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 $X[1\,.\,.\,n]$

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 $\Omega(\min\{m, n\})$.

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

Exemplos: X = A B C B D A B
Y = B D C A B A
ssco = B C A

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
O
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 $Y[1\,..\,n{-}1]$.
- 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.

$c[i,j] =$ 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 $X[i] = Y[j]$

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

Recorrência

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

A que classe Ω pertence $T(m,n)$?

Recorrência

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

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

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

onde

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

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

$T(m,n)$ é exponecial

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 ...

$c[4,5]$, $c[3,6]$ 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

i \ j	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0							
3	0				$\star$	$\star$		
4	0				$\star$	??		
5	0							
6	0							
7	0							
8	0							

Simulação

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

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

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 $i \leftarrow 0$ **até** m **faça** $c[i, 0] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça** $c[0, j] \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça**
  para $j \leftarrow 1$ **até** n **faça**
    se $X[i] = Y[j]$
      então $c[i, j] \leftarrow c[i-1, j-1] + 1$
      senão se $c[i-1, j] \geq c[i, j-1]$
        então $c[i, j] \leftarrow c[i-1, j]$
        senão $c[i, j] \leftarrow c[i, j-1]$
devolva $c[m, n]$

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

Conclusão

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

Subsequência comum máxima

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

j i

Algoritmo de programação dinâmica

LEC-LENGTH (X, m, Y, n)
para $i \leftarrow 0$ **até** m **faça** $c[i, 0] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça** $c[0, j] \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça**
  para $j \leftarrow 1$ **até** n **faça**
    se $X[i] = Y[j]$
      então $c[i, j] \leftarrow c[i-1, j-1] + 1$
        $b[i, j] \leftarrow$ "↖"
      senão se $c[i-1, j] \geq c[i, j-1]$
        então $c[i, j] \leftarrow c[i-1, j]$
          $b[i, j] \leftarrow$ "↑"
        senão $c[i, j] \leftarrow c[i, j-1]$
          $b[i, j] \leftarrow$ "←"
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 $i\le j$) 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.