

Aula 12 - Programação Dinâmica

Maior subseqüência Comum

Prof. Marco Aurélio Stefanés
marco em dct.ufms.br
www.dct.ufms.br/~marco

Aula 12 - Programação Dinâmica – p. 1

Método de Programação Dinâmica

- Aplicável a alguns problemas de otimização
- Combina soluções de subproblemas
- Subproblemas não são independentes
- Método tabular
- Para usar Programação Dinâmica devemos
 - (P1) Caracterizar a estrutura da solução ótima
 - (P2) Definir recursivamente o valor da solução ótima
 - (P3) Computar o valor da solução ótima da forma bottom-up
 - (P4) Construir um solução ótima a partir dos valores computados

Aula 12 - Programação Dinâmica – p. 2

Maior Subseqüência Comum - LCS

Uma subseqüência de uma seqüência de símbolos X é uma seqüência X' com zero ou mais símbolos de X removidos. Exemplo: $Z = \{B, C, D, B\}$ é uma subseqüência de $X = \{A, B, C, B, D, A, B\}$

Problema da Maior Subseqüência Comum : Dadas $X[1..m]$ e $Y[1..n]$ encontrar uma Z tal que Z é subseqüência de X e de Y e tem comprimento (número de símbolos) máximo.

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

$LCS(X, Y) = \{B, C, B, A\}$

Aula 12 - Programação Dinâmica – p. 3

Algoritmo Força Bruta

Teste todas as subseqüências de X para ver se ela é também uma subseqüência de Y .

Análise

- Cada seqüência gasta tempo $\Theta(n)$ para ser verificada: encontre o primeiro símbolo da subseqüência de X em Y . A partir deste ponto encontre o segundo símbolo, e assim por diante...
- Há 2^m subseqüências de X para serem verificadas
- Tempo total : $\Theta(n2^m)$ - Exponencial

Aula 12 - Programação Dinâmica – p. 4

Substrutura Ótima

Teorema: Seja $Z = \langle z_1, z_2, \dots, z_k \rangle$ uma LCS de $X = \langle x_1 \dots x_m \rangle$ e $Y = \langle y_1 \dots y_n \rangle$.

1. Se $x_m = y_n$ então $z_k = y_n = x_m$ e Z_{k-1} é uma LCS de X_{m-1} e Y_{n-1}
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que Z_k é uma LCS de x_{m-1} e Y_n
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que Z_k é uma LCS de x_m e Y_{n-1}

Prova:

(1) Caso $z_k \neq x_m$, poderíamos concatenar x_m a Z e teríamos uma subsequência comum maior que $|Z|$, uma contradição, pois Z é uma LCS.

O prefixo Z_{k-1} claramente é um subsequência de X_{m-1} e Y_{n-1} de comprimento $k - 1$. Suponha que haja W uma subsequência comum de X_{m-1} e Y_{n-1} de comprimento $> k - 1$, então Wx_m tem comprimento maior que $|Z|$, uma contradição.

Substrutura Ótima

Prova: Continuação ...

- (2) Caso $z_k \neq x_m$, então Z_k é claramente uma subsequência comum de X_{m-1} e Y . Se houvesse uma subsequência W comum de X_{m-1} e Y maior que Z_k , esta subsequência também seria uma subsequência comum de X_m e Y , e maior que Z_k , a LCS de X e Y , uma contradição.
- (3) Similar ao caso (2)

O teorema mostra que uma LCS de duas seqüência contém dentro dela uma LCS dos prefixos das seqüências. A propriedade da **Substrutura Ótima**.

Substrutura Ótima

Simplificação

- Olhe para a maior subsequência comum a X e Y
- Estenda o algoritmo para que ele encontre a maior subsequência

Dada $X = \langle x_1 \dots x_m \rangle$, o i -ésimo prefixo de X , denotado por X_i , é $X_i = \langle x_1 x_2 \dots x_i \rangle$ para $i = 0, \dots, m$

Considere prefixos de X e Y .

- $C[i, j] = |LCS(X_i, Y_j)|$
- $C[m, n] = |LCS(X, Y)|$

Solução Recursiva

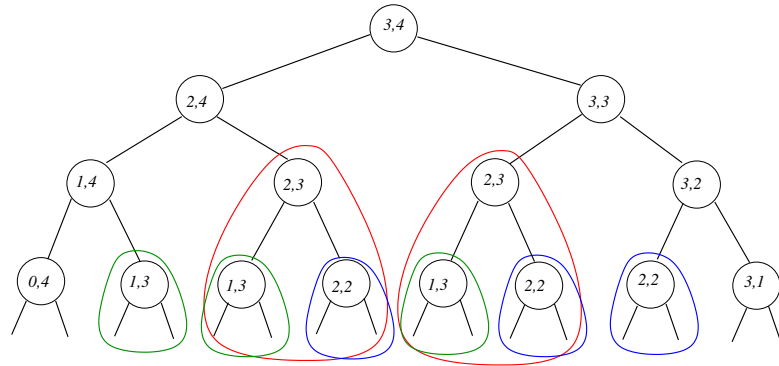
Como podemos calcular $C[m, n]$?

$$c[i, j] = \begin{cases} 0 & \text{se } i, j = 0 \\ c[i - 1, j - 1] + 1 & \text{se } i, j > 0 \text{ e } x_i = y_j \\ \max\{c[i, j - 1], c[i - 1, j]\} & \text{se } i, j > 0 \text{ e } x_i \neq y_j \end{cases}$$

- Podemos escrever um algoritmo recursivo para o problema
- Este algoritmo é ineficiente pela quantidade de subproblemas (Exponencial)
- Há vários subproblemas repetidos

Solução Recursiva

Exemplo para $m = 3$ e $n = 4$



Podemos notar que há sobreposição de subproblemas
Podem ser armazenados em uma tabela para não ser
preciso recalcular

Aula 12 - Programação Dinâmica - p. 9

Computando o tamanho da LCS

$LCS_Length(X, m, Y, n)$

```

1: for  $i = 1$  to  $m$  do
2:    $c[i, 0] = 0$ 
3: for  $j = 0$  to  $n$  do
4:    $c[0, j] = 0$ 
5: for  $i = 1$  to  $m$  do
6:   for  $j = 1$  to  $n$  do
7:     if  $x_i = x_j$  then
8:        $c[i, j] = c[i - 1, j - 1] + 1$ 
9:        $b[i, j] = " \nwarrow "$ 
10:    else
11:      if  $c[i - 1, j] \geq c[i, j - 1]$  then
12:         $c[i, j] = c[i - 1, j]$ 
13:         $b[i, j] = " \uparrow "$ 
14:      else
15:         $c[i, j] = c[i, j - 1]$ 
16:         $b[i, j] = " \leftarrow "$ 

```

Aula 12 - Programação Dinâmica - p. 11

Solução Recursiva

Sobreposição de subproblemas

- Uma solução recursiva contém uma pequena quantidade de subproblemas repetidos muitas vezes
- No caso o número de subproblemas LCS distintos para duas seqüências de tamanho m e n é apenas mn
- Construindo a solução de forma bottom-up podemos calcular os valores e armazená-los em uma matriz. Cálculos subsequentes são evitados usando os valores já previamente calculados.

Aula 12 - Programação Dinâmica - p. 10

Computando o tamanho da LCS

Linhas 1-4

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

Aula 12 - Programação Dinâmica - p. 12

Computando o tamanho da LCS

Linhas 5-16 - $i = 1$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0							
C	0							
A	0							
B	0							
A	0							

Computando o tamanho da LCS

Linhas 5-16 - $i = 3$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0							
B	0							
A	0							

Computando o tamanho da LCS

Linhas 5-16 - $i = 2$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0							
A	0							
B	0							
A	0							

Computando o tamanho da LCS

Linhas 5-16 - $i = 4$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0							
A	0							

Computando o tamanho da LCS

Linhas 5-16 - $i = 5$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0							

Construindo uma LCS

Print_LCS(b, X, i, j)

```
1: if  $i = 0$  ou  $j = 0$  then
2:   pare
3: if  $b[i, j] = "\nwarrow"$  then
4:   Print_LCS( $b, X, i - 1, j - 1$ )
5:   escreva  $x_i$ 
6: else
7:   if  $b[i, j] = "\uparrow"$  then
8:     Print_LCS( $b, X, i - 1, j$ )
9:   else
10:    Print_LCS( $b, X, i, j - 1$ )
```

Chamada Inicial: **Print_LCS**(b, X, m, n)

Computando o tamanho da LCS

Linhas 5-16 - $i = 6$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Construindo a LCS

Linhas 3-8 - $i = 6$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Saída A

Constuindo a LCS

Linhas 3-8 - $i = 5$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Saída A B

Constuindo a LCS

Linhas 3-8 - $i = 3$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Saída A B C B

Constuindo a LCS

Linhas 3-8 - $i = 3$

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Saída A B C

Complexidade do LCS

Complexidade:

- Length_LCS
 - Tempo $\Theta(mn)$
 - Espaço $\Theta(mn)$
- Print_LCS
 - Tempo $\Theta(m + n)$
 - Espaço $O(1)$

Melhorias

- Jogar fora b , podemos calcular em tempo $O(1)$ qual dos valores foi usado para calcular $c[i, j]$
- Podemos usar apenas espaço $\theta(n)$ para calcular o comprimento da LCS