

Programação dinâmica

CLRS cap 15

- = “recursão-com-tabela”
- = transformação inteligente de recursão em iteração

Números de Fibonacci

$$F_0 = 0 \quad F_1 = 1 \quad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Números de Fibonacci

$$F_0 = 0 \quad F_1 = 1 \quad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para F_n :

FIBO-REC (n)

1 se $n \leq 1$

2 então devolva n

3 senão $a \leftarrow$ FIBO-REC ($n - 1$)

4 $b \leftarrow$ FIBO-REC ($n - 2$)

5 devolva $a + b$

Consumo de tempo

FIBO-REC (n)

1 se $n \leq 1$

2 então devolva n

3 senão $a \leftarrow$ FIBO-REC ($n - 1$)

4 $b \leftarrow$ FIBO-REC ($n - 2$)

5 devolva $a + b$

Tempo em segundos:

n	16	32	40	41	42	43	44	45	47
tempo	0.002	0.06	2.91	4.71	7.62	12.37	19.94	32.37	84.50

$F_{47} = 2971215073$

Consumo de tempo

FIBO-REC (n)

```
1  se  $n \leq 1$ 
2    então devolva  $n$ 
3    senão  $a \leftarrow$  FIBO-REC ( $n - 1$ )
4           $b \leftarrow$  FIBO-REC ( $n - 2$ )
5          devolva  $a+b$ 
```

$T(n)$:= número de somas feitas por FIBO-REC (n)

linha	número de somas
1-2	= 0
3	= $T(n - 1)$
4	= $T(n - 2)$
5	= 1
<hr/>	
$T(n)$	= $T(n - 1) + T(n - 2) + 1$

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \quad \text{para } n = 2, 3, \dots$$

A que classe Ω pertence $T(n)$?

A que classe O pertence $T(n)$?

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \text{ para } n = 2, 3, \dots$$

A que classe Ω pertence $T(n)$?

A que classe O pertence $T(n)$?

Solução: $T(n) > (3/2)^n$ para $n \geq 6$.

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
$(3/2)^n$	1	1.5	2.25	3.38	5.06	7.59	11.39	17.09	25.63	38.44

Recorrência

Prova: $T(6) = 12 > 11.40 > (3/2)^6$ e $T(7) = 20 > 18 > (3/2)^7$.

Se $n \geq 8$, então

$$\begin{aligned}T(n) &= T(n-1) + T(n-2) + 1 \\&\stackrel{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\&= (3/2 + 1)(3/2)^{n-2} + 1 \\&> (5/2)(3/2)^{n-2} \\&> (9/4)(3/2)^{n-2} \\&= (3/2)^2(3/2)^{n-2} \\&= (3/2)^n.\end{aligned}$$

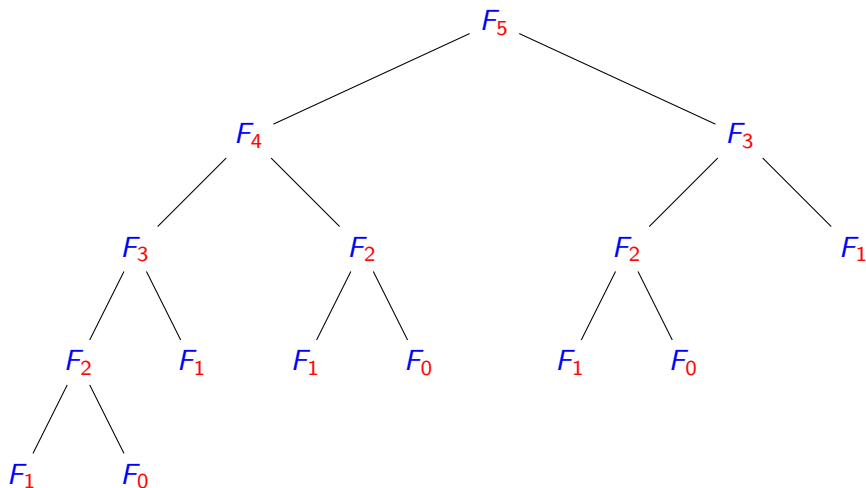
Logo, $T(n)$ é $\Omega((3/2)^n)$.

Verifique que $T(n)$ é $O(2^n)$.

Consumo de tempo

Consumo de tempo é **exponencial**.

Algoritmo resolve subproblemas muitas vezes.



Resolve subproblemas muitas vezes

FIBO-REC(5)

FIBO-REC(4)

FIBO-REC(3)

FIBO-REC(2)

FIBO-REC(1)

FIBO-REC(0)

FIBO-REC(1)

FIBO-REC(2)

FIBO-REC(1)

FIBO-REC(0)

FIBO-REC(3)

FIBO-REC(2)

FIBO-REC(1)

FIBO-REC(0)

FIBO-REC(1)

FIBO-REC(5) = 5

Resolve subproblemas muitas vezes

FIBO-REC(8)	FIBO-REC(1)	FIBO-REC(2)
FIBO-REC(7)	FIBO-REC(2)	FIBO-REC(1)
FIBO-REC(6)	FIBO-REC(1)	FIBO-REC(0)
FIBO-REC(5)	FIBO-REC(0)	FIBO-REC(1)
FIBO-REC(4)	FIBO-REC(5)	FIBO-REC(2)
FIBO-REC(3)	FIBO-REC(4)	FIBO-REC(1)
FIBO-REC(2)	FIBO-REC(3)	FIBO-REC(0)
FIBO-REC(1)	FIBO-REC(2)	FIBO-REC(3)
FIBO-REC(0)	FIBO-REC(1)	FIBO-REC(2)
FIBO-REC(1)	FIBO-REC(0)	FIBO-REC(1)
FIBO-REC(2)	FIBO-REC(1)	FIBO-REC(0)
FIBO-REC(1)	FIBO-REC(2)	FIBO-REC(1)
FIBO-REC(0)	FIBO-REC(1)	FIBO-REC(4)
FIBO-REC(3)	FIBO-REC(0)	FIBO-REC(3)
FIBO-REC(2)	FIBO-REC(3)	FIBO-REC(2)
FIBO-REC(1)	FIBO-REC(2)	FIBO-REC(1)
FIBO-REC(0)	FIBO-REC(1)	FIBO-REC(0)
FIBO-REC(1)	FIBO-REC(0)	FIBO-REC(1)
FIBO-REC(4)	FIBO-REC(1)	FIBO-REC(2)
FIBO-REC(3)	FIBO-REC(6)	FIBO-REC(1)
FIBO-REC(2)	FIBO-REC(5)	FIBO-REC(0)
FIBO-REC(1)	FIBO-REC(4)	
FIBO-REC(0)	FIBO-REC(3)	

Programação dinâmica

"Dynamic programming is a fancy name for divide-and-conquer with a table. Instead of solving subproblems recursively, solve them sequentially and store their solutions in a table. The trick is to solve them in the right order so that whenever the solution to a subproblem is needed, it is already available in the table. Dynamic programming is particularly useful on problems for which divide-and-conquer appears to yield an exponential number of subproblems, but there are really only a small number of subproblems repeated exponentially often. In this case, it makes sense to compute each solution the first time and store it away in a table for later use, instead of recomputing it recursively every time it is needed."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Versão recursiva com memoização

MEMOIZED-FIBO (f, n)

- 1 para $i \leftarrow 0$ até n faça
- 2 $f[i] \leftarrow -1$
- 3 devolva LOOKUP-FIBO (f, n)

LOOKUP-FIBO (f, n)

- 1 se $f[n] \geq 0$
- 2 então devolva $f[n]$
- 3 se $n \leq 1$
- 4 então $f[n] \leftarrow n$
- 5 senão $f[n] \leftarrow$ LOOKUP-FIBO($f, n - 1$)
 + LOOKUP-FIBO($f, n - 2$)
- 6 devolva $f[n]$

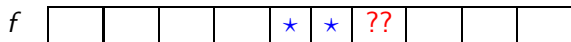
Não recalcula valores de f .

Algoritmo de programação dinâmica

Sem recursão:

```
FIBO ( $n$ )  
1  $f[0] \leftarrow 0$   
2  $f[1] \leftarrow 1$   
3 para  $i \leftarrow 2$  até  $n$  faça  
4    $f[i] \leftarrow f[i - 1] + f[i - 2]$   
5 devolva  $f[n]$ 
```

Note a tabela $f[0..n-1]$.



Consumo de tempo (e de espaço) é $\Theta(n)$.

Algoritmo de programação dinâmica

Versão com economia de espaço.

FIBO (n)

```
0 se  $n = 0$  então devolva 0
1  $f\_ant \leftarrow 0$ 
2  $f\_atual \leftarrow 1$ 
3 para  $i \leftarrow 2$  até  $n$  faça
4      $f\_prox \leftarrow f\_atual + f\_ant$ 
5      $f\_ant \leftarrow f\_atual$ 
6      $f\_atual \leftarrow f\_prox$ 
7 devolva  $f\_atual$ 
```

Algoritmo de programação dinâmica

Versão com economia de espaço.

```
FIBO ( $n$ )
0  se  $n = 0$  então devolva 0
1   $f\_ant \leftarrow 0$ 
2   $f\_atual \leftarrow 1$ 
3  para  $i \leftarrow 2$  até  $n$  faça
4     $f\_prox \leftarrow f\_atual + f\_ant$ 
5     $f\_ant \leftarrow f\_atual$ 
6     $f\_atual \leftarrow f\_prox$ 
7  devolva  $f\_atual$ 
```

Consumo de tempo é $\Theta(n)$.

Consumo de espaço é $\Theta(1)$.

Corte de hastes

Hastes de aço são vendidas em pedaços de tamanho inteiro.
As usinas produzem hastes longas,
e os comerciantes cortam em pedaços para vender.

Suponha que o preço de uma haste de tamanho i
esteja tabelado como p_i .

Corte de hastes

Hastes de aço são vendidas em pedaços de tamanho inteiro.
As usinas produzem hastes longas,
e os comerciantes cortam em pedaços para vender.

Suponha que o preço de uma haste de tamanho i
esteja tabelado como p_i .

Problema: Dada uma haste de tamanho n e a tabela p de preços,
qual a melhor forma de cortar a haste para maximizar o preço de
venda total?

Corte de hastes

Hastes de aço são vendidas em pedaços de tamanho inteiro.
As usinas produzem hastes longas,
e os comerciantes cortam em pedaços para vender.

Suponha que o preço de uma haste de tamanho i
esteja tabelado como p_i .

Problema: Dada uma haste de tamanho n e a tabela p de preços,
qual a melhor forma de cortar a haste para maximizar o preço de
venda total?

Versão simplificada: qual o maior valor q_n
que se pode obter de uma haste de tamanho n ?

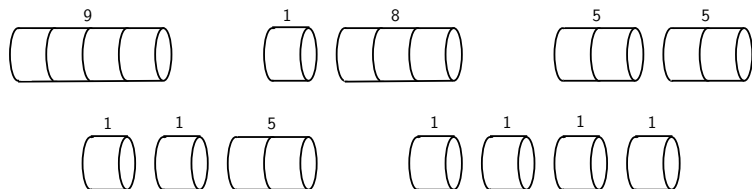
Exemplo

n	1	2	3	4	5	6	7	8	9
p_n	1	5	8	9	10	17	17	20	24

Exemplo

n	1	2	3	4	5	6	7	8	9
p_n	1	5	8	9	10	17	17	20	24

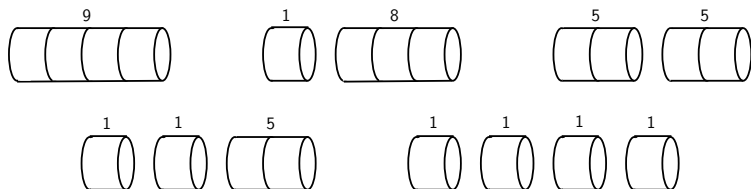
Possíveis cortes para $n = 4$:



Exemplo

n	1	2	3	4	5	6	7	8	9
p_n	1	5	8	9	10	17	17	20	24

Possíveis cortes para $n = 4$:



Melhor corte (de maior lucro): o terceiro, com valor 10.

Solução recursiva

Corta-se um primeiro pedaço de tamanho i e o pedaço restante, de tamanho $n - i$, do melhor jeito possível. O valor desse corte é

$$p_i + q_{n-i}.$$

Solução recursiva

Corta-se um primeiro pedaço de tamanho i e o pedaço restante, de tamanho $n - i$, do melhor jeito possível. O valor desse corte é

$$p_i + q_{n-i}.$$

A questão é escolher o melhor i ; o que maximiza a expressão acima:

$$q_n = \max_{1 \leq i \leq n} \{p_i + q_{n-i}\}.$$

$$q_0 = 0.$$

Primeiro código

CORTA-HASTE (p, n)

```
1 se  $n = 0$ 
2   então devolva 0
3  $q \leftarrow -\infty$ 
4 para  $i \leftarrow 1$  até  $n$ 
5    $q \leftarrow \max\{q, p[i] + \text{CORTA-HASTE}(p, n - i)\}$ 
6 devolva  $q$ 
```

Primeiro código

CORTA-HASTE (p, n)

- 1 se $n = 0$
- 2 então devolva 0
- 3 $q \leftarrow -\infty$
- 4 para $i \leftarrow 1$ até n
- 5 $q \leftarrow \max\{q, p[i] + \text{CORTA-HASTE}(p, n - i)\}$
- 6 devolva q

Consumo de tempo:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

Primeiro código

CORTA-HASTE (p, n)

```
1 se  $n = 0$ 
2   então devolva 0
3  $q \leftarrow -\infty$ 
4 para  $i \leftarrow 1$  até  $n$ 
5    $q \leftarrow \max\{q, p[i] + \text{CORTA-HASTE}(p, n - i)\}$ 
6 devolva  $q$ 
```

Consumo de tempo:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i) = 2^n.$$

Com memoização

Note que r funciona como variável global.

CORTA-HASTE-MEMOIZADO (p, n)

1 $r[0] \leftarrow 0$

2 **para** $i \leftarrow 1$ até n

3 $r[i] \leftarrow -\infty$

4 **devolva** CORTA-HASTE-MEMOIZADO-REC (p, n, r)

Com memoização

Note que r funciona como variável global.

CORTA-HASTE-MEMOIZADO (p, n)

- 1 $r[0] \leftarrow 0$
- 2 **para** $i \leftarrow 1$ até n
- 3 $r[i] \leftarrow -\infty$
- 4 **devolva** CORTA-HASTE-MEMOIZADO-REC (p, n, r)

CORTA-HASTE-MEMOIZADO-REC (p, n, r)

- 1 **se** $r[n] \geq 0$
- 2 **devolva** $r[n]$
- 3 **senão** $q \leftarrow -\infty$
- 4 **para** $i \leftarrow 1$ até n
- 5 $q \leftarrow \max\{q, p[i] + \text{CORTA-HASTE-MEMOIZADO-REC}(p, n - i, r)\}$
- 6 $r[n] \leftarrow q$
- 7 **devolva** q

Bottom up

CORTA-HASTE-BOTTOM-UP (p, n)

```
1   $r[0] \leftarrow 0$   
2  para  $j \leftarrow 1$  até  $n$   
3       $q \leftarrow -\infty$   
4      para  $i \leftarrow 1$  até  $j$   
5           $q \leftarrow \max\{q, p[i] + r[j - i]\}$   
6       $r[j] \leftarrow q$   
7  devolva  $q$ 
```

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0				

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1			

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	2		

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5		

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	6	

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	6	

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	9

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	10

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	10

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	10

Recuperando o melhor corte

CORTA-HASTE-BOTTOM-UP-COMPLETO (p, n)

```
1   $r[0] \leftarrow 0$ 
2  para  $j \leftarrow 1$  até  $n$ 
3       $q \leftarrow -\infty$ 
4      para  $i \leftarrow 1$  até  $j$ 
5          se  $q < p[i] + r[j - i]$ 
6               $q \leftarrow p[i] + r[j - i]$ 
7               $d[j] \leftarrow i$ 
8   $r[j] \leftarrow q$ 
9  devolva  $q$  e  $d$ 
```

Exemplo

n	1	2	3	4
p_n	1	5	8	9

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1			
d_n			1			

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	2		
d_n			1	1		

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5		
d_n			1	2		

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	6	
d_n			1	2	1	

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	
d_n			1	2	3	

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	9
d_n			1	2	3	1

Exemplo

n		1	2	3	4
p_n		1	5	8	9

n		0	1	2	3	4
r_n		0	1	5	8	10
d_n			1	2	3	2

Listando os cortes

(usando concatenação de listas, estilo python)

LISTA-CORTES(d, n)

1 se $n = 0$ ou $d[n] = n$

2 devolva [] ▷ lista vazia

3 senão

4 devolva [$d[n]$].LISTA-CORTES($d, n - d[n]$)

Listando os cortes

(usando concatenação de listas, estilo python)

```
LISTA-CORTES( $d, n$ )  
1  se  $n = 0$  ou  $d[n] = n$   
2    devolva [ ]    ▷ lista vazia  
3  senão  
4    devolva [  $d[n]$  ].LISTA-CORTES( $d, n - d[n]$ )
```

Consumo de tempo: $O(n)$