

Fibonacci(n)

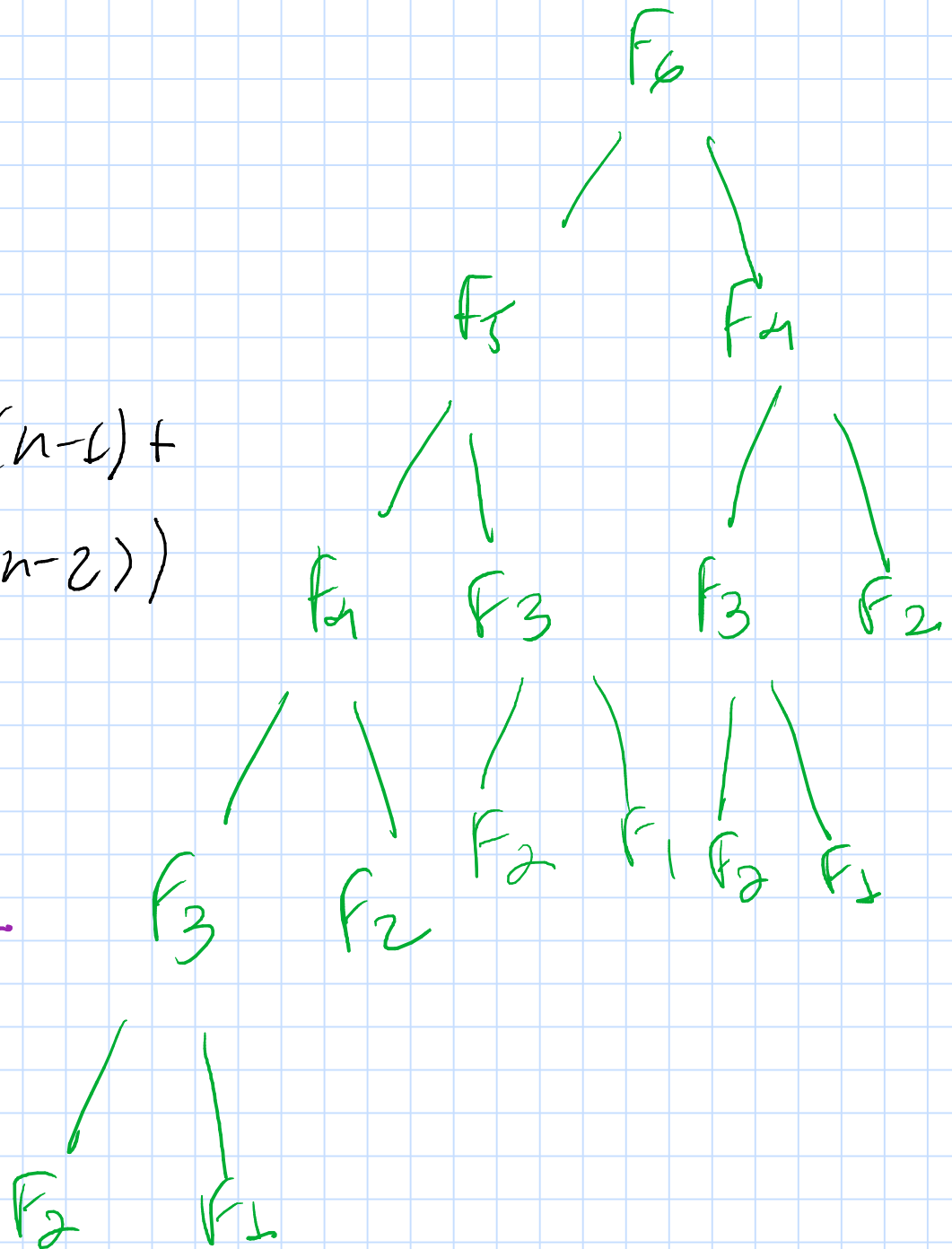
Se  $n \leq 2$

Retorna 1

Retorna (Fibonacci(n-1) +  
Fibonacci(n-2))

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

$$O(2^n)$$



Fibonacci\_iterativo(n)

Se  $n \leq 2$

Retorna 1

fib1  $\leftarrow 1$

fib2  $\leftarrow 1$

Para  $i \leftarrow 3$  até  $n$

fib3  $\leftarrow$  fib2 + fib1

fib1  $\leftarrow$  fib2

fib2  $\leftarrow$  fib3

Ⓜ n

Fibonacci\_memo(n)

Se  $n \leq 2$

Returna 1

cria vector auxiliar  $A[1..n]$

$A[1] \leftarrow 1$

$A[2] \leftarrow 1$

Pora  $i \leftarrow 3$  até  $n$

$A[i] \leftarrow 0$

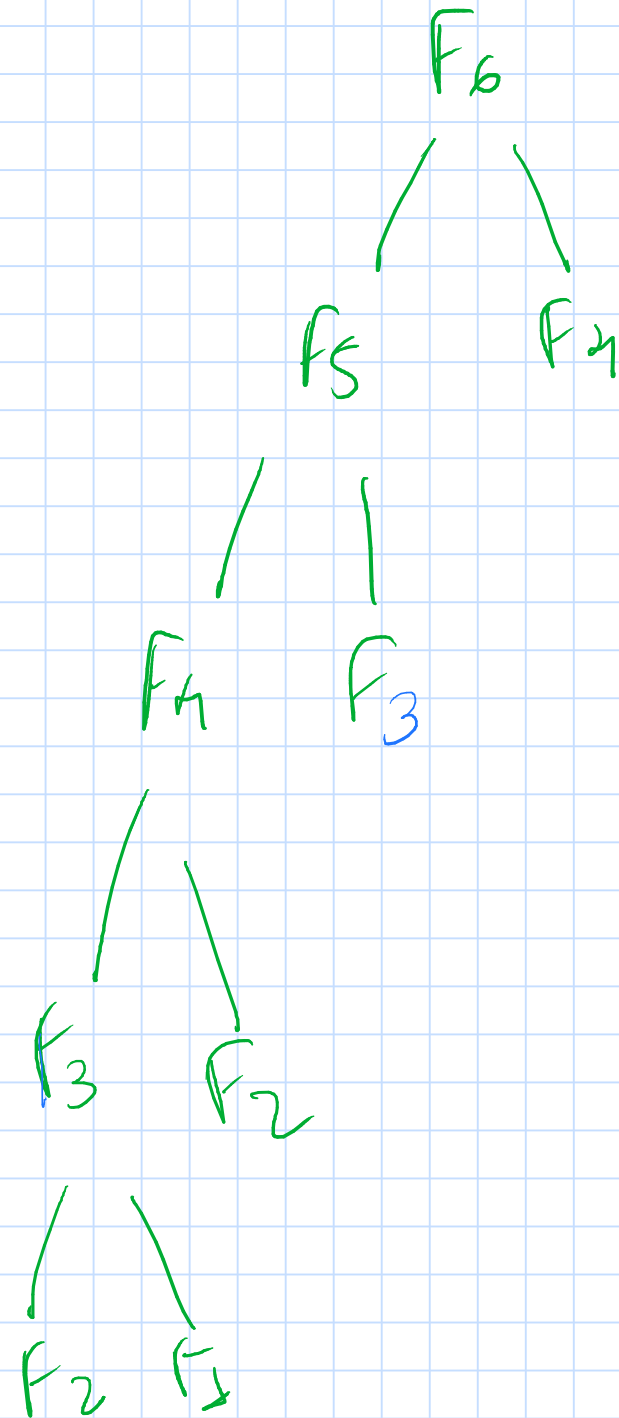
Returna fibonacci\_rec(n)

fibonacci\_rec(n)

Se  $A[n] = 0$

$A[n] \leftarrow \text{fibonacci\_rec}(n-1) +$   
 $\text{fibonacci\_rec}(n-2)$

Returna  $A[n]$



# Programação Dinâmica

↳ Tabelas!

↳ Problemas de otimização

↳ subestrutura ótima:

- A solução ótima é obtida a partir de soluções ótimas de subproblemas

Prob. otimização:

- Possui muitas soluções
- Cada solução tem um valor?

\* Queremos encontrar solução com valor máximo ou mínimo

≠ Divisão e conquista

Em P.D.:

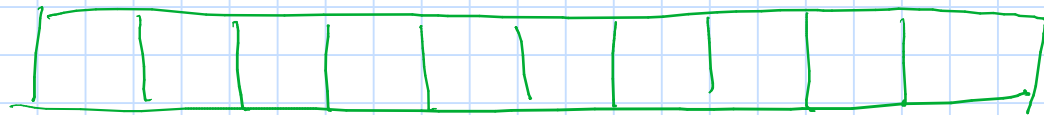
- Subproblemas se sobrepõem

- Subproblemas não idênticos ao original  
(um trabalho inteiro p/ combinar)

## Problema do corte de madeiras

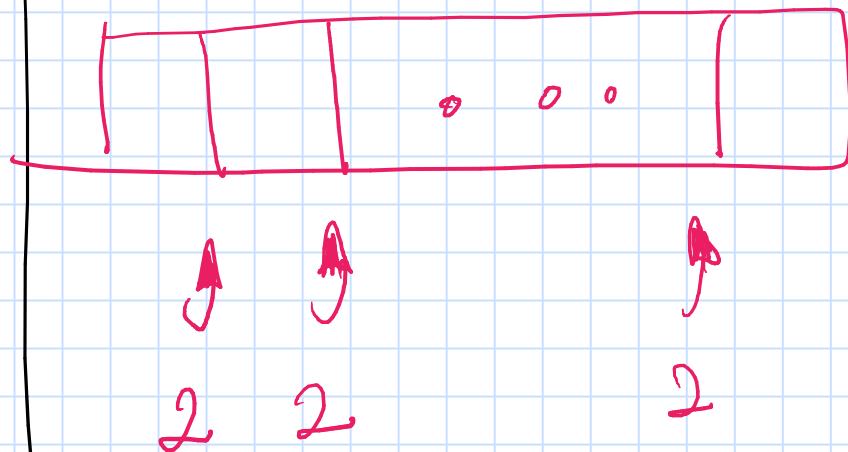
Entrada: haste de tamanho  $n$  (metros ...) e tabela de preços  $p_i$  ( $1 \leq i \leq n$ ) para um pedaço de haste de tamanho  $i$ .

Pergunta: Qual a melhor forma de cortar a haste, de forma a obter o maior lucro possível?



tam. $i$	1	2	3	4	5	6	7	8	9	10
preço $p_i$	1	5	8	9	10	17	17	20	24	30

Quantas formas de cortar a haste?



Partindo nos subproblemas...

- barra de tamanho 1

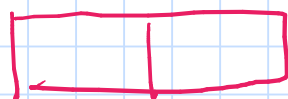
tam. $i$	1	2	3	4
menor $p_i$	1	5	8	9

$n_i$  = melhor solução de tamanho  $i$

$i = 1$

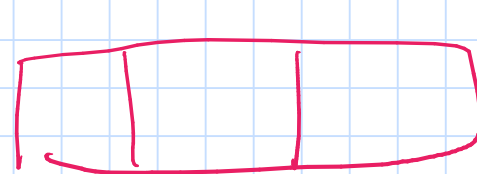
  $n_1 = 1$

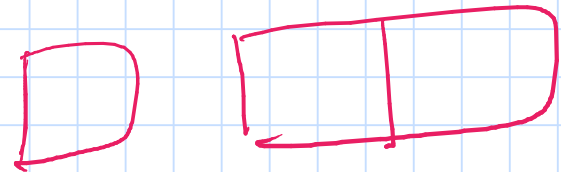
$i = 2$

 5       $n_2 = 5$

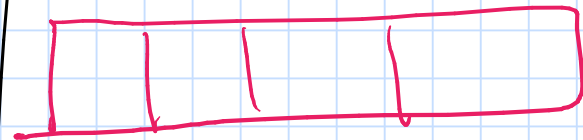
 2


$i = 3$


 8       $n_3 = 8$

  $1 + 5 = 6$

$i = 4$

 = 9

  $1 + 8 = 9$

 =  $5 + 5 = 10$

Reconstrução que relaciona  
os problemas:

→ Melhor solução de tamanho  $n$   
 $R_n = \max(p_n, R_1 + R_{n-1},$

$R_2 + R_{n-2}, R_3 + R_{n-3} + \dots$

$R_{n-1} + R_1)$

Outra forma de fazer a  
reconstrução:

$$R_n = \max_{1 \leq i \leq n} (p_i + R_{n-i})$$

CH-rec( $p, n$ )

Se  $n = 0$

Retorna 0

$q \leftarrow -\infty$

Para  $i \leftarrow 1$  até  $n$

$q \leftarrow \max(q,$   
 $p[i] + \text{CH-rec}(p, n-i))$

Retorna  $q$

Programação dinâmica:  
Abordagem "bottom-up"

Link-de-Mostes-iter( $p, n$ )

criar vetor  $r[0..n]$  e  $\Delta[0..n]$

$r[0] \leftarrow 0$

Para  $j \leftarrow 1$  até  $n$

$q \leftarrow -\infty$

Para  $i \leftarrow 1$  até  $j$

Se  $q < p[i] + r[j-i]$

$q \leftarrow p[i] + r[j-i]$

$\Delta[j] \leftarrow i$

$r[j] \leftarrow q$

Retornar  $r[n]$  e  $\Delta[n]$

$O(n^2)$



# Passos para a construção de algoritmos de programação dinâmica:

1. Definição dos subproblemas
2. Obtenção da recorrência que relaciona os problemas
3. Reconhecimento e resolução dos casos base.

Corte de Haste Memorizado ( $p, n$ )

criar  $n[0..n]$

Para  $i \in 0$  até  $n$

$n[i] \leftarrow -\infty$

Retorna CHM( $p, n, n$ )

CHM( $p, n, n$ )

Se  $n[n] \geq 0$

Retorna  $n[n]$

Se  $n = 0$

$q \leftarrow 0$

Senão  $q \leftarrow -\infty$

Para  $i \in 1$  até  $n$

$q \leftarrow \max(q, p[i] + \text{CHM}(p, n-i, n))$

...

$n[n] \leftarrow q$

Retorna  $q$