Projeto e Análise de Algoritmos

Projeto de algoritmos por indução e divisão e conquista

Cid Carvalho de Souza, Cândida Nunes da Silva et al.

Primeiro Semestre de 2017

Recorrências

Resolução de Recorrências

- Relações de recorrência expressam a complexidade de algoritmos recursivos como os algoritmos de divisão e conquista.
- É preciso saber resolver as recorrências para que possamos efetivamente determinar a complexidade dos algoritmos recursivos.

Mergesort

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

Qual é a complexidade de MERGESORT?

Seja $T(n) :=$ o consumo de tempo máximo (pior caso) em função de $n = r - p + 1$

Complexidade do Mergesort

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

linha	consumo de tempo
1	?
2	?
3	?
4	?
5	?

$T(n) = ?$

Complexidade do Mergesort

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

linha	consumo de tempo
1	b_0
2	b_1
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	an

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + (b_0 + b_1)$$

Resolução de recorrências

- Queremos resolver a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + b \quad \text{para } n \geq 2. \end{aligned}$$

- Resolver uma recorrência significa encontrar uma fórmula fechada para $T(n)$.
- Não é necessário achar uma solução exata. Basta encontrar uma função $f(n)$ tal que $T(n) \in \Theta(f(n))$.

Resolução de recorrências

Alguns métodos para resolução de recorrências:

- substituição
- iteração
- árvore de recorrência

Veremos também um resultado bem geral que permite resolver várias recorrências diretamente: Teorema Master.

Método da substituição

- Idéia básica: "adivinhe" qual é a solução e prove por indução que ela funciona!
- Método poderoso mas nem sempre aplicável (obviamente).
- Com prática e experiência fica mais fácil de usar!

Exemplo

Considere a recorrência:

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

Chuto que $T(n) \in O(n \lg n)$.

Mais precisamente, chuto que $T(n) \leq 3n \lg n$.

(Lembre que $\lg n = \log_2 n$.)

Exemplo

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \\ &\leq 3 \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + 3 \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\ &\leq 3 \left\lceil \frac{n}{2} \right\rceil \lg n + 3 \left\lfloor \frac{n}{2} \right\rfloor (\lg n - 1) + n \\ &= 3 \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) \lg n - 3 \left\lfloor \frac{n}{2} \right\rfloor + n \\ &= 3n \lg n - 3 \left\lfloor \frac{n}{2} \right\rfloor + n \\ &\leq 3n \lg n. \end{aligned}$$

(Yeeeeeeesssssss!)

Exemplo

- Mas espere um pouco!
- $T(1) = 1$ e $3.1. \lg 1 = 0$ e a base da indução não funciona!
- Certo, mas lembre-se da definição da classe $O(\ )$.

  Só preciso provar que $T(n) \leq 3n \lg n$ para $n \geq n_0$ onde n_0 é alguma constante.

  Vamos tentar com $n_0 = 2$. Nesse caso

$$T(2) = T(1) + T(1) + 2 = 4 \leq 3.2. \lg 2 = 6,$$

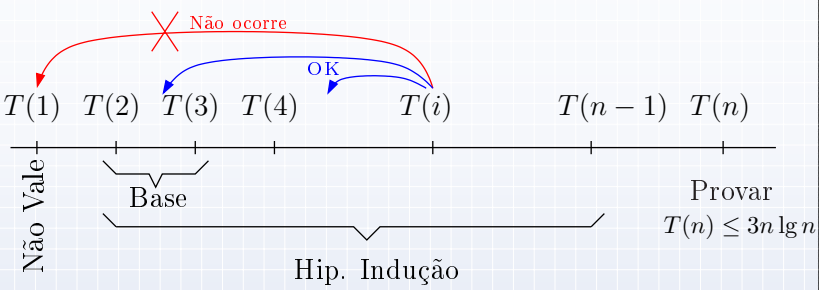
$$T(3) = T(2) + T(1) + 3 = 8 \leq 3.3. \lg 3 \approx 14{,}26,$$

  Assim, o chute vale para $n = 2$ e $n = 3$ (esta será a Base!)

  Como a recorrência de $T(n)$, para $n > 3$, recai em recorrências menores até cair na base, estamos feitos.

Recorrência e Indução

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



Exemplo

- Certo, funcionou quando $T(1) = 1$.
- Mas e se por exemplo $T(1) = 8$?

  Então $T(2) = 8 + 8 + 2 = 18$ e $3.2.\lg 2 = 6$.
  Não deu certo...
- Certo, mas aí basta escolher uma constante maior. Mostra-se do mesmo jeito que $T(n) \leq 10n \lg n$, pois
  $T(2) = 18 \leq 10.2.\lg 2 = 20$,
  $T(3) = T(1)+T(2)+3 = 8+18+3 = 29 \leq 10.3.\lg 3 \approx 47{,}55$.
- De modo geral, se o passo de indução funciona ($T(n) \leq cn \lg n$), é possível escolher c e a base da indução de modo conveniente!

Como achar as constantes?

- Tudo bem. Dá até para chutar que $T(n)$ pertence a classe $O(n \lg n)$.
- Mas como descobrir que $T(n) \leq 3n \lg n$? Como achar a constante 3?
- Eis um método simples: suponha como hipótese de indução que $T(n) \leq cn \lg n$ para $n \geq n_0$ onde c e n_0 são constantes que vou tentar determinar.

Primeira tentativa

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \\ &\leq c \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + c \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\ &\leq c \left\lceil \frac{n}{2} \right\rceil \lg n + c \left\lfloor \frac{n}{2} \right\rfloor \lg n + n \\ &= c \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) \lg n + n \\ &= cn \lg n + n \end{aligned}$$

(Hummm, não deu certo...)

Segunda tentativa

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

Para garantir a última desigualdade basta que $-c\lceil n/2 \rceil + n \leq 0$ e $c = 3$ funciona. (Yeeeeeeesssssss!)

Completando o exemplo

Mostramos que a recorrência

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

satisfaz $T(n) \in O(n \lg n)$.

Mas quem garante que $T(n)$ não é "menor"?

O melhor é mostrar que $T(n) \in \Theta(n \lg n)$.

Resta então mostrar que $T(n) \in \Omega(n \lg n)$. A prova é similar. (Exercício!)

Como chutar?

Não há nenhuma receita genérica para adivinhar soluções de recorrências. A experiência é o fator mais importante.

Felizmente, há várias idéias que podem ajudar.

Considere a recorrência

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

Ela é quase idêntica à anterior e podemos chutar que $T(n) \in \Theta(n \lg n)$. Isto de fato é verdade. (Exercício ou consulte o CLRS)

Como chutar?

Considere agora a recorrência

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

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 $T(\lfloor n/2 \rfloor)$ e $T(\lfloor n/2 \rfloor + 17)$ não é tanta.

Chuto então que $T(n) \in \Theta(n \lg n)$. (Exercício!)

Truques e sutilezas

Algumas vezes adivinhamos corretamente a solução de uma recorrência, mas as contas aparentemente não funcionam! Em geral, o que é necessário é fortalecer a hipótese de indução.

Considere a recorrência

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

Chutamos que $T(n) \in O(n)$ e tentamos mostrar que $T(n) \leq cn$ para alguma constante c.

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

(Humm, falhou...)

E agora? Será que erramos o chute? Será que $T(n) \in \Theta(n^2)$?

Truques e sutilezas

Na verdade, adivinhamos corretamente. Para provar isso, é preciso usar uma hipótese de indução mais forte.

Vamos mostrar que $T(n) \leq cn - b$ onde $b > 0$ é uma constante.

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

onde a última desigualdade vale se $b \geq 1$.
(Yeeeessss!)

Método da iteração

- Não é necessário adivinhar a resposta!
- Precisa fazer mais contas!
- Idéia: expandir (iterar) a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n e das condições iniciais.
- Precisa conhecer limitantes para várias somatórias.

Método da iteração

Considere a recorrência

$$\begin{aligned} T(n) &= b && \text{para } n \leq 3, \\ T(n) &= 3T(\lfloor n/4 \rfloor) + n && \text{para } n \geq 4. \end{aligned}$$

Iterando a recorrência obtemos

$$\begin{aligned} T(n) &= n + 3T(\lfloor n/4 \rfloor) \\ &= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) \\ &= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))) \\ &= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor). \end{aligned}$$

Certo, mas quando devo parar?
O i-ésimo termo da série é $3^i\lfloor n/4^i \rfloor$. Ela termina quando $\lfloor n/4^i \rfloor \leq 3$, ou seja, $i \geq \log_4 n$.

Método da iteração

Como $\lfloor n/4^i \rfloor \leq n/4^i$ temos que

$$\begin{aligned} T(n) &\leq n + 3n/4 + 9n/16 + 27n/64 + \cdots + 3^j b \\ T(n) &\leq n + 3n/4 + 9n/16 + 27n/64 + \cdots + d.3^{\log_4 n} \\ &\leq n.(1 + 3/4 + 9/16 + 27/64 + \cdots) + dn^{\log_4 3} \\ &\leq n \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + dn^{\log_4 3} \\ &= 4n + dn^{\log_4 3} \end{aligned}$$

pois $3^{\log_4 n} = n^{\log_4 3}$ e $\sum_{i=0}^{\infty} q^i = \frac{1}{1-q}$ para $0 < q < 1$.

Como $\log_4 3 < 1$ segue que $n^{\log_4 3} \in o(n)$ e logo, $T(n) \in O(n)$.

Método de iteração

- As contas ficam mais simples se supormos que a recorrência está definida apenas para potências de um número, por exemplo, $n = 4^i$.
- Note, entretanto, que a recorrência deve ser provada para todo natural suficientemente grande.
- Muitas vezes, é possível depois de iterar a recorrência, adivinhar a solução e usar o método da substituição!

Método de iteração

$$\begin{aligned} T(n) &= b && \text{para } n \leq 3, \\ T(n) &= 3T(\lfloor n/4 \rfloor) + n && \text{para } n \geq 4. \end{aligned}$$

Chuto que $T(n) \leq cn$.

$$\begin{aligned} T(n) &= 3T(\lfloor n/4 \rfloor) + n \\ &\leq 3c\lfloor n/4 \rfloor + n \\ &\leq 3c(n/4) + n \\ &\leq cn \end{aligned}$$

onde a última desigualdade vale se $c \geq 4$.
(Yeeessss!)

Árvore de recorrência

- 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.

Árvore de recorrência

Considere a recorrência

$$\begin{aligned} T(n) &= \Theta(1) && \text{para } n = 1, 2, 3, \\ T(n) &= 3T(\lfloor n/4 \rfloor) + cn^2 && \text{para } n \geq 4, \end{aligned}$$

onde $c > 0$ é uma constante.

Costuma-se (CLRS) usar a notação $T(n) = \Theta(1)$ para indicar que $T(n)$ é uma constante.

Árvore de recorrência

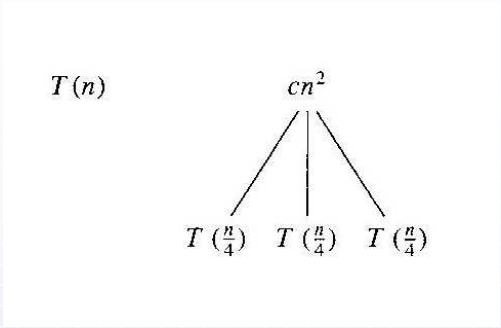
Simplificação

Vamos supor que a recorrência está definida apenas para potências de 4

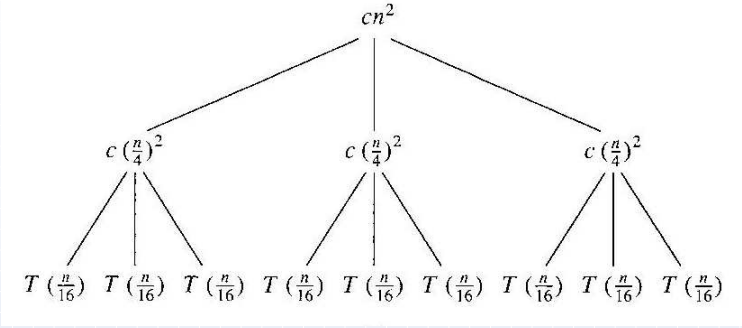
$$\begin{aligned} T(n) &= \Theta(1) && \text{para } n = 1, \\ T(n) &= 3T(n/4) + cn^2 && \text{para } n = 4, 16, \ldots, 4^i, \ldots. \end{aligned}$$

Isto permite descobrir mais facilmente a solução. Depois usamos o método da substituição para formalizar.

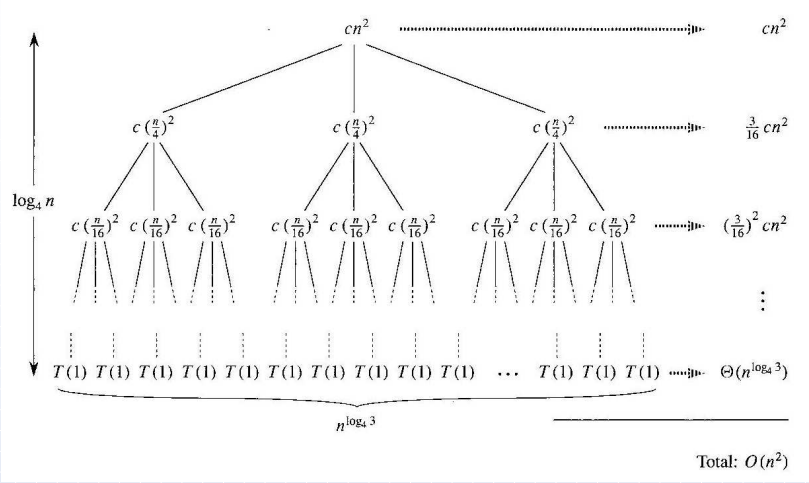
Árvore de recorrência



Árvore de recorrência



Árvore de recorrência



Árvore de recorrência

- O número de níveis é $\log_4 n + 1$.
- No nível i o tempo gasto (sem contar as chamadas recursivas) é $(3/16)^i cn^2$.
- No último nível há $3^{\log_4 n} = n^{\log_4 3}$ folhas. Como $T(1) = \Theta(1)$ o tempo gasto é $\Theta(n^{\log_4 3})$.

Árvore de recorrência

Logo,

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 + \cdots + \\ &\quad + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4 3}) \\ &\leq \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4 3}) = \frac{16}{13}cn^2 + \Theta(n^{\log_4 3}), \end{aligned}$$

e $T(n) \in O(n^2)$.

Árvore de recorrência

Mas $T(n) \in O(n^2)$ é realmente a solução da recorrência original?

Com base na árvore de recorrência, chutamos que $T(n) \leq dn^2$ para alguma constante $d > 0$.

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

onde a última desigualdade vale se $d \geq (16/13)c$.
(Yeeesssss!)

Árvore de recorrência

Resumo

- O número de nós em cada nível da árvore é o número de chamadas recursivas.
- Em cada nó indicamos o "tempo" ou "trabalho" gasto naquele nó que **não** corresponde a chamadas recursivas.
- Na coluna mais à direita indicamos o tempo total naquele nível que **não** corresponde a chamadas recursivas.
- Somando ao longo da coluna determina-se a solução da recorrência.

Vamos tentar juntos?

Eis um exemplo um pouco mais complicado.

Vamos resolver a recorrência

$$\begin{aligned} T(n) &= 1 && \text{para } n = 1, 2, \\ T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n && \text{para } n \geq 3. \end{aligned}$$

Qual é a solução da recorrência?

Resposta: $T(n) \in O(n \lg n)$. (Resolvido em aula)

Recorrências com O à direita (CLRS)

Uma "recorrência"

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

representa todas as recorrências da forma

$$\begin{aligned} T(n) &= a && \text{para } n = 1, 2, \\ T(n) &= 3T(\lfloor n/4 \rfloor) + bn^2 && \text{para } n \geq 3 \end{aligned}$$

onde a e $b > 0$ são constantes.

As soluções exatas dependem dos valores de a e b, mas estão todas na mesma classe Θ.

A "solução" é $T(n) = \Theta(n^2)$, ou seja, $T(n) \in \Theta(n^2)$.

As mesmas observações valem para as classes O, Ω, o, ω.

Recorrência do Mergesort

Podemos escrever a recorrência de tempo do Mergesort da seguinte forma

$$\begin{aligned} T(1) &= \Theta(1) \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) && \text{para } n \geq 2. \end{aligned}$$

A solução da recorrência é $T(n) = \Theta(n \lg n)$.

A prova é essencialmente a mesma do primeiro exemplo. (Exercício!)

Cuidados com a notação assintótica

A notação assintótica é muito versátil e expressiva. Entretanto, deve-se tomar alguns cuidados.

Considere a recorrência

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

É similar a recorrência do Mergesort!

Mas eu vou "provar" que $T(n) = O(n)$!

Cuidados com a notação assintótica

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

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

Por quê?

Não foi feito o passo indutivo, ou seja, não foi mostrado que $T(n) \leq cn$.

Teorema Master

- Veremos agora um resultado que descreve soluções para recorrências da forma

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

  onde $a \geq 1$ e $b > 1$ são constantes.
- Isso é, convenciona-se para alguma constante n_0, vale $T(n) = aT(n/b) + f(n)$ para todo $n \geq n_0$.
- A expressão n/b pode indicar tanto $\lfloor n/b \rfloor$ quanto $\lceil n/b \rceil$.
- O Teorema Master **não** fornece a resposta para todas as recorrências da forma acima.

Teorema Master

Teorema (**Master (CLRS)**)

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 $T(n)$ pode ser limitada assintoticamente da seguinte maneira:

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 para n suficientemente grande, então $T(n) \in \Theta(f(n))$*

Exemplos de Recorrências

Exemplos onde o Teorema Master 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:
  $T(n) = T(3n/4) + n \log n$

Exemplos de Recorrências

Exemplos onde o Teorema Master **não se aplica**:

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

Projeto de algoritmos por indução

Projeto de algoritmos por indução

- A seguir, usaremos a técnica de indução para desenvolver algoritmos para certos problemas.
- Isto é, a formulação do algoritmo vai ser análoga ao desenvolvimento de uma demonstração por indução.
- Assim, para resolver o problema P falamos do projeto de um algoritmo em dois passos:
  1. Exibir a resolução de uma ou mais instâncias pequenas de P (casos base);
  2. Exibir como a solução de toda instância de P pode ser obtida a partir da solução de uma ou mais instâncias menores de P.

Projeto de algoritmos por indução

Este processo indutivo resulta em algoritmos recursivos, em que:

- a base da indução corresponde à resolução dos casos base da recursão;
- a aplicação da hipótese de indução corresponde a uma ou mais chamadas recursivas; e
- o passo da indução corresponde ao processo de obtenção da resposta para o caso geral a partir daquelas devolvidas pelas chamadas recursivas.

Projeto de algoritmos por indução

- Um benefício imediato é que o uso (correto) da técnica nos dá uma prova da corretude do algoritmo.
- A complexidade do algoritmo resultante é expressa numa recorrência
- Frequentemente o algoritmo é eficiente, embora existam exemplos simples em que isso não acontece.
- Iniciaremos com dois exemplos que usam indução:
  1. cálculo do valor de polinômios e
  2. obtenção de subgrafos maximais com restrições de grau.

Exemplo 1 - Cálculo de polinômios

Problema:

Dada uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$, *e um número real* x, *calcular o valor do polinômio*

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0.$$

Naturalmente este é um problema bem simples.
Estamos interessados em projetar um algoritmo que faça o menor número de operações aritméticas (multiplicações, principalmente).

Exemplo 1 - Cálculo de polinômios

Problema:

Dados uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$, *e um número real* x, *calcular o valor do polinômio*

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0.$$

Hipótese de indução: (primeira tentativa)

Dados uma sequência de números reais $a_{n-1}, \ldots, a_1, a_0$, e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$.

- **Caso base**: $n = 0$. A solução é a_0.
- Para calcular $P_n(x)$, basta calcular x^n, multiplicar o resultado por a_n e somar o resultado com $P_{n-1}(x)$.

Exemplo 1 - Solução 1 - Algoritmo

CálculoPolinômio(A,x)

▷ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.
▷ **Saída:** O valor de $P_n(x)$.
1. **se** $n = 0$ **então** $P \leftarrow a_0$
2. **senão**
3. $A' \leftarrow a_{n-1}, \ldots, a_1, a_0$
4. $P' \leftarrow$ CálculoPolinômio(A', x)
5. $xn \leftarrow 1$
6. **para** $i \leftarrow 1$ **até** n **faça** $xn \leftarrow xn * x$
7. $P \leftarrow P' + a_n * xn$
8. **devolva** (P)

Exemplo 1 - Solução 1 - Complexidade

Chamando de $T(n)$ o número de operações aritméticas realizadas pelo algoritmo, temos a seguinte recorrência para $T(n)$:

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Não é difícil ver que

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{ adição}] \\ &= (n+3)n/2 \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Nota: o número de multiplicações pode ser diminuído calculando x^n com o algoritmo de exponenciação rápida que veremos mais tarde.

Segunda solução indutiva

- Desperdício cometido na primeira solução: recálculo de potências de x.
- Alternativa: eliminar essa computação desnecessária trazendo o cálculo de x^{n-1} para dentro da hipótese de indução.

Hipótese de indução reforçada:

Sabemos calcular o valor de $P_{n-1}(x) = a_{n-1}x^{n-1} + \ldots + a_1x + a_0$ e também o valor de x^{n-1}.

- Então, no passo de indução, primeiro calculamos x^n multiplicando x por x^{n-1}, conforme exigido na hipótese. Em seguida, calculamos $P_n(x)$ multiplicando x^n por a_n e somando o valor obtido com $P_{n-1}(x)$.
- Note que para o caso base $n = 0$, a solução agora é $(a_0, 1)$.

Exemplo 1 - Solução 2 - Algoritmo

CálculoPolinômio(A,x)

▷ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.
▷ **Saída:** O valor de $P_n(x)$ e o valor de x^n.
1. **se** $n = 0$ **então** $P \leftarrow a_0; xn \leftarrow 1$
2. **senão**
3. $A' \leftarrow a_{n-1}, \ldots, a_1, a_0$
4. $P', x' \leftarrow$ CálculoPolinômio(A', x)
5. $xn \leftarrow x * x'$
6. $P \leftarrow P' + a_n * xn$
7. **devolva** (P, xn)

Exemplo 1 - Solução 2 - Complexidade

Novamente, se $T(n)$ é o número de operações aritméticas realizadas pelo algoritmo, $T(n)$ é dado por:

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2 \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução da recorrência é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (2 \text{ multiplicações} + 1 \text{ adição}) \\ &= 2n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Terceira solução indutiva

- A escolha de considerar o polinômio $P_{n-1}(x)$ na hipótese de indução não é a única possível.
- Podemos **reforçar** ainda mais a h.i. e ter um ganho de complexidade:

Hipótese de indução mais reforçada:

Sabemos calcular o valor do polinômio $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} \ldots + a_1$.

- Note que $P_n(x) = xP'_{n-1}(x) + a_0$. Assim, bastam uma multiplicação e uma adição para obtermos $P_n(x)$ a partir de $P'_{n-1}(x)$.
- O caso base é trivial pois, para $n = 0$, a solução é a_0.

Exemplo 1 - Solução 3 - Algoritmo

CálculoPolinômio(A,x)

▷ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.
▷ **Saída:** O valor de $P_n(x)$.
1. **se** $n = 0$ **então** $P \leftarrow a_0$
2. **senão**
3. $A' \leftarrow a_n, a_{n-1}, \ldots, a_1$
4. $P' \leftarrow$ CálculoPolinômio(A', x)
5. $P \leftarrow x * P' + a_0$
6. **devolva** (P)

Exemplo 1 - Solução 3 - Complexidade

Temos que $T(n)$ é dado por

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 1 \text{ multiplicação} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (1 \text{ multiplicação} + 1 \text{ adição}) \\ &= n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Essa forma de calcular $P_n(x)$ é chamada de **regra de Horner**.

Projeto por indução - Exemplo 2
Subgrafos Maximais

- Suponha que você esteja planejando uma festa onde queira maximizar as interações entre as pessoas. Uma festa animada, mas sem recursos ilícitos para aumentar a animação.
- Uma das maneiras de conseguir isso é fazer uma lista dos possíveis convidados e, para cada um desses, a lista dos convidados com quem ele(a) interagiria durante a festa.
- Em seguida, é só localizar o maior subgrupo de convidados que interagiriam com, no mínimo, digamos, k outros convidados dentro do subgrupo.

Exemplo 2 - Subgrafos Maximais

Formulação do problema usando grafos:

Dado um grafo simples G (não direcionado) com n vértices e um inteiro $k \leq n$, encontrar em G um subgrafo induzido H com o número máximo de vértices, tal que o grau de cada vértice de H seja $\geq k$. Se um tal subgrafo H não existir, o algoritmo deve identificar esse fato.

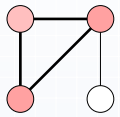
Definições:

H é um subgrafo induzido de um grafo G, uma aresta de G está em H se, e somente se, tem ambos os seus extremos em H.

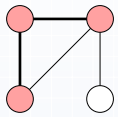
Um subgrafo qualquer G' de G é maximal com relação a uma propriedade P se G' satisfizer P e não existir em G outro subgrafo próprio que contenha G' e satisfaça P.

Exemplo 2 (cont.)

- Exemplo de um subgrafo que é induzido e de outro subgrafo que não é induzido em um grafo G:



INDUZIDO



NÃO INDUZIDO

- Como exemplo de subgrafos maximais (propriedade relativa à continência) que não são máximos (propriedade relativa ao tamanho) em um grafo G, veja definição de clique em grafos.

- No nosso exemplo, o conceito de *maximal* é equivalente ao de *máximo* (número de vértices).

  Você pode provar esta afirmativa ?

Exemplo 2 - Indução

Usando o método indutivo:

Hipótese de indução:

Dados inteiros n, k e um grafo G com menos que n vértices, sabemos como encontrar um subgrafo maximal H de G com grau mínimo $\geq k$.

Caso base: o primeiro valor de n para o qual faz sentido buscarmos tais subgrafos H é $n = k + 1$, caso contrário não há como satisfazer a restrição de grau mínimo.

Assim, quando $n = k + 1$, a única forma de existir em G um subgrafo induzido com grau mínimo k é que **todos** os vértices tenham grau igual a k.

Se isso ocorrer, a resposta é $H = G$; caso contrário $H = \emptyset$.

Exemplo 2 - Indução (cont.)

- Seja então G um grafo com n vértices e $k \geq 0$ um inteiro tal que $n > k + 1$.
- Se todos os vértices de G têm grau $\geq k$ não há nada mais a fazer. Devolva $H = G$.
- Caso contrário, seja v um vértice com grau $< k$. É fácil ver que nenhum subgrafo que é solução para o problema conterá v. Assim, v pode ser removido e a hipótese de indução aplicada ao grafo resultante G'.
- Seja H' o subgrafo devolvido pela aplicação da h.i. em G'. Então H' também é resposta para G. ■

Exemplo 2 - Algoritmo

SubgrafoMaximal(G, k)

▷ **Entrada:** Grafo G com n vértices e um inteiro $k \geq 0$.
▷ **Saída:** Subgrafo induzido H de G com grau mínimo k.
1. **se** $(n < k + 1)$ **então** $H \leftarrow \emptyset$
2. **senão se** todo vértice de G tem grau $\geq k$
3. **então** $H \leftarrow G$
4. **senão**
5. seja v um vértice de G com grau $< k$
6. $H \leftarrow$ SubgrafoMaximal$(G - v, k)$
7. **devolva** H

Projeto por indução - Exemplo 3
Fatores de balanceamento em árvores binárias

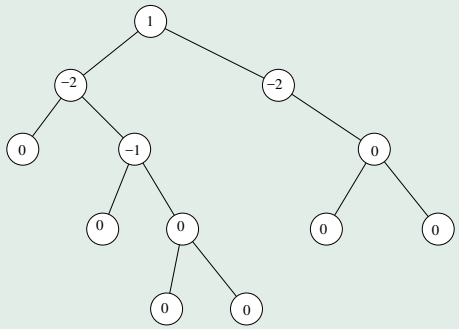
Definição:

Árvores binárias balanceadas são estruturas de dados que minimizam o tempo de busca de informações nela armazenadas. A idéia é que, para todo nó v da árvore, o *fator de balanceamento (f.b.)* de v, isto é, a diferença entre a altura da subárvore esquerda e a altura da subárvore direita de v não desvie muito de zero.

- Convenciona-se que a árvore vazia tem fator de balanceamento *zero* e altura igual a -1.
- Árvores AVL são exemplos de árvores binárias balanceadas, em que o f.b. de cada nó é $-1, 0$ ou $+1$.

Exemplo 3 (cont.)

Exemplo de uma árvore binária e os fatores de balanceamento de seus nós.



Exemplo 3 (cont.)

Problema:

Dada uma árvore binária A com n nós, calcular os fatores de balanceamento de cada nó de A.

- Vamos projetar o algoritmo indutivamente.

Hipótese de indução:

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

- **Caso base**: quando $n = 0$, convencionamos que o f.b. é igual a zero.
- Vamos mostrar agora como usar a hipótese para calcular f.b.s de uma árvore A com exatamente n nós.

Exemplo 3 - indução (cont.)

A idéia é aplicar a h.i. às subárvores esquerda e direita da raiz e, em seguida, calcular o f.b. da raiz.

Dificuldade: o f.b. da raiz depende das alturas das subárvores esquerda e direita e não dos seus f.b.s.

Conclusão: é necessário uma h.i. mais forte!

Nova hipótese de indução:

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Exemplo 3 - indução (cont.)

- Novamente, o caso base $n = 0$ é fácil pois, por convenção, o f.b. é nulo e a altura igual a -1.
- Seja A uma árvore binária com n nós, para um $n > 0$ qualquer.

  Sejam (f_e, h_e) e (f_d, h_d) os f.b.s e alturas das subárvores esquerda (A_e) e direita (A_d) de A que, por h.i., sabemos como calcular.

  Então, o f.b. da raiz de A é $h_e - h_d$ e a altura da árvore é $\max(h_e, h_d) + 1$.

  Isto completa o cálculo dos f.b.s e da altura de A. ■

De novo, o fortalecimento da hipótese tornou a resolução do problema mais fácil.

Exemplo 3 - Algoritmo

FatorAltura (A)

```
▷ Entrada: Uma árvore binária A com n ≥ 0 nós
▷ Saída: A árvore A os f.b.s nos seus nós e a altura de A
1.  se n = 0 então h ← −1
2.  senão
3.      seja r a raiz de A
3.      h_e ← FatorAltura(A_e)
4.      h_d ← FatorAltura(A_d)
5.      ▷ armazena o f.b. na raiz em r.f
6.      r.f ← h_e − h_d
7.      h ← max(h_e, h_d) + 1
8.  devolva h
```

Exemplo 3 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular os f.b.s e a altura de uma árvore A de n nós. Então

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n_e) + T(n_d) + \Theta(1), & n > 0, \end{cases}$$

onde n_e, n_d são os números de nós das subárvores esquerda e direita.

Exemplo 3 - Complexidade

O pior caso da recorrência parece ser quando, ao longo da recursão, um de n_e, n_d é sempre igual a zero (e o outro é sempre igual a $n-1$). Isto é, quando $T(n) = T(n-1) + T(0) + \Theta(1)$.

Chega-se então a:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(1) \\ &= T(n-2) + 2T(0) + 2\Theta(1) \\ &\vdots \\ &= T(0) + nT(0) + n\Theta(1) \\ &= (n+1)\Theta(1) + n\Theta(1) \in \Theta(n). \end{aligned}$$

Exercício:

Há algum ganho em complexidade quando ambos n_e e n_d são aproximadamente $n/2$ ao longo da recursão?

Projeto por Indução - Exemplo 4: o problema da celebridade

Definição

Num conjunto S de n pessoas, uma *celebridade* é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém. (Celebridades são pessoas de difícil convívio...).

Note que pode existir no máximo uma celebridade em S!

Problema:

Queremos saber se existe uma celebridade em S.

Projeto por Indução - Exemplo 4: o problema da celebridade

Vamos formalizar melhor: para um conjunto de n pessoas, associamos uma matriz $n \times n$ M tal que $M[i,j] = 1$ se a pessoa i conhece a pessoa j e $M[i,j] = 0$ caso contrário.

Problema:

Dado um conjunto de n pessoas e a matriz associada M encontrar (se existir) uma celebridade no conjunto.
Isto é, determinar um k tal que todos os elementos da coluna k (exceto $M[k,k]$) são 1s e todos os elementos da linha k (exceto $M[k,k]$) são 0s.

Existe uma solução simples mas laboriosa: para cada pessoa i, verifique todos os outros elementos da linha i e da coluna i. O custo dessa solução é $2n(n-1)$.

Exemplo 4 - Indução

Um argumento indutivo que parece ser mais eficiente é o seguinte:

Hipótese de Indução:

Sabemos encontrar uma celebridade (se existir) em um conjunto de $n-1$ pessoas.

- Se $n = 1$, podemos considerar que o único elemento é uma celebridade.
- Outra opção seria considerarmos o caso base como $n = 2$, o primeiro caso interessante.
  A solução é simples: existe uma celebridade se, e somente se, $M[1,2] \oplus M[2,1] = 1$. Mais uma comparação define a celebridade: se $M[1,2] = 0$, então a celebridade é a pessoa 1; se não, é a pessoa 2.

Exemplo 4 - Indução (cont.)

Tome então um conjunto $S = \{1, 2, \ldots, n\}, n > 2$, de pessoas e a matriz M associada. Considere o conjunto $S' = S \setminus \{n\}$;

Há dois casos possíveis:

1. Existe uma celebridade em S', digamos a pessoa k; então, k é celebridade em S se, e somente se, $M[n,k] = 1$ e $M[k,n] = 0$.
2. Se não existir celebridade em S', então a pessoa n é celebridade em S se $M[n,j] = 0$ e $M[j,n] = 1, \forall j < n$; caso contrário não há celebridade em S.

Essa primeira tentativa, infelizmente, também conduz a um algoritmo quadrático. Por quê?

Exemplo 4 - Segunda tentativa

A segunda tentativa baseia-se em um fato muito simples:

*Dadas duas pessoas i e j, é possível determinar se uma delas **não** é uma celebridade com apenas uma comparação: se $M[i,j] = 1$, então i não é celebridade; caso contrário j não é celebridade.*

Vamos usar esse argumento aplicando a hipótese de indução sobre o conjunto de $n-1$ pessoas obtidas removendo dentre as n uma pessoa que sabemos não ser celebridade.

- O caso base e a hipótese de indução são os mesmos que anteriormente.

Exemplo 4 - Segunda tentativa

Tome então um conjunto arbitrário de $n > 2$ pessoas e a matriz M associada.

Sejam i e j quaisquer duas pessoas e suponha que, usando o argumento acima, determinemos que j não é celebridade.

Seja $S' = S \setminus \{j\}$ e considere os dois casos possíveis:

1. Existe uma celebridade em S', digamos a pessoa k. Se $M[j,k] = 1$ e $M[k,j] = 0$, então k é celebridade em S; caso contrário não há uma celebridade em S.
2. Não existe uma celebridade em S'; então não existe uma celebridade em S.

Exemplo 4 - Algoritmo

Celebridade(S, M)

▷ **Entrada:** conjunto de pessoas $S = \{1, 2, \ldots, n\}$;
M, a matriz que define quem conhece quem em S.
▷ **Saída:** Um inteiro $k \leq n$ que é celebridade em S ou $k = 0$

1. **se** $|S| = 1$ **então** $k \leftarrow$ elemento em S
2. **senão**
3. sejam i, j quaisquer duas pessoas em S
4. **se** $M[i, j] = 1$ **então** $s \leftarrow i$ **senão** $s \leftarrow j$
5. $S' \leftarrow S \setminus \{s\}$
6. $k \leftarrow$ Celebridade(S', M)
7. **se** $k > 0$ **então**
8. **se** $(M[s, k] \neq 1)$ **ou** $(M[k, s] \neq 0)$ **então** $k \leftarrow 0$
9. **devolva** k

Exemplo 4 - Complexidade

O algoritmo resultante tem complexidade linear em n.

A recorrência $T(n)$ para o número de operações executadas pelo algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Projeto por indução - Exemplo 5
Subsequência consecutiva máxima (SCM)

Problema:

Dada uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais (não necessariamente positivos), encontrar uma **subsequência consecutiva** *$Y = x_i, x_{i+1}, \ldots, x_j$ de X, onde $1 \leq i, j \leq n$, cuja soma seja máxima dentre todas as subsequências consecutivas.*

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

Exemplo 5 - Indução

Como antes, vamos examinar o que podemos obter de uma **hipótese de indução** simples:

Hipótese de indução:

Sabemos calcular a SCM *de sequências de comprimento $n - 1$.*

- Seja então $X = x_1, x_2, \ldots, x_n$ uma sequência qualquer de comprimento $n > 1$.
- Considere a sequência X' obtida de X removendo-se x_n.
- Seja $Y' = x_i, x_{i+1}, \ldots, x_j$ a SCM de X', obtida aplicando-se a h.i.

Exemplo 5 - Indução

Há três casos a examinar:

1. $Y' = [\ \ ]$. Neste caso, $Y = x_n$ se $x_n \geq 0$ ou $Y = [\ \ ]$ se $x_n < 0$.
2. $j = n - 1$. Como no caso anterior, temos $Y = Y'||x_n$ se $x_n \geq 0$ ou $Y = Y'$ se $x_n < 0$.
3. $j < n - 1$. Aqui há dois subcasos a considerar:
   - 3.1 Y' também é SCM de X; isto é, $Y = Y'$.
   - 3.2 Y' não é a SCM de X. Isto significa que x_n é parte de uma SCM Y de X. Esta tem que ser da forma $x_k, x_{k+1}, \ldots, x_{n-1}, x_n$, para algum $k \leq n - 1$.

Exemplo 5 - Indução

- A hipótese de indução nos permite resolver todos os casos anteriores, exceto o último.
  Não há informação suficiente na h.i. para permitir a resolução deste caso.
  O que falta na h.i.?
- É evidente que, quando

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

  então $x_k, x_{k+1}, \ldots, x_{n-1}$ é um *sufixo* de X' de soma máxima entre os *sufixos* de X'.
- Assim, se conhecermos o sufixo máximo de X', além da SCM, teremos resolvido o problema completamente para X.

Exemplo 5 - Indução

Parece então natural enunciar a seguinte h.i. fortalecida:

Hipótese de indução reforçada:

Sabemos calcular a SCM e o sufixo máximo de sequências de comprimento $n - 1$.

É clara desta discussão também, a base da indução: para $n = 1$, a SCM de $X = x_1$ é x_1 caso $x_1 \geq 0$, e a sequência vazia caso contrário. Nesse caso, o sufixo máximo é igual a SCM.

Exemplo 5 - Algoritmo

MSC(X, n)

- **Entrada:** um inteiro n e uma sequência de n números reais $X = [x_1, x_2, \ldots, x_n]$.
- **Saída:** Inteiros i, j, k e reais $MaxSeq, MaxSuf$ tais que:
  - x_i, x_j são o primeiro e último elementos da SCM de X, cujo valor é $MaxSeq$; e
  - x_k é o primeiro elemento do sufixo máximo de X, cujo valor é $MaxSuf$.
  - O valor $j = 0$ significa que X é composta de negativos somente. Neste caso, convencionamos $MaxSeq = 0$.
  - O valor $k = 0$ significa que o sufixo máximo de X é vazio. Neste caso, $MaxSuf = 0$.

Exemplo 5 - Algoritmo (cont.)

SCM(X, n): (cont.)

```
1.  se n = 1
2.  então
3.     se x_1 < 0
4.        então i, j, k ← 0; MaxSeq, MaxSuf ← 0
5.        senão i, j, k ← 1; MaxSeq, MaxSuf ← x_1
6.  senão
7.     (i, j, k, MaxSeq, MaxSuf) ← SCM(X, n − 1)
8.     se MaxSuf = 0 então k ← n
9.     MaxSuf ← MaxSuf + x_n
10.    se MaxSuf > MaxSeq
11.       então i ← k; j ← n; MaxSeq ← MaxSuf
12.       senão se MaxSuf < 0 então MaxSuf ← 0; k ← 0
13.  devolva (i, j, k, MaxSeq, MaxSuf)
```


Exemplo 5 - Complexidade

A complexidade $T(n)$ de SCM é simples de ser calculada.

Como no último exemplo,

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Reforçar a hipótese de indução: preciso lembrar disso ...

Projeto por indução - Erros comuns

Os erros discutidos nas provas por indução, naturalmente, traduzem-se em erros no projeto de um algoritmo por indução.

Exemplo:

Problema: *Dado um grafo conexo G, verificar se G é bipartido ou não. Caso seja, devolver a partição dos vértices.*

Definição:

Um grafo é *bipartido* se seu conjunto de vértices pode ser particionado em dois conjuntos, de forma que toda aresta de G tenha extremos em conjuntos diferentes.

Teorema:

Se G é conexo e bipartido, então a bipartição é única.

Projeto por indução - Erros comuns

O que há de **errado** com o seguinte (esboço de um) algoritmo recursivo para verificar se um grafo conexo é bipartido?

- Sejam G um grafo conexo, v um vértice de G e considere o grafo $G' = G - \{v\}$.
- Se G' não for bipartido, então G também não é. Caso contrário, sejam A e B os dois conjuntos da bipartição de G' obtidos recursivamente.
- Considere agora o vértice v e sua relação com os vértices de A e B.
- Se v tiver um vizinho em A e outro em B, então G não é bipartido (já que a bipartição, se existir, deve ser única).
- Caso contrário, adicione v a A ou B, o conjunto no qual v não tem vizinhos. A bipartição está completa.

Divisão e Conquista

Projeto de Algoritmos por Divisão e Conquista

- **Revisando:** Um algoritmo de divisão e conquista resolve um problema:
  - obtendo soluções para subproblemas recursivamente;
  - combinando as soluções parciais de (um ou mais)
- É mais um paradigma de projeto de algoritmos baseado no princípio da indução.
- Informalmente podemos dizer que:
  - o paradigma incremental utiliza indução fraca
  - o paradigma de divisão e conquista utiliza indução forte
- É natural, portanto, demonstrar a corretude de algoritmos de divisão e conquista por indução.

Algoritmo Genérico

DivisaoConquista(x)

▷ **Entrada:** A instância x
▷ **Saída:** Solução y do problema em questão para x
1. **se** x é suficientemente pequeno **então**
   ▷ *Solucao*(x) algoritmo para pequenas instâncias
2. **retorne** *Solucao*(x)
3. **senão**
   ▷ divisão
4. decomponha x em instâncias menores $x_1, x_2, \cdots, x_k$
5. **para** i **de** 1 **até** k **faça** $y_i := \mathit{DivisaoConquista}(x_i)$
   ▷ conquista
6. combine as soluções y_i para obter a solução y de x.
7. **retorne**(y)

Projeto por Divisão e Conquista - Exemplo 1 Exponenciação

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Primeira solução, por indução fraca:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro* $n > 0$ *e real* a, *sei calcular* a^{n-1}.
- **Passo da indução:** Queremos provar que conseguimos calcular a^n, para $n > 0$. Por hipótese de indução, sei calcular a^{n-1}. Então, calculo a^n multiplicando a^{n-1} por a.

Exemplo 1 - Solução 1 - Algoritmo

Exponenciacao(a, n)

▷ **Entrada:** A base a e o expoente n.
▷ **Saída:** O valor de a^n.
1. **se** $n = 0$ **então**
2. **retorne**(1) {caso base}
3. **senão**
4. $an' := Exponenciacao(a, n-1)$
5. $an := an' * a$
6. **retorne**(an)

Exemplo 1 - Solução 1 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} c_1, & n = 0 \\ T(n-1) + c_2, & n > 0, \end{cases}$$

onde c_1 e c_2 representam, respectivamente, o tempo (constante) executado na atribuição da base e multiplicação do passo.

Neste caso, não é difícil ver que

$$T(n) = c_1 + \sum_{i=1}^{n} c_2 = c_1 + nc_2 = \Theta(n).$$

Este algoritmo é linear no tamanho da entrada ?

Exemplo 1 - Solução 2 - Divisão e Conquista

Vamos agora projetar um algoritmo para o problema usando indução forte de forma a obter um algoritmo de divisão e conquista.

Segunda solução, por indução forte:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro $n > 0$ e real a, sei calcular a^k, para todo $k < n$.*
- **Passo da indução:** Queremos provar que conseguimos calcular a^n, para $n > 0$. Por hipótese de indução sei calcular $a^{\lfloor \frac{n}{2} \rfloor}$. Então, calculo a^n da seguinte forma:

$$a^n = \begin{cases} \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ par;} \\ a \cdot \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ ímpar.} \end{cases}$$

Exemplo 1 - Solução 2 - Algoritmo

ExponenciacaoDC(a, n)

▷ **Entrada:** A base a e o expoente n.
▷ **Saída:** O valor de a^n.
1. **se** $n = 0$ **então**
2. **retorne**(1) {caso base}
3. **senão**
▷ divisão
4. $an' := ExponenciacaoDC(a, n \ div \ 2)$
▷ conquista
5. $an := an' * an'$
6. **se** $(n \ mod \ 2) = 1$
7. $an := an * a$
8. **retorne**(an)

Exemplo 1 - Solução 2 - Complexidade

- Seja $T(n)$ o número de operações executadas pelo algoritmo de divisão e conquista para calcular a^n.
- Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} c_1, & n = 0 \\ T(\lfloor \frac{n}{2} \rfloor) + c_2, & n > 0, \end{cases}$$

- Não é difícil ver que $T(n) \in \Theta(\log n)$. Por quê?

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Dado um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que $A[i] = x$, ou que não existe tal i.

- O projeto de um algoritmo para este problema usando indução simples, nos leva a um algoritmo incremental de complexidade de pior caso $\Theta(n)$. Pense em como seria a indução !
- Se utilizarmos indução forte para projetar o algoritmo, podemos obter um algoritmo de divisão e conquista que nos leva ao algoritmo de busca binária. Pense na indução !
- Como o vetor está ordenado, conseguimos determinar, com apenas uma comparação, que *metade* das posições do vetor não pode conter o valor x.

Exemplo 2 - Algoritmo

BuscaBinaria(A, e, d, x)

▷ **Entrada:** Vetor A, delimitadores e e d do subvetor e x.
▷ **Saída:** Índice $1 \leq i \leq n$ tal que $A[i] = x$ ou $i = 0$.

```
1.  se e = d então se A[e] = x então retorne(e)
2.        senão retorne(0)
3.  senão
4.     i := (e + d) div 2
5.     se A[i] = x retorne(i)
6.     senão se A[i] > x
7.        i := BuscaBinaria(A, e, i − 1, x)
8.        senão {A[i] < x}
9.        i := BuscaBinaria(A, i + 1, d, x)
10. retorne(i)
```


Exemplo 2 - Complexidade

- O número de operações $T(n)$ executadas na busca binária no pior caso é:

$$T(n) = \begin{cases} c_1, & n = 1 \\ T(\lceil \frac{n}{2} \rceil) + c_2, & n > 1, \end{cases}$$

- Não é difícil ver que $T(n) \in \Theta(\log n)$. Por quê?
- O algoritmo de busca binária (divisão e conquista) tem complexidade de pior caso $\Theta(\log n)$, que é assintoticamente melhor que o algoritmo de busca linear (incremental).
- E se o vetor não estivesse ordenado, qual paradigma nos levaria a um algoritmo assintoticamente melhor ?

Um exemplo real

De um aluno de mestrado do IC

Entrada:

- m listas L_l tem n_l inteiros ordenados do menor para maior
- número k

Saída:

- k tuplas $(a_1, a_2, \ldots, a_m)$ com menores somas

OBS: soma de uma tupla $(a_1, a_2, \ldots, a_k)$ é $a_1 + a_2 + \cdots + a_m$.

Um exemplo real - Algoritmo

Algoritmo trivial

1. Enumerar todas as tuplas
2. Ordenar elementos
3. Selecionar k primeiros

- **Complexidade:** $\tilde{O}(n_1 \times n_2 \times \cdots \times n_m)$
- Como fazer melhor? divisão e conquista

Um exemplo real - Divisão e conquista

Estratégia

- Caso base:
  - $m = 1$
  - $m = 2$ também!
- Caso geral:
  - Divisão: dividimos em sub-somas (associativa e comutativa!)
  - Conquista: é o mesmo problema, para $m = 2$!

Algoritmo e análise: no quadro