Subsequência comum máxima

CLRS 15.4

```
= transformação inteligente de recursão em
                                                         = "recursão-com-tabela"
iteração
```

Exercício

Problema: Decidir se z[0:m] é subsequência de

Subsequências

 $\begin{array}{l} z\,[\,0\,\colon\! k\,] \ \mbox{\'e} \ subsequência \ de \times [\,0\,\colon\! m\,] \\ se \ existem \ indices \ i_0 < \cdots < i_{k-1} \ tais \ que \end{array}$

$$\mathbf{z}[0] = \mathbf{x}[\mathbf{i}_0] \quad \dots \quad \mathbf{z}[\mathbf{k}-1] = \mathbf{x}[\mathbf{i}_{\mathbf{k}-1}]$$

EXEMPLOS:

A A D A A é subseq de A B R A C A D A B R A

```
A B R A C A D A
B

B

カ

ト
```

Exercício

Problema: Decidir se z[0:m] é subsequência de x[0:m]

```
def sub_seq(z, x):

i = len(z)-1

j = len(x)-1
                         while i >= 0 and j >= 0:
if z[i] == x[j]:
i -= 1
return i<0
           j-= 1
```

Exercício

Problema: Decidir se z[0:m] é subsequência de

```
def sub_seq(z,x):

i = len(z)-1

j = len(x)-1
            while i >= 0 and j >= 0:

if z[i] == x[j]:

i -= 1

j-= 1
return i < 0
```

Consumo de tempo é O(m + n)

Subsequência comum máxima

```
ssco = subsequência comum
                                                                                                                                                                                  z é subseq comum de x e y
                                                                            Exemplos: x = ABCBDAB
                                                                                                                                                        se z é subsequência de x e de y
Outra ssco = B D A B
                                                  y = BDCABA
```

```
Problema: Encontrar uma ssco máxima de x e y.

Exemplos: x = A B C B D A B y = B D C A B A ssco = B C A ssco maximal = A B A ssco máxima = B C A B
```

LCS = Longest Common Subsequence

Outra ssco máxima = B D A B

diff -u

Função recursiva

Retorna o comprimento de uma ssco máxima de x[0:1] e y[0:j].

```
def lcs_rec(x, i, y, j):
    if i == 0 or j == 0:
        return 0
    if x[i-1] == y[j-1]:
        return lcs_rec(x, i-1, y, j-1) + 1
    a = lcs_rec(x, i-1, y, j)
    b = lcs_rec(x, i, y, j-1)
    return max(a, b)
```

Subestrutura ótima

Suponha que z[0:k] é ssco máxima de x[1:m] e y[0:n].

```
▶ Se x [m-1] = y [n-1], então z[k-1] = x[m-1] = y[n-1] e z[0:k-1] é ssco máxima de x[0:m-1] e y[0:n-1].
```

• Se
$$x[m-1] \neq y[n-1]$$
, então $z[k-1] \neq x[m-1]$ implica que $z[0:k]$ é ssco máxima de $x[0:m-1]$ e $y[0:n]$.

• Se
$$x[m-1] \neq y[n-1]$$
, então $z[k-1] \neq y[n-1]$ implica que $z[0:k]$ é ssco máxima de $x[0:m]$ e $y[0:n-1]$.

Consumo de tempo

```
número máximo de comparações feitas por 1 cs rec(x, m, y, n)
```

Recorrência

```
T(0, \mathbf{n}) = 0

T(\mathbf{m}, 0) = 0

T(\mathbf{m}, \mathbf{n}) = T(\mathbf{m} - 1, \mathbf{n}) + T(\mathbf{m}, \mathbf{n} - 1) + 1

para \mathbf{n} \ge 0 e \mathbf{m} \ge 0
```

T(m, n) é exponencial

מולים ולים

O consumo de tempo do algoritmo lcs_rec() é exponecial.

Programação dinâmica

ssco máxima. Problema: encontrar o comprimento de uma

c[i][j] = comprimento de uma ssco máxima de <math>x[0:i] e y[0:j]

Recorrência:

$$c[0][j] = c[i][0] = 0$$

$$c[i][j] = c[i-1][j-1] + 1 \text{ se } x[i-1] = y[j-1]$$

$$c[i][j] = \max(c[i][j-1], c[i-1][j]) \text{ se }$$

$$x[i-1] \neq y[j-1]$$

Programação dinâmica

7	6	S	4	ω	12	_	0	
0	0	0	0	0	0	0	0	0
							0	_(
							0	2
							0	ယ
				*	*		0	4
				??	*		0	5
							0	6
							0	7
								u.

$$T(m, m) = {2m \choose m} - 1$$

$$> \frac{4^m}{2m+1} - 1.$$

Portanto, $T(m, m) > 4^m/m$.

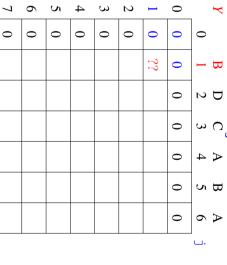
Programação dinâmica Cada subproblema, comprimento de uma ssco máxima de x[0:i] e y[1:j], é resolvido uma só ۷ez.

Em que ordem calcular os componentes da tabela c?

Para calcular ${\tt c[3,5]}$ preciso de ${\tt c[3,4]}, {\tt c[2,5]}$ e de ${\tt c[2,4]}.$

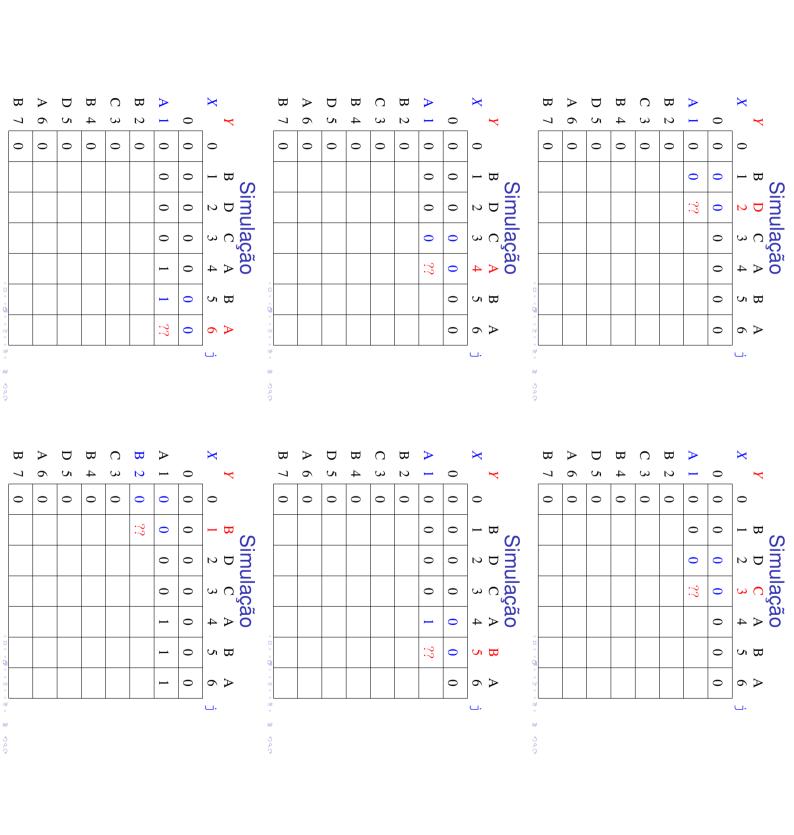
Calcule todos os
$$c[\mathtt{i},\mathtt{j}]$$
 com $\mathtt{i}=\mathtt{1},$ $\mathtt{j}=\mathtt{0},\mathtt{1},\ldots,\mathtt{n},$ depois todos com $\mathtt{i}=\mathtt{2},\mathtt{j}=\mathtt{0},\mathtt{1},\ldots,\mathtt{n},$ depois todos com $\mathtt{i}=\mathtt{3},\mathtt{j}=\mathtt{0},\mathtt{1},\ldots,\mathtt{n},$ etc.

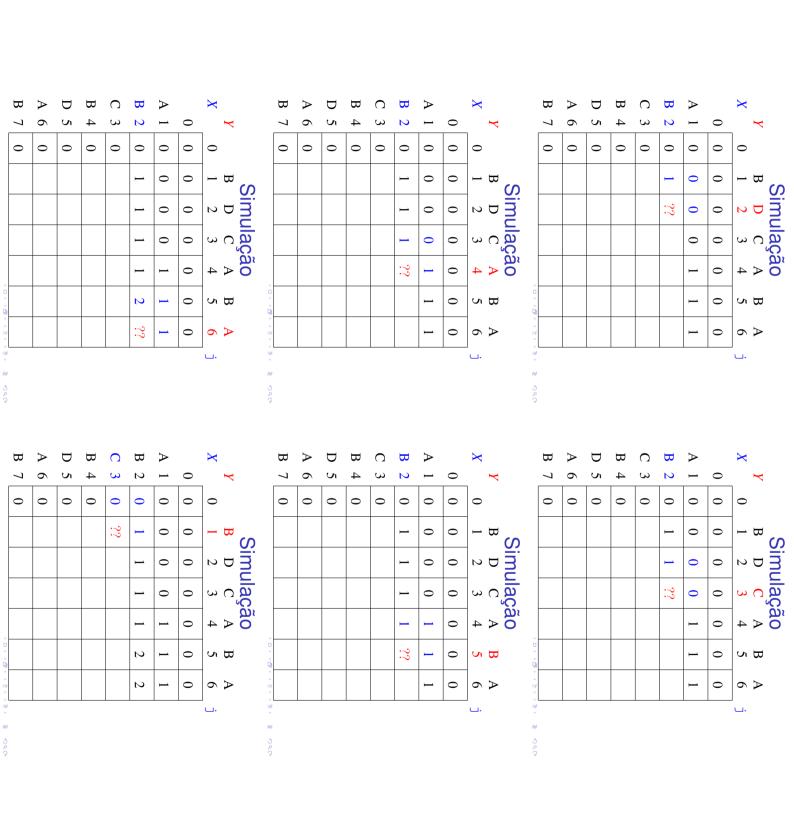
Simulação

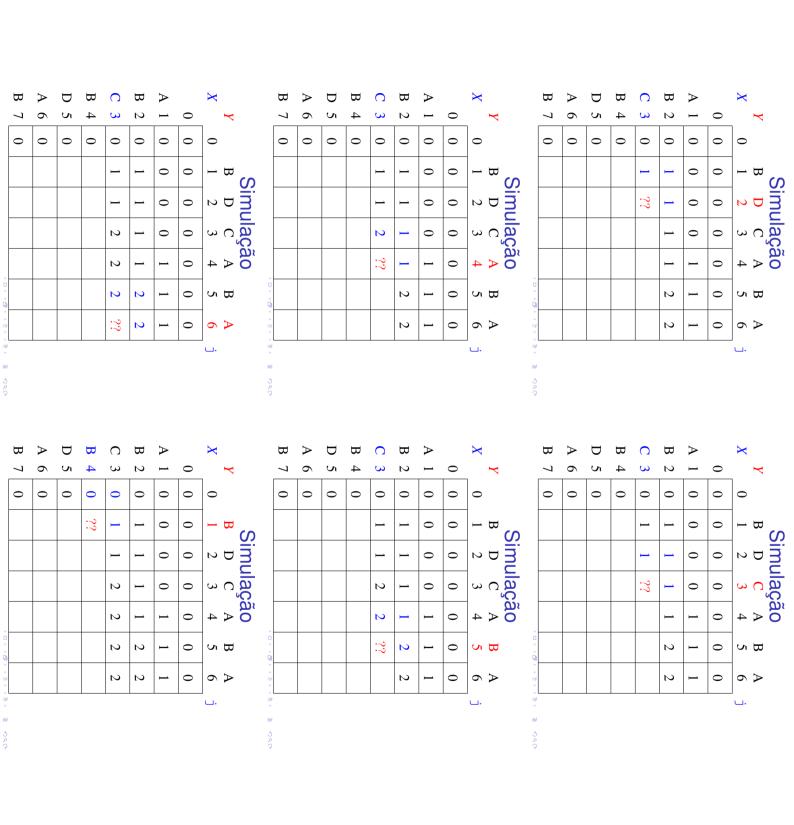


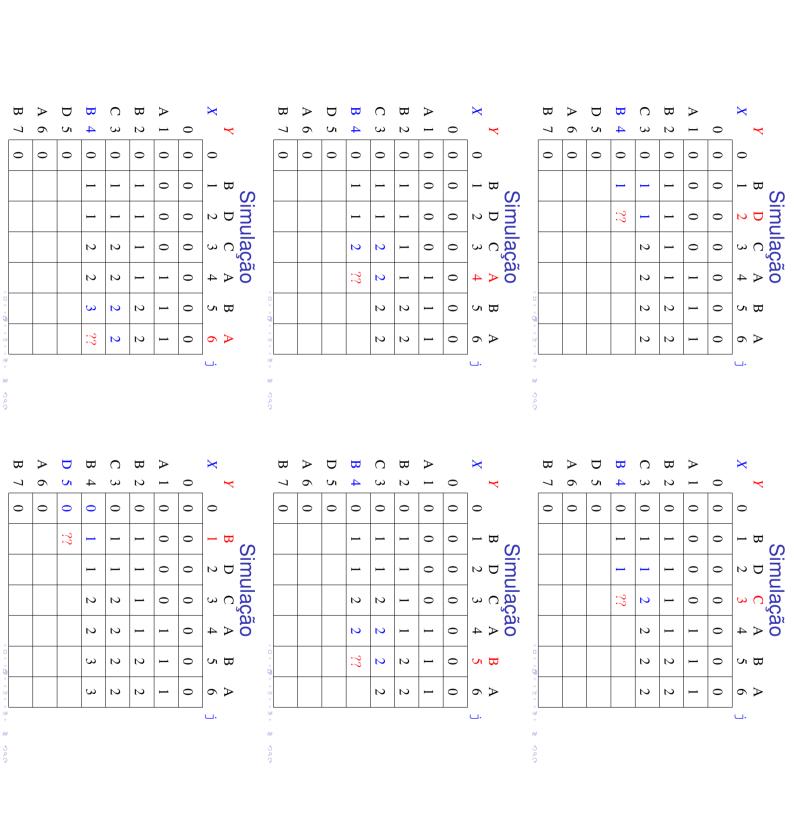
 \bigcirc \mathbf{B}

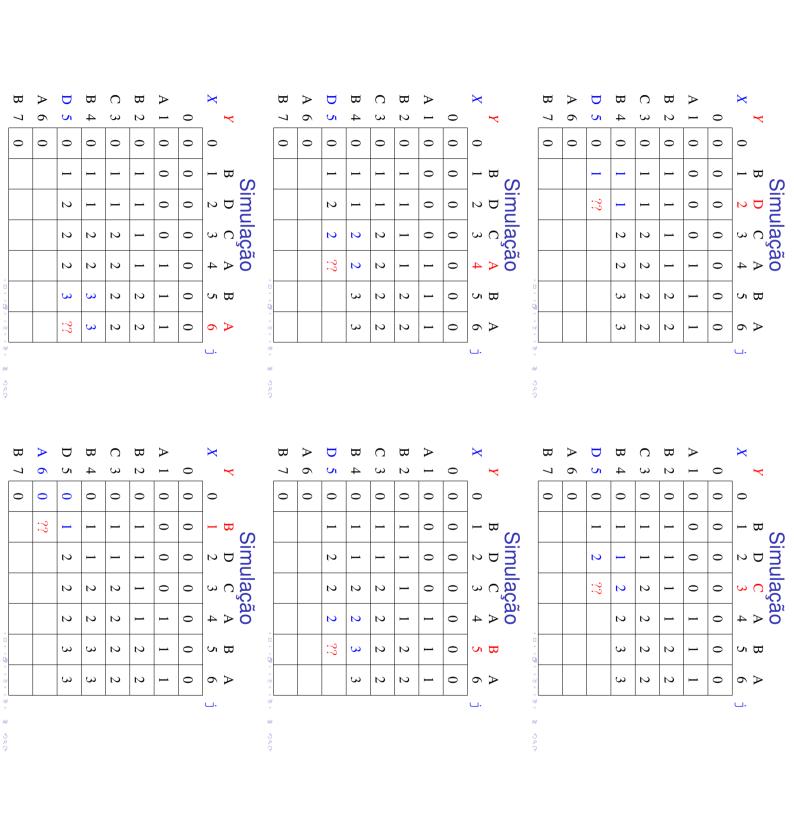
D ₿ \triangleright

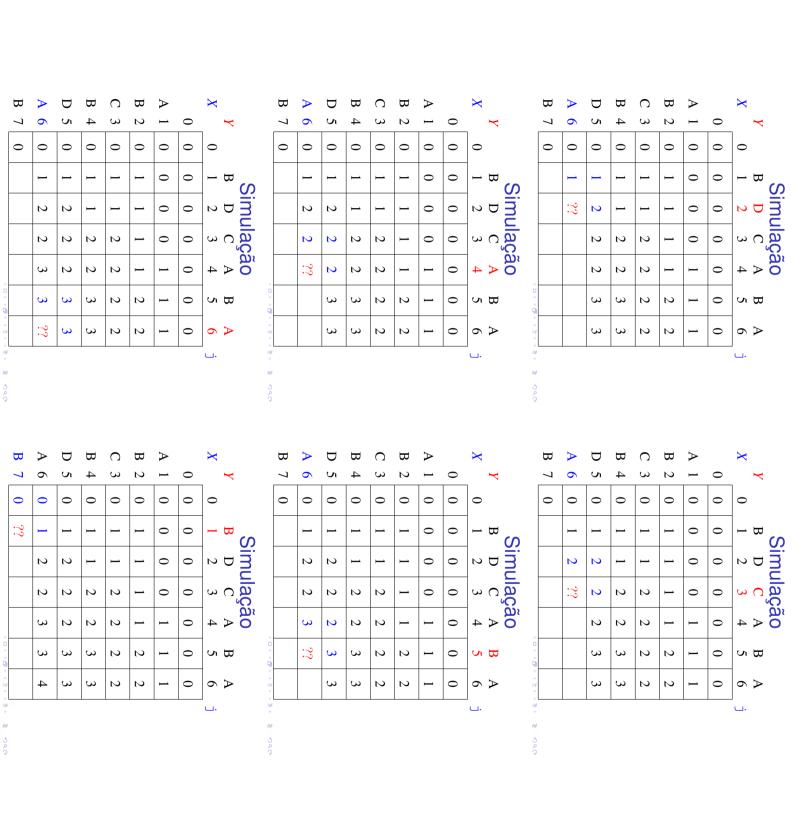


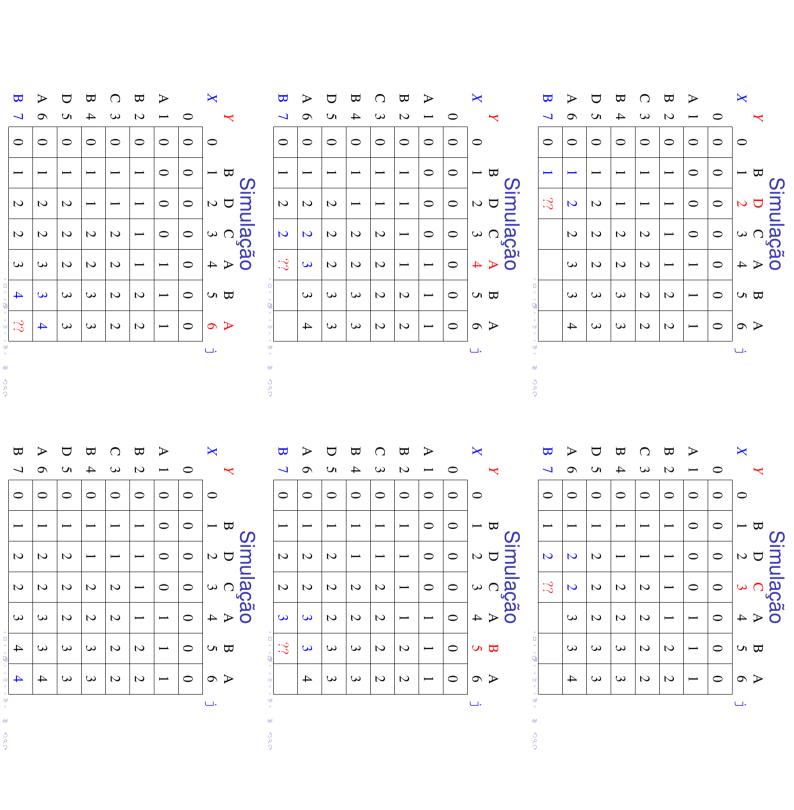












Função de prog-din

Retorna o comprimento de uma ssco máxima

```
de x[0:m] e y[0:n].
def lcs_prog_din(x, m, y, n):
    c = crie_matriz(m+1,n+1)
c[i][j] = \max(c[i-1][j], c[i][j-1])
return c[m][n]
                                                                                                                                                                 for i in range(m+1): c[i][0] = 0
for j in range(n+1): c[0][j] = 0
                                                                                                                                          for i in range(1,m+1):
                                                                      for j in range(1,n+1):

if x[i-1] == y[j-1]:

c[i][j] = c[i-1][j-1] + 1
```

```
Subsequência comum máxima
```

 $def get_lcs(x, b)$:

get_lcs()

while i > 0 and j > 0: if $b[i][j] == "\neg "$: z.insert(0, x[i])

j -= 1 elif b[i][j] == "←": j -= 1

<u>i</u> = 1

j = len(b[0])i = len(b)

return z

else:

Conclusão

O consumo de tempo da função

Função de prog-din

```
c[i][j] = c[i][j-1]
b[i][j] = "\leftarrow"
return c[m][n]
                 def lcs_prog_din(x, m, y, n):
```