Mais programação dinâmica

CLRS 15.4

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

Subsequências

$\langle z_1, \ldots, z_k \rangle$ é subsequência de $\langle x_1, \ldots, x_m \rangle$
se existem índices $i_1 < \cdots < i_k$ tais que

$$z_1 = x_{i_1} \quad \ldots \quad z_k = x_{i_k}$$

EXEMPLOS:

$\langle 5, 9, 2, 7 \rangle$ é subsequência de $\langle 9, 5, 6, 9, 6, 2, 7, 3 \rangle$

$\langle$A, A, D, A, A$\rangle$ é subsequência de $\langle$A, B, R, A, C, A, D, A, B, R, A$\rangle$

A			A			D	A			A
|			|			|	|			|
A	B	R	A	C	A	D	A	B	R	A

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subsequência de $X[1\,..\,n]$

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subsequência de $X[1\,..\,n]$

```
SUB-SEQ (Z, m, X, n)
i ← m
j ← n
enquanto i ≥ 1 e j ≥ 1 faça
    se Z[i] = X[j]
        então i ← i − 1
    j ← j − 1
se i ≥ 1
    então devolva "não é subsequência"
    senão devolva "é subsequência"
```

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subsequência de $X[1\,..\,n]$

SUB-SEQ (Z, m, X, n)

$i \leftarrow m$

$j \leftarrow n$

enquanto $i \geq 1$ **e** $j \geq 1$ **faça**

    se $Z[i] = X[j]$

        então $i \leftarrow i - 1$

    $j \leftarrow j - 1$

se $i \geq 1$

    então devolva "não é subsequência"

    senão devolva "é subsequência"

Consumo de tempo é $\mathrm{O}(n)$ e $\Omega(\min\{m, n\})$.

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subsequência de $X[1\,..\,n]$

```
SUB-SEQ (Z, m, X, n)
i ← m
j ← n
enquanto i ≥ 1 e j ≥ 1 faça
    se Z[i] = X[j]
        então i ← i − 1
    j ← j − 1
se i ≥ 1
    então devolva "não é subsequência"
    senão devolva "é subsequência"
```

Invariantes:

(i0) $Z[i{+}1\,..\,m]$ é subsequência de $X[j{+}1\,..\,n]$

(i1) $Z[i\,..\,m]$ **não** é subsequência de $X[j{+}1\,..\,n]$

Subsequência comum máxima

Z é subseq comum de X e Y

se Z *é subsequência de* X *e de* Y

ssco = subseq comum

Subsequência comum máxima

Z é subseq comum de X e Y

se Z é subsequência de X e de Y

ssco = subseq comum

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

Subsequência comum máxima

Z é subseq comum de X e Y

se Z é subsequência de X e de Y

ssco = subseq comum

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

Outra ssco = B D A B

Problema

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

Outra ssco máxima = B D A B

LCS = Longest Common Subsequence

diff

```
> more abracadabra
A
B
R
A
C
A
D
A
B
R
A
```

```
> more yabbadabbadoo
Y
A
B
B
A
D
A
B
B
A
D
D
O
```

diff -u abracadabra yabbadabbadoo

```
+Y
 A
 B
-R
-A
-C
+B
 A
 D
 A
 B
-R
+B
 A
+D
+O
+O
```

Subestrutura ótima

Suponha que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

- Se $X[m] = Y[n]$,
  então $Z[k] = X[m] = Y[n]$ e
  $Z[1\,..\,k{-}1]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n{-}1]$.

Subestrutura ótima

Suponha que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

- Se $X[m] = Y[n]$,
  então $Z[k] = X[m] = Y[n]$ e
  $Z[1\,..\,k{-}1]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n{-}1]$.

- Se $X[m] \neq Y[n]$,
  então $Z[k] \neq X[m]$ implica que
  $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n]$.

Subestrutura ótima

Suponha que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

- Se $X[m] = Y[n]$,
  então $Z[k] = X[m] = Y[n]$ e
  $Z[1\,..\,k{-}1]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n{-}1]$.

- Se $X[m] \neq Y[n]$,
  então $Z[k] \neq X[m]$ implica que
  $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n]$.

- Se $X[m] \neq Y[n]$,
  então $Z[k] \neq Y[n]$ implica que
  $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n{-}1]$.

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

$c[i,j] =$ comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

$c[i,j] =$ comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$

Recorrência:

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

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

$c[i,j] =$ comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$

Recorrência:

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

$c[i,j] = c[i-1,j-1] + 1$ se $X[i] = Y[j]$

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

$c[i,j] =$ comprimento de uma ssco máxima de $X[1\,.\,.\,i]$ e $Y[1\,.\,.\,j]$

Recorrência:

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

$$c[i,j] = c[i{-}1,j{-}1] + 1 \text{ se } X[i] = Y[j]$$

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

Algoritmo recursivo

Devolve o comprimento de uma ssco máxima de $X[1..i]$ e $Y[1..j]$.

REC-LCS-LENGTH (X, i, Y, j)
se $i = 0$ **ou** $j = 0$ **então devolva** 0
se $X[i] = Y[j]$
    então $c[i,j] \leftarrow$ REC-LCS-LENGTH $(X, i{-}1, Y, j{-}1) + 1$
    senão $q_1 \leftarrow$ REC-LCS-LENGTH $(X, i{-}1, Y, j)$
            $q_2 \leftarrow$ REC-LCS-LENGTH $(X, i, Y, j{-}1)$
            se $q_1 \geq q2$
                então $c[i,j] \leftarrow q_1$
                senão$c[i,j] \leftarrow q_2$
devolva $c[i,j]$

Consumo de tempo

$T(m,n) :=$ número de comparações feitas por
REC-LCS-LENGTH (X,m,Y,n) no pior caso

Consumo de tempo

$T(m,n) :=$ número de comparações feitas por
REC-LCS-LENGTH (X,m,Y,n) no pior caso

Recorrência

$$T(0,n) = 0$$
$$T(m,0) = 0$$
$$T(m,n) \geq T(m-1,n) + T(m,n-1) + 1 \quad \text{para } n \geq 1 \text{ e } m \geq 1$$

Consumo de tempo

$T(m,n) :=$ número de comparações feitas por REC-LCS-LENGTH (X,m,Y,n) no pior caso

Recorrência

$$
\begin{aligned}
T(0,n) &= 0 \\
T(m,0) &= 0 \\
T(m,n) &\geq T(m-1,n) + T(m,n-1) + 1 \quad \text{para } n \geq 1 \text{ e } m \geq 1
\end{aligned}
$$

A que classe Ω pertence $T(m,n)$?

Recorrência

Note que $T(m, n) = T(n, m)$ para $n = 0, 1, \ldots$ e $m = 0, 1, \ldots$.

Recorrência

Note que $T(m,n) = T(n,m)$ para $n = 0,1,\ldots$ e $m = 0,1,\ldots$.

Seja $k := \min\{m,n\}$. Temos que

$$T(m,n) \geq T(k,k) \geq S(k),$$

onde

$$\begin{aligned} S(0) &= 0 \\ S(k) &= 2S(k-1)+1 \quad \text{para } k = 1,2,\ldots \end{aligned}$$

Recorrência

Note que $T(m,n) = T(n,m)$ para $n = 0,1,\ldots$ e $m = 0,1,\ldots$.

Seja $k := \min\{m,n\}$. Temos que

$$T(m,n) \geq T(k,k) \geq S(k),$$

onde

$$\begin{aligned} S(0) &= 0 \\ S(k) &= 2S(k-1) + 1 \quad \text{para } k = 1,2,\ldots \end{aligned}$$

$S(k)$ é $\Theta(2^k) \Rightarrow T(m,n)$ é $\Omega(2^{\min\{m,n\}})$

Recorrência

Note que $T(m,n) = T(n,m)$ para $n = 0,1,\ldots$ e $m = 0,1,\ldots$.

Seja $k := \min\{m,n\}$. Temos que

$$T(m,n) \geq T(k,k) \geq S(k),$$

onde

$$\begin{aligned} S(0) &= 0 \\ S(k) &= 2S(k-1)+1 \quad \text{para } k = 1,2,\ldots \end{aligned}$$

$S(k)$ é $\Theta(2^k) \Rightarrow T(m,n)$ é $\Omega(2^{\min\{m,n\}})$

$T(m,n)$ é exponencial

Conclusão

O consumo de tempo do algoritmo REC-LEC-LENGTH é $\Omega(2^{\min\{m,n\}})$.

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,..\,i] \quad \text{e} \quad Y[1\,..\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1..i] \quad \text{e} \quad Y[1..j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

$c[4,5]$, $c[3,6]$ e de $c[3,5]$.

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,..\,i] \quad \text{e} \quad Y[1\,..\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

$c[4,5]$, $c[3,6]$ e de $c[3,5]$.

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

Programação dinâmica

	1	2	3	4	5	6	7	8	j
1	0	0	0	0	0	0	0	0	
2	0								
3	0				⋆	⋆			
4	0				⋆	??			
5	0								
6	0								
7	0								
8	0								
i									

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	??						
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	??					
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	??				
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	??			
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	*j*
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	??		
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	??	
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	??						
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

			B	D	C	A	B	A	
X	Y	0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	??					
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	??				
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	??			
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	??		
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	??	
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	??						
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	??					
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	??				
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	??			
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	??		
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	??	
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

			B	D	C	A	B	A	
X	Y	0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	??						
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X \ Y			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	??					
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	??				
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	*j*
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	??			
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X \ Y			B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	??		
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	??	
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	??						
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	??					
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	??				
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	??			
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	??		
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	??	
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A
X		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	??					
B	7	0						

j

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	??					
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	??				
B	7	0							
	i								

Simulação

			B	D	C	A	B	A	
X	Y	0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	??			
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	??		
B	7	0							
	i								

Simulação

X	Y	0	1 (B)	2 (D)	3 (C)	4 (A)	5 (B)	6 (A)
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	??
B	7	0						

i j

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	??						
	i								

Simulação

			B	D	C	A	B	A	
X	Y	0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	??					
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	??				
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	??			
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	??		
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	??	
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	4	
	i								

Algoritmo de programação dinâmica

Devolve o comprimento de
uma ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

```
LEC-LENGTH (X, m, Y, n)
para i ← 0 até m faça c[i,0] ← 0
para j ← 1 até n faça c[0,j] ← 0
para i ← 1 até m faça
    para j ← 1 até n faça
        se X[i] = Y[j]
            então c[i,j] ← c[i−1,j−1] + 1
            senão se c[i−1,j] ≥ c[i,j−1]
                      então c[i,j] ← c[i−1,j]
                      senão c[i,j] ← c[i,j−1]
devolva c[m,n]
```

Algoritmo de programação dinâmica

Devolve o comprimento de
uma ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

```
LEC-LENGTH (X, m, Y, n)
para i ← 0 até m faça c[i,0] ← 0
para j ← 1 até n faça c[0,j] ← 0
para i ← 1 até m faça
    para j ← 1 até n faça
        se X[i] = Y[j]
            então c[i,j] ← c[i−1,j−1] + 1
            senão se c[i−1,j] ≥ c[i,j−1]
                    então c[i,j] ← c[i−1,j]
                    senão c[i,j] ← c[i,j−1]
devolva c[m,n]
```

Consumo de tempo: $\mathrm{O}(mn)$

Conclusão

$\boxed{\text{O consumo de tempo do algoritmo LEC-LENGTH é } \Theta(mn).}$

Subsequência comum máxima

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	★	★	★	★	★	★	★	
A	1	★	←	←	←	↖	↑	↖	
B	2	★	↖	↑	↑	←	↖	↑	
C	3	★	←	←	↖	↑	←	←	
B	4	★	↖	←	←	←	↖	↑	
D	5	★	←	↖	←	←	←	←	
A	6	★	←	←	←	↖	←	↖	
B	7	★	↖	←	←	←	↖	←	
	i								

Algoritmo de programação dinâmica

```
LEC-LENGTH (X, m, Y, n)
para i ← 0 até m faça c[i,0] ← 0
para j ← 1 até n faça c[0,j] ← 0
para i ← 1 até m faça
    para j ← 1 até n faça
        se X[i] = Y[j]
            então c[i,j] ← c[i−1,j−1] + 1
                  b[i,j] ← "↖"
            senão se c[i−1,j] ≥ c[i,j−1]
                      então c[i,j] ← c[i−1,j]
                            b[i,j] ← "↑"
                      senão c[i,j] ← c[i,j−1]
                            b[i,j] ← "←"
devolva c e b
```

Algoritmo de programação dinâmica

```
LEC-LENGTH (X, m, Y, n)
para i ← 0 até m faça c[i,0] ← 0
para j ← 1 até n faça c[0,j] ← 0
para i ← 1 até m faça
    para j ← 1 até n faça
        se X[i] = Y[j]
            então c[i,j] ← c[i−1,j−1] + 1
                  b[i,j] ← "↖"
            senão se c[i−1,j] ≥ c[i,j−1]
                then c[i,j] ← c[i−1,j]
                     b[i,j] ← "↑"
                senão c[i,j] ← c[i,j−1]
                      b[i,j] ← "←"
devolva c e b
```

Consumo de tempo: $\mathrm{O}(mn)$

Get-LCS

```
GET-LCS (X, m, n, b, máxcomp)
  k ← máxcomp
  i ← m
  j ← n
  enquanto i > 0 e j > 0 faça
      se b[i,j] = "↖"
          então Z[k] ← X[i]
                k ← k − 1    i ← i − 1    j ← j − 1
          senão se b[i,j] = "←"
                    então i ← i − 1
                    senão j ← j − 1
  devolva Z
```

Consumo de tempo é $\mathrm{O}(m+n)$ e $\Omega(\min\{m,n\})$.

Exercícios

Exercício 20.A
Escreva um algoritmo para decidir se $\langle z_1, \ldots, z_k \rangle$ é subsequência de $\langle x_1, \ldots, x_m \rangle$. Prove rigorosamente que o seu algoritmo está correto.

Exercício 20.B
Suponha que os elementos de uma sequência $\langle a_1, \ldots, a_n \rangle$ são distintos dois a dois. Quantas subsequências tem a sequência?

Exercício 20.C
Uma subsequência crescente Z de uma sequência X e é *máxima* se não existe outra subsequência crescente mais longa. A subsequência $\langle 5, 6, 9 \rangle$ de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$ é máxima? Dê uma sequência crescente máxima de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$. Mostre que o algoritmo "guloso" óbvio não é capaz, em geral, de encontrar uma subsequência crescente máxima de uma sequência dada. (Algoritmo guloso óbvio: escolha o menor elemento de X; a partir daí, escolha sempre o próximo elemento de X que seja maior ou igual ao último escolhido.)

Exercício 20.D
Escreva um algoritmo de programação dinâmica para resolver o problema da subsequência crescente máxima.

Mais exercícios

Exercício 20.E [CLRS 15.4-5]
Mostre como o algoritmo da subsequência comum máxima pode ser usado para resolver o problema da subsequência crescente máxima de uma sequência numérica. Dê uma delimitação justa, em notação Θ, do consumo de tempo de sua solução.

Exercício 20.F [Printing neatly. CLRS 15-2]
Considere a sequência $P_1, P_2, \ldots, P_n$ de palavras que constitui um parágrafo de texto. A palavra P_i tem l_i caracteres. Queremos imprimir as palavras em linhas, na ordem dada, de modo que cada linha tenha no máximo M caracteres. Se uma determinada linha contém as palavras $P_i, P_{i+1}, \ldots, P_j$ (com $i \leq j$) e há exatamente um espaço entre cada par de palavras consecutivas, o número de espaços no fim da linha é

$$M - (l_i + 1 + l_{i+1} + 1 + \cdots + 1 + l_j).$$

É claro que não devemos permitir que esse número seja negativo. Queremos minimizar, com relação a todas as linhas exceto a última, a soma dos cubos dos números de espaços no fim de cada linha. (Assim, se temos linhas $1, 2, \ldots, L$ e b_p espaços no fim da linha p, queremos minimizar $b_1^3 + b_2^3 + \cdots + b_{L-1}^3$).
Dê um exemplo para mostrar que algoritmos inocentes não resolvem o problema. Dê um algoritmo de programação dinâmica que resolva o problema. Qual a "optimal substructure property" para esse problema? Faça uma análise do consumo de tempo do algoritmo.

Mais programação dinâmica

KT 6.4

Aproveite para olhar todo o Cap 6 do KT,
que é sobre programação dinâmica.

= "recursão–com–tabela"
= transformação inteligente de recursão em iteração

Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$x \cdot w \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \text{ para todo } i$$

Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$x \cdot w \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \text{ para todo } i$$

O valor de uma mochila é o número $x \cdot v$.

Uma mochila é ótima se tem valor máximo.

Problema booleano da mochila

Uma mochila $x[1 \mathinner{..} n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila booleana ótima.

Problema booleano da mochila

Uma mochila $x[1\,..\,n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila booleana ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila booleana ótima para o problema (w, v, n, W).

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila booleana ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila booleana ótima para
$(w, v, n-1, W-w[n])$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila booleana ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila booleana ótima para
$(w, v, n - 1, W - w[n])$

senão $x[1\,..\,n{-}1]$ é mochila booleana ótima para
$(w, v, n - 1, W)$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila booleana ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila booleana ótima para
$(w, v, n-1, W - w[n])$

senão $x[1\,..\,n{-}1]$ é mochila booleana ótima para
$(w, v, n-1, W)$

NOTA. Não há nada de especial acerca do índice n.
Uma afirmação semelhante vale para qualquer índice i.

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

$t[i, Y] =$ valor de uma mochila booleana ótima
para (w, v, i, Y)

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

$t[i, Y] =$ valor de uma mochila booleana ótima para (w, v, i, Y)

$=$ valor da expressão $x \cdot v$ sujeito às restrições

$$x \cdot w \leq Y,$$

onde x é uma mochila booleana ótima

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

$t[i, Y] =$ valor de uma mochila booleana ótima
para (w, v, i, Y)

$=$ valor da expressão $x \cdot v$ sujeito às restrições

$$x \cdot w \leq Y,$$

onde x é uma mochila booleana ótima

Possíveis valores de Y: $0, 1, 2, \ldots, W$

Recorrência

$t[i, Y] =$ valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i{-}1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i{-}1, Y{-}w[i]] + v[i]\}$ se $w[i] \leq Y$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

REC-MOCHILA (w, v, n, W)

se $n = 0$ **ou** $W = 0$
    então devolva 0
se $w[n] > W$
    então devolva REC-MOCHILA $(w, v, n{-}1, W)$
$a \leftarrow$ REC-MOCHILA $(w, v, n{-}1, W)$
$b \leftarrow$ REC-MOCHILA $(w, v, n{-}1, W{-}w[n]) + v[n]$
devolva $\max\{a, b\}$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

REC-MOCHILA (w, v, n, W)
1 **se** $n = 0$ **ou** $W = 0$
2 **então devolva** 0
3 **se** $w[n] > W$
4 **então devolva** REC-MOCHILA $(w, v, n-1, W)$
5 $a \leftarrow$ REC-MOCHILA $(w, v, n-1, W)$
6 $b \leftarrow$ REC-MOCHILA $(w, v, n-1, W-w[n]) + v[n]$
7 **devolva** $\max\{a, b\}$

Consumo de tempo no pior caso é $\Omega(2^n)$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

REC-MOCHILA (w, v, n, W)
se $n = 0$ **ou** $W = 0$
  então devolva 0
se $w[n] > W$
  então devolva REC-MOCHILA $(w, v, n{-}1, W)$
$a \leftarrow$ REC-MOCHILA $(w, v, n{-}1, W)$
$b \leftarrow$ REC-MOCHILA $(w, v, n{-}1, W{-}w[n]) + v[n]$
devolva $\max\{a, b\}$

Consumo de tempo no pior caso é $\Omega(2^n)$

Por que demora tanto?
O mesmo subproblema é resolvido muitas vezes.

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Olhe a recorrência e pense...

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$ se $w[i] \leq Y$

Programação dinâmica

	0	1	2	3	4	5	6	7	Y
0	0	0	0	0	0	0	0	0	
1	0								
2	0	⋆	⋆	⋆	⋆	⋆			
3	0					??			
4	0								
5	0								
6	0								
7	0								

i

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0					
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0				
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0		
2	0					
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	
2	0					
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0					
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0				
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400			
3	0					
4	0					

Y

i

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400		
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0	0	500	500	
2	0	0	400	400	500	500	
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	700	700	800
4	0	300	400	700	700	850

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

MOCHILA-BOOLEANA (w, v, n, W)

```
para Y ← 0 até W faça
    t[0, Y] ← 0
    para i ← 1 até n faça
        a ← t[i−1, Y]
        se w[i] > Y
            então b ← 0
            senão b ← t[i−1, Y−w[i]] + v[i]
        t[i, Y] ← max{a, b}
devolva t[n, W]
```

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

```
MOCHILA-BOOLEANA (w, v, n, W)
para Y ← 0 até W faça
    t[0, Y] ← 0
    para i ← 1 até n faça
        a ← t[i−1, Y]
        se w[i] > Y
            então b ← 0
            senão b ← t[i−1, Y−w[i]] + v[i]
        t[i, Y] ← max{a, b}
devolva t[n, W]
```

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$,
mais lento que o algoritmo força bruta!

Obtenção da mochila

MOCHILA (w, n, W, t)

```
Y ← W
para i ← n decrescendo até 1 faça
    se t[i, Y] = t[i−1, Y]
        então x[i] ← 0
        senão x[i] ← 1
              Y ← Y − w[i]
devolva x
```

Obtenção da mochila

MOCHILA (w, n, W, t)

```
1  Y ← W
2  para i ← n decrescendo até 1 faça
3      se t[i, Y] = t[i−1, Y]
4          então x[i] ← 0
5          senão x[i] ← 1
6                  Y ← Y − w[i]
7  devolva x
```

Consumo de tempo é $\Theta(n)$.

Versão recursiva

```
MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)
  para i ← 0 até n faça
      para Y ← 0 até W faça
          t[i, Y] ← ∞
  devolva LOOKUP-MOC (w, v, n, W)
```

Versão recursiva

```
LOOKUP-MOC (w, v, i, Y)
    se t[i, Y] < ∞
        então devolva t[i, Y]
    se i = 0 ou Y = 0 então t[i, Y] ← 0
    senão
        se w[i] > Y
            então
                t[i, Y] ← LOOKUP-MOC (w, v, i−1, Y)
            senão
                a ← LOOKUP-MOC (w, v, i−1, Y)
                b ← LOOKUP-MOC (w, v, i−1, Y−w[i]) + v[i]
                t[i, Y] ← max {a, b}
    devolva t[i, Y]
```

Exercício das bandeiras

No dia da Bandeira na Rússia o proprietário de uma loja decidiu decorar a vitrine de sua loja com faixas de tecido das cores branca, azul e vermelha.

Ele deseja satisfazer as seguintes codições: faixas da mesma cor não podem ser colocadas uma ao lado da outra. Uma faixa azul sempre está entre uma branca e uma vermelha, ou uma vermelha e uma branca.

Escreva um programa que, dado o número n de faixas a serem colocadas na vitrine, calcule o número de maneiras de satisfazer as condições do proprietário.

Exemplo: Para $n = 3$, o resultado são as seguintes combinações: BVB, VBV, BAV, VAB.