T(n) \leq cn$
- Mas mostramos que $T(n) \leq (c+1)n$
- Não concluímos o passo indutivo

Recorrências

- Teorema Mestre

Teorema Mestre

Veremos um teorema para resolver recorrências da forma

$$T(n) = aT(n/b) + f(n),$$

- os valores $a \geq 1$ e $b > 1$ são constantes
- $f(n)$ é uma função assintoticamente positiva
- também vale se substituirmos n/b por $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$
- o teorema não se aplica a todas as recorrências dessa forma

Teorema Mestre

Teorema Mestre

Sejam $a \geq 1$ e $b > 1$ constantes, seja $f(n)$ uma função e seja $T(n)$ definida para os inteiros não negativos pela relação de recorrência

$$T(n) = aT(n/b) + f(n).$$

Então $T(n)$ tem os seguintes limitantes assintóticos:

1. Se $f(n) \in O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$
2. Se $f(n) \in \Theta(n^{\log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log n)$
3. Se $f(n) \in \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$ e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e todo n suficientemente grande, então $T(n) \in \Theta(f(n))$

Exemplos de Recorrências

Exemplos para os quais Teorema Mestre se aplica:

- Caso 1:
  $T(n) = 9T(n/3) + n$
  $T(n) = 4T(n/2) + n \log n$

- Caso 2:
  $T(n) = T(2n/3) + 1$
  $T(n) = 2T(n/2) + (n + \log n)$

- Caso 3:
  $T(n) = 3T(n/4) + n \log n$

Exemplos de Recorrências

Exemplos para os quais Teorema Mestre **não** se aplica:

- $T(n) = T(n-1) + n$
- $T(n) = T(n-a) + T(a) + n$, ($a \geq 1$ inteiro)
- $T(n) = T(\alpha n) + T((1-\alpha)n) + n$, ($0 < \alpha < 1$)
- $T(n) = T(n-1) + \log n$
- $T(n) = 2T(\frac{n}{2}) + n \log n$