Aula 7

- Continuando no Capítulo 3 do livro, veremos hoje:
  - Recursão / Dividir e conquistar (Seção 3.7)
  - Árvores de decisão (Seção 3.8)
  - Limite inferior para ordenação (Seção 3.9)
- Depois disso, vamos pular para o Capítulo 5
  - programação dinâmica
  - algoritmos gulosos
  - condensação

veremos hoje, se der tempo

RECURSÃO

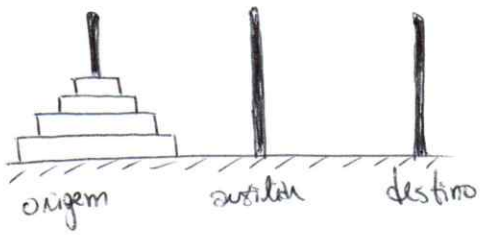
DIVIDIR - Decomposição de um problema dado em subproblemas menores. Estes, por sua vez, em subproblemas ainda menores, assim por diante

conquistar - Resolvidos os subproblemas, aplica-se uma composição das soluções, até resolver o problema original.

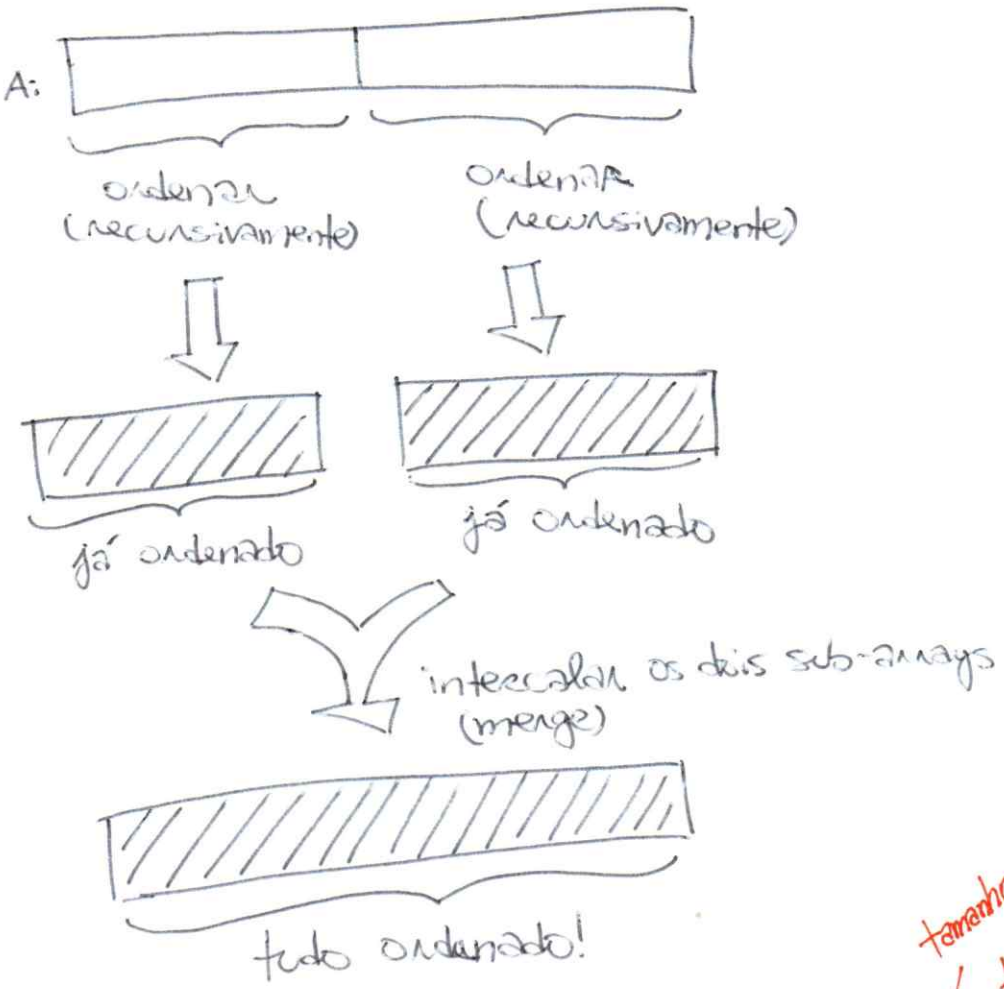
Exemplo

Algoritmo hanoi
dados: m, origem, destino, auxiliar

mover m-1 discos de origem para auxiliar
mover 1 disco de origem para destino
mover m-1 discos de auxiliar para destino



Exemplo Mergesort



Algoritmo: mergesort

dados: A, p, R

se $p \leq R$ então

$q \leftarrow \left\lfloor \frac{p+R}{2} \right\rfloor$

mergesort(A, p, q)

mergesort(A, q+1, R)

merge(A, p, q, R)

Ideia do algoritmo merge:

tamanho q-p+1

tamanho R-q

A_1:	2	7	13	20	∞
A_2:	1	9	11	12	∞

qual o menor? → 1

A_1:	2	7	13	20	∞
A_2:	~~1~~	9	11	12	∞

qual o menor? → 1 2

Resultado:

1 2 7 9 11 12 13 20

Complexidade do algoritmo <u>merge</u>:

tempo $O(n)$, em que n é o numero de elementos

Complexidade do <u>mergesort</u>:

$$\begin{cases} T(1) = O(1) \\ T(n) = 2T(n/2) + O(n), \text{ para } n > 1 \end{cases}$$

Há diversas formas de resolver essa relação de recorrência. No final encontramos

$$T(n) = O(n \log n)$$

<u>Teorema Mestre</u>. (Resultado útil para Dividir & Conquistar)

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

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

em que $\frac{n}{b} \equiv \left\lceil \frac{n}{b} \right\rceil$ ou $\frac{n}{b} \equiv \left\lfloor \frac{n}{b} \right\rfloor$.

Então:

1. Se $f(n) = O(n^{\log_b a - \varepsilon})$ para algum $\varepsilon > 0$ constante,

   temos $T(n) = \Theta(n^{\log_b a})$

2. Se $f(n) = \Theta(n^{\log_b a})$,

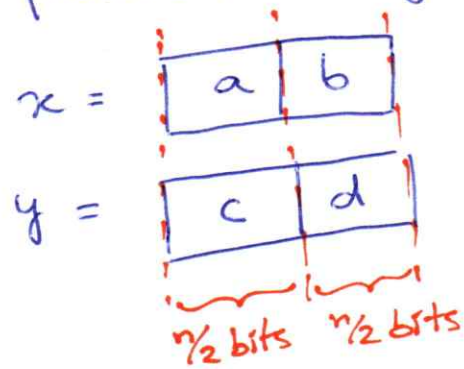
   temos $T(n) = \Theta(n^{\log_b a} \log n)$

3. Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ para algum $\varepsilon > 0$ constante e se $a f(\frac{n}{b}) \leq c f(n)$ para algum $c < 1$ e para todo n sufic. grande,

   temos $T(n) = \Theta(f(n))$

Mais um exemplo — Algoritmo de Karatsuba (multiplicação de inteiros)

- Sejam x, y dois números inteiros de n bits cada
- Por simplicidade, assuma n potência de 2
- Podemos particionar x, y como segue



- Logo, $xy = (a \cdot 2^{n/2} + b)(c \cdot 2^{n/2} + d)$

$$= \underbrace{ac}_{=v} \cdot 2^n + \underbrace{(ad+bc)}_{=(a+b)(c+d)-ac-bd} 2^{n/2} + \underbrace{bd}_{=w}$$

- Então o algoritmo precisa calcular o seguinte

$$\left.\begin{aligned} u &:= (a+b)\cdot(c+d) \\ v &:= a \cdot c \\ w &:= b \cdot d \end{aligned}\right\} \text{3 multiplicações feitas recursivamente, para números de } n/2 \text{ bits}$$

e retornar o resultado

$$v \cdot 2^n + (u - v - w) \cdot 2^{n/2} + w \quad \Big\} \text{ algumas somas e "shifts", em tempo linear}$$

- Por simplicidade, ignore por enquanto o "vai-um" que pode aparecer ao calcular u.
- Complexidade de tempo do algoritmo é

$$T(n) = \begin{cases} k, & \text{para } n = 1 \\ 3\,T(n/2) + kn, & \text{para } n > 1 \end{cases}$$

Pelo Teorema Mestre, $T(n) = O(n^{\log_2 3})$
$= O(n^{1.59})$

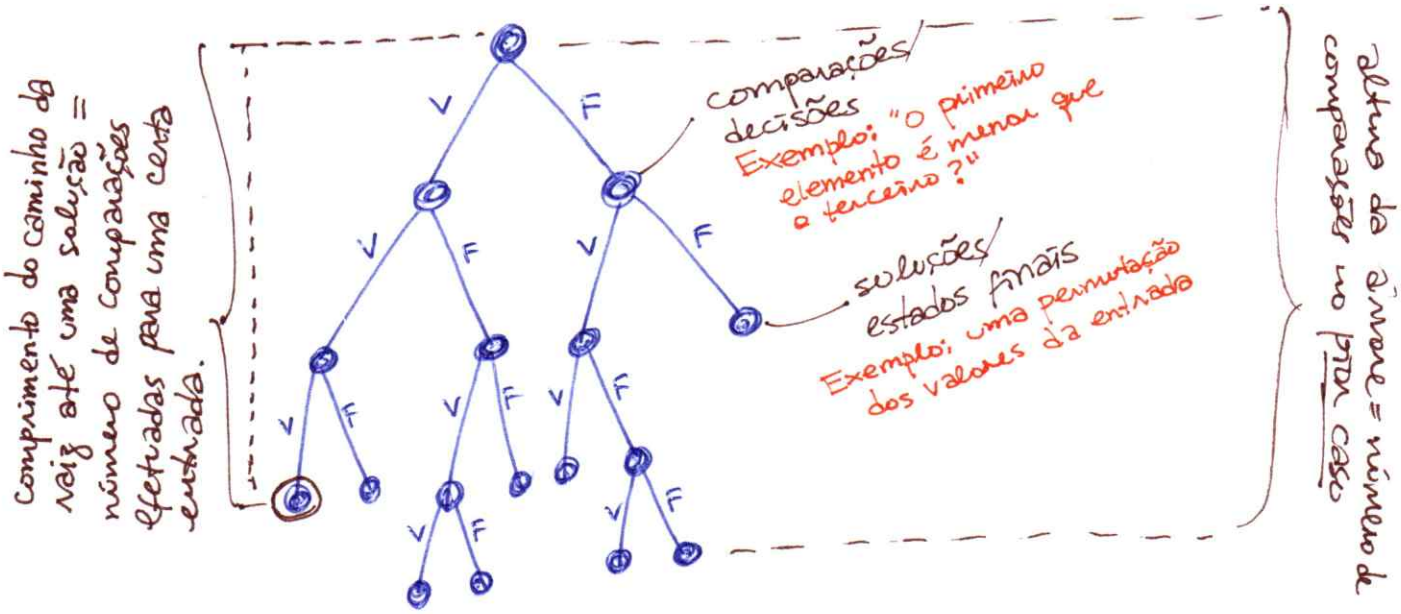
- Algoritmo "ingênuo" é $O(n^2)$.
- Kolmogorov conjecturou $\Omega(n^2)$
- Karatsuba mostrou algoritmo aos 23 anos

ÁRVORES DE DECISÃO

- Técnica útil para, por exemplo, determinar lower bound para certos problemas

- Seja P um problema e α um algoritmo que resolve P, de tal modo que a operação dominante em α seja a comparação. Cada comparação é binária (duas respostas possíveis).

- Árvore de decisão: árvore estritamente binária. Cada comparação efetuada é representada por um vértice interior. Filhos esquerdo e direito correspondem a resultado verdadeiro ou falso da comparação. Folhas correspondem a cada estado final possível.



- Limite inferior para a altura das árvores de decisão de um problema corresponde a um limite inferior (lower bound) para o problema.

LIMITE INFERIOR PARA ORDENAÇÃO

<u>Lema</u> Uma árvore binária de altura h possui no máximo 2^h folhas.

Prova. Indução em h. ■

<u>Lema</u>. $\log(n!) = \Omega(n \log n)$

Prova. Para $n > 1$, temos

$$n! \geq n(n-1)(n-2)\cdots\left(\lceil \tfrac{n}{2} \rceil\right) > \left(\frac{n}{2}\right)^{n/2}$$

Então, $\log n! \geq \frac{n}{2} \log \frac{n}{2}$ ■

- Lembre-se que o número de comparações para o pior caso de um algoritmo de ordenação por comparação é a <u>altura</u> de sua árvore de decisão.

<u>Teorema</u>. Qualquer árvore de decisão que ordene n elementos possui altura pelo menos igual a $\log n!$

Prova. (esboço)
- Resultado da ordenação de n elementos pode ser qualquer uma das $n!$ permutações da entrada
- Então, árvore de decisão tem pelo menos $n!$ folhas
- Logo, $2^h \geq n!$

$$h \geq \log n!$$

$$h = \Omega(n \log n)$$

■

PROGRAMAÇÃO DINÂMICA

- Semelhante a D&C, porém aplicada quando problemas são "sobrepostos" (ou seja, não são subproblemas independentes como na D&C).

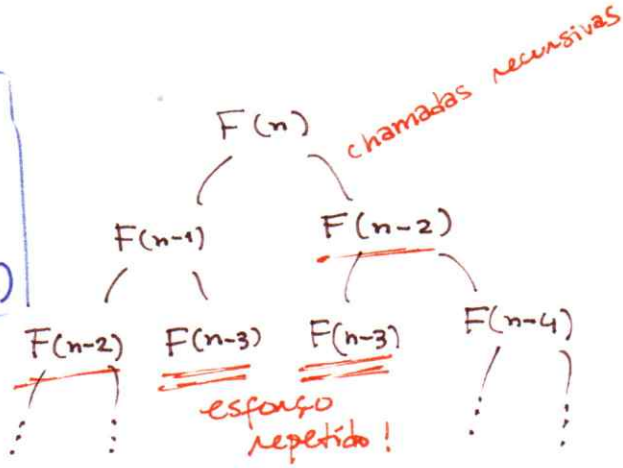
Prog. Dinâmica funciona muito bem quando temos
- subestruturas ótimas
- subproblemas sobrepostos

- Resolve instâncias pequenas primeiro, guarda resultados parciais em uma tabela e procura por eles quando necessário (em vez de recomputar)

- Para alguns problemas, D&C pode ter complexidade exponencial enquanto Prog. Dinâmica polinomial.

Exemplo simples: Fibonacci

```
se n<2 então
    retorna n
caso contrário
    retorna Fib(n-1)+Fib(n-2)
```



Complexidade:

$$\begin{cases} T(1) = O(1) \\ T(n) = T(n-1) + T(n-2) + O(1) \end{cases}$$

Exponencial!

- Uma modificação simples resolveria o problema

Seja F um vetor com n posições, inicializadas com null

```
se n<2 então
    retorna n

se F[n]=null então
    F[n] := Fib(n-1)+Fib(n-2)
retorna F[n]
```

Linear!

Mais um exemplo: Problema da subsequência comum mais longa (LCS)

Definição: Seja uma sequência $X = \langle x_1, x_2, \dots, x_m \rangle$. Uma outra sequência $Z = \langle z_1, z_2, \dots, z_k \rangle$ é uma subsequência de X se existe uma sequência crescente $\langle i_1, i_2, \dots, i_k \rangle$ de índices de X t.q. para todo $j = 1, 2, \dots, k$ temos $x_{i_j} = z_j$.

Exemplo: $X = \langle A, T, G, C, A, T, G, C \rangle$

Então $Z = \langle A, G, C, T \rangle$ é subsequencia de X com índices $\langle 1, 3, 4, 6 \rangle$

Definição: Dadas duas sequências X e Y, dizemos que Z é uma subsequência comum de X e Y se é subsequência de X e de Y simultaneamente.

Exemplo: $X = \langle A, T, G, T, C, A, T \rangle$
$Y = \langle T, C, G, A, T, A \rangle$

Então $\langle T, G, A \rangle$ é subsequência comum de X e Y

$\langle T, G, T, A \rangle$ e $\langle T, C, A, T \rangle$ também são!
Não há LCS de comprimento 5 neste exemplo.

Problema. Entrada: duas sequências $X = \langle x_1, x_2, \dots, x_m \rangle$ e $Y = \langle y_1, y_2, \dots, y_n \rangle$

Saída: - o comprimento de uma LCS
- opcional: uma LCS

Notação: Seja $X = \langle x_1, x_2, \dots, x_m \rangle$. Então $X_j = \langle x_1, x_2, \dots, x_j \rangle$
$X_0 = \langle \ \rangle$

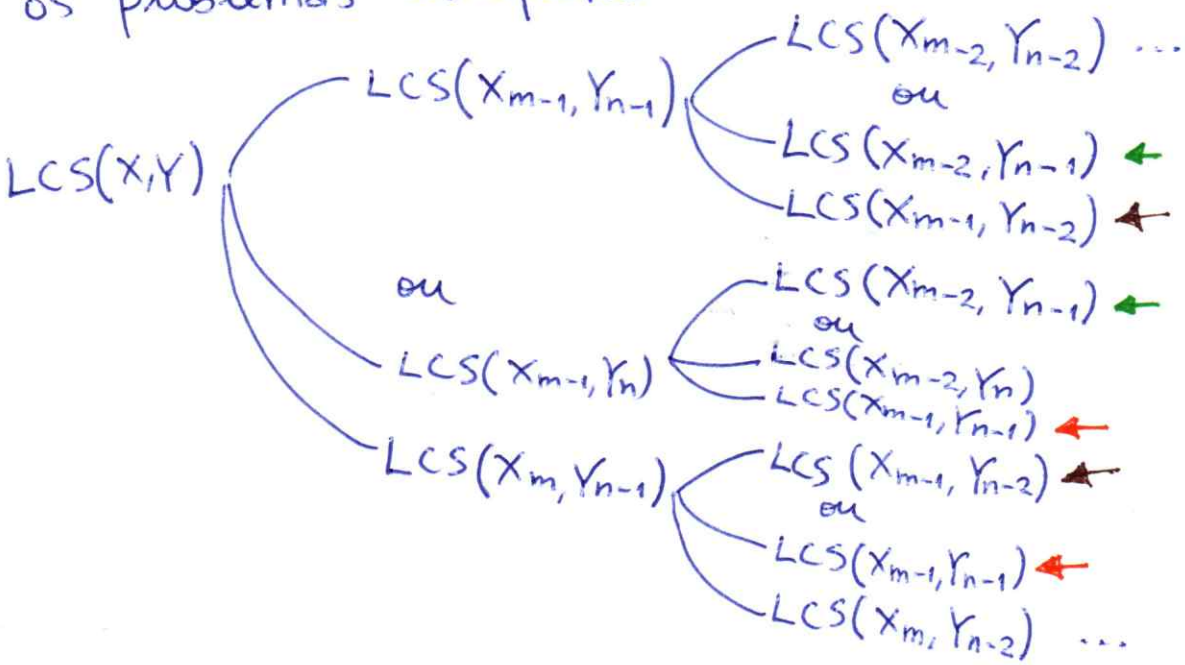
- Queremos encontrar $LCS(X_m, Y_n)$

Então temos que resolver 1 ou 2 subproblemas:

Se $x_m = y_n$ ⟶ $LCS(X_{m-1}, Y_{n-1})$

Se $x_m \neq y_n$ ⟶ $LCS(X_{m-1}, Y)$ ou $LCS(X, Y_{n-1})$

- Note os problemas sobrepostos



- Com isso podemos formular uma expressão recursiva para o comprimento de uma LCS de X e Y.

Seja $c[i,j] = |LCS(X_i, Y_j)|$

Então

$$c[i,j] = \begin{cases} 0, & \text{se } i=0 \text{ ou } j=0 \\ c[i-1, j-1] + 1, & \text{se } i,j>0,\ x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{se } i,j>0,\ x_i \neq y_j \end{cases}$$

<u>Exemplo</u> Entrada: $X = \langle G, T, G, G, A \rangle$

$Y = \langle A, T, G, A, C \rangle$

Vamos criar uma tabela c e preencher de acordo com os critérios da página anterior. Podemos também criar uma tabela (digamos, b) para guardar qual critério foi usado em cada passo e, assim, poder reconstruir uma LCS.

C $\boxed{B}$	$j=0$ y_j	A	T	G	A	C
$i=0$ x_i						
G						
T						
G						
G						
A						

No final, temos:

C	j=0 (y_j)	A	T	G	A	C
i=0 (x_i)	0	0	0	0	0	0
G	0	0 ↑	0 ↑	1 ↖	1 ←	1 ←
T	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑
G	0	0 ↑	1 ↑	2 ↖	2 ←	2 ←
G	0	0 ↑	1 ↑	2 ↖	2 ↑	2 ↑
A	0	1 ↖	1 ↑	2 ↑	3 ↖	3 ←

Então a LCS encontrada é $\langle T, G, A \rangle$, que possui comprimento 3.