Capítulo I

Programação Dinâmica

(Dynamic Programming)

Técnica da Programação Dinâmica (1)

Qual é o valor de $\mathcal{L}_T(2,5)$?

$$\mathcal{L}_T(i,j) = \begin{cases} 0 & j = 0 \\ T[i][j] & i = 0 \text{ e } j \geq 1 \\ \max\limits_{0 \leq k \leq j} \big(T[i][k] + \mathcal{L}_T(i-1, j-k) \big) & i \geq 1 \text{ e } j \geq 1 \end{cases}$$

T	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	5.5	5.5	5.5
2	0.0	2.0	3.0	5.5	6.0	6.0

Técnica da Programação Dinâmica (2)

Passo 1: Obter memória para tabelar a função

$$\mathcal{L}_T(i,j) = \begin{cases} 0 & j = 0 \\ T[i][j] & i = 0 \text{ e } j \geq 1 \\ \max\limits_{0 \leq k \leq j} \Big(T[i][k] + \mathcal{L}_T(i-1, j-k) \Big) & i \geq 1 \text{ e } j \geq 1 \end{cases}$$

T	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	5.5	5.5	5.5
2	0.0	2.0	3.0	5.5	6.0	6.0

$\mathcal{L}_T$	**0**	**1**	**2**	**3**	**4**	**5**
0						
1						
2						

Técnica da Programação Dinâmica (3)

Passo 2: Tratar a(s) base(s) da recursividade

(Caso $j = 0$, primeira coluna)

$$\mathcal{L}_T(i,j) = \begin{cases} 0 & j = 0 \\ T[i][j] & i = 0 \text{ e } j \geq 1 \\ \max\limits_{0 \leq k \leq j} \big(\, T[i][k] + \mathcal{L}_T(i-1, j-k) \, \big) & i \geq 1 \text{ e } j \geq 1 \end{cases}$$

T	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	5.5	5.5	5.5
2	0.0	2.0	3.0	5.5	6.0	6.0

$\mathcal{L}_T$	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0					
1	0.0					
2	0.0					

Técnica da Programação Dinâmica (4)

Passo 2: Tratar a(s) base(s) da recursividade

(Caso $i = 0$ e $j \geq 1$, restante primeira linha)

$$\mathcal{L}_T(i,j) = \begin{cases} 0 & j = 0 \\ T[i][j] & i = 0 \text{ e } j \geq 1 \\ \max\limits_{0 \leq k \leq j} \Big(T[i][k] + \mathcal{L}_T(i-1, j-k) \Big) & i \geq 1 \text{ e } j \geq 1 \end{cases}$$

T	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	5.5	5.5	5.5
2	0.0	2.0	3.0	5.5	6.0	6.0

$\mathcal{L}_T$	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0					
2	0.0					

Técnica da Programação Dinâmica (5)

Passo 3: Tratar o caso geral

(Caso $i \geq 1$ e $j \geq 1$, tabelação por linhas)

$$\mathcal{L}_T(i,j) = \begin{cases} 0 & j = 0 \\ T[i][j] & i = 0 \ \text{ e } \ j \geq 1 \\ \max\limits_{0 \leq k \leq j} \Big(\, T[i][k] + \mathcal{L}_T(i-1, j-k) \, \Big) & i \geq 1 \ \text{ e } \ j \geq 1 \end{cases}$$

T	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	5.5	5.5	5.5
2	0.0	2.0	3.0	5.5	6.0	6.0

$\mathcal{L}_T$	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	6.5	8.5	9.5
2	0.0					

Técnica da Programação Dinâmica (6)

Passo 3: Tratar o caso geral

(Caso $i \geq 1$ e $j \geq 1$, tabelação por linhas)

$$\mathcal{L}_T(i,j) = \begin{cases} 0 & j = 0 \\ T[i][j] & i = 0 \text{ e } j \geq 1 \\ \max\limits_{0 \leq k \leq j} \Big(T[i][k] + \mathcal{L}_T(i-1, j-k) \Big) & i \geq 1 \text{ e } j \geq 1 \end{cases}$$

T	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	5.5	5.5	5.5
2	0.0	2.0	3.0	5.5	6.0	6.0

$\mathcal{L}_T$	**0**	**1**	**2**	**3**	**4**	**5**
0	0.0	1.5	3.5	4.5	6.0	6.5
1	0.0	2.5	5.0	6.5	8.5	9.5
2	0.0	2.5	5.0	7.0	8.5	10.5

```
double L( double[][] T )
{
    int numRows = T.length;
    int numCols = T[0].length;
    double[][] tabL = new double[numRows][numCols];
    // Column 0: Base case.
    for ( int i = 0;   i < numRows;   i++ )
        tabL[i][0] = 0;
    // Row 0: Base case.
    for ( int j = 1;   j < numCols;   j++ )
        tabL[0][j] = T[0][j];
    // Recursive case; filling by rows.
    .............................................
    return tabL[numRows - 1][numCols - 1];
}
```

```
// Recursive case; filling by rows.
for ( int i = 1;  i < numRows;  i++ )
    for ( int j = 1;  j < numCols;  j++ )
    {
        tabL[i][j] = T[i][0] + tabL[i-1][j];
        for ( int k = 1;  k ≤ j;  k++ )
        {
            double value = T[i][k] + tabL[i-1][j-k];
            if ( value > tabL[i][j] )
                tabL[i][j] = value;
        }
    }
```

Problema dos Caixotes de Morangos

Caixotes de Morangos (1)

O dono de uma pequena cadeia de ($L \geq 1$) mercearias adquiriu ($C \geq 1$) caixotes de morangos e quer saber como deve **distribuir os caixotes pelas lojas** de forma a **maximizar o lucro** que pode obter.

Devido às características de cada loja (localização, capacidade de armazenamento, número médio de clientes, etc.), o lucro esperado com a venda dos morangos varia, não só de loja para loja, como também consoante o número de caixotes enviados para cada loja. Mas o dono sabe estimar o **lucro** obtido com o envio de um número de caixotes para uma determinada loja (para qualquer número de caixotes entre 1 e C e qualquer loja). Estas estimativas estão na tabela **Lucros**.

Por razões administrativas, cada caixote é indivisível (i.e., o seu conteúdo não pode ser repartido por várias lojas). Não é necessário enviar caixotes para todas as lojas.

Caixotes de Morangos (2)

Assuma que há três lojas ($L = 3$), cinco caixotes de morangos ($C = 5$) e que a tabela Lucros tem os seguintes valores (em euros):

Lucros	Número de Caixotes					
	0	**1**	**2**	**3**	**4**	**5**
Loja 0	0.00	1.50	3.50	4.50	6.00	6.50
Loja 1	0.00	2.50	5.00	5.50	5.50	5.50
Loja 2	0.00	2.00	3.00	5.50	6.00	6.00

Se se enviassem **três** caixotes para a Loja 0, **zero** caixotes para a Loja 1 e **dois** caixotes para a Loja 2, o lucro seria:

$$4.50 + 0.00 + 3.00 \ = \ 7.50 \text{ euros.}$$

Qual é o lucro máximo? Como encontrar uma distribuição ótima?

Resolução do Problema (1)

Identificação das lojas:	$0, 1, \ldots, L-1$
Número de caixotes a distribuir:	C
Lucro com o envio para a Loja i de j caixotes:	Lucros$[i][j]$
(com $i = 0, \ldots, L-1$ e $j = 0, \ldots, C$)	

Lucro máximo com o envio para as lojas $0, 1, \ldots, i$ de j caixotes

$$\boxed{\mathcal{L}(i, j)}$$

-
-
-

Resolução do Problema (2)

Identificação das lojas:	$0, 1, \ldots, L-1$
Número de caixotes a distribuir:	C
Lucro com o envio para a Loja i de j caixotes:	Lucros[i][j]
(com $i = 0, \ldots, L-1$ e $j = 0, \ldots, C$)	

Lucro máximo com o envio para as lojas $0, 1, \ldots, i$ **de** j **caixotes**

$$\boxed{\mathcal{L}(i, j)}$$

- (**0 caixotes**) Se $j = 0$, $\quad \mathcal{L}(i, j) = 0$;
-
-

Resolução do Problema (3)

Identificação das lojas:	$0, 1, \ldots, L-1$
Número de caixotes a distribuir:	C
Lucro com o envio para a Loja i de j caixotes:	Lucros[i][j]
(com $i = 0, \ldots, L-1$ e $j = 0, \ldots, C$)	

Lucro máximo com o envio para as lojas $0, 1, \ldots, i$ de j caixotes

$$\boxed{\mathcal{L}(i, j)}$$

- (**0 caixotes**) Se $j = 0$, $\mathcal{L}(i, j) = 0$;
- (**1 só loja**) Se $i = 0$ e $j \geq 1$, $\mathcal{L}(i, j) =$ Lucros[i][j];
-

Resolução do Problema (4)

Identificação das lojas:	$0, 1, \ldots, L-1$
Número de caixotes a distribuir:	C
Lucro com o envio para a Loja i de j caixotes:	Lucros$[i][j]$

(com $i = 0, \ldots, L-1$ e $j = 0, \ldots, C$)

Lucro máximo com o envio para as lojas $0, 1, \ldots, i$ de j caixotes

- (**0 caixotes**) Se $j = 0$, $\mathcal{L}(i, j) = 0$;
- (**1 só loja**) Se $i = 0$ e $j \geq 1$, $\mathcal{L}(i, j) = \text{Lucros}[i][j]$;
- (**Caso geral**) Se $i \geq 1$ e $j \geq 1$, enviam-se k caixotes para a Loja i, para algum $k = 0, 1, \ldots, j$, e os restantes para as lojas $0, 1, \ldots, i-1$.

  Portanto:

$$\mathcal{L}(i, j) = \max_{0 \leq k \leq j} \Big(\text{Lucros}[i][k] + \mathcal{L}(i-1, j-k) \Big).$$

Programação Dinâmica da Função $\mathcal{L}$

$$\mathcal{L}(i,j) = \begin{cases} 0 & j = 0 \\ \text{Lucros}[i][j] & i = 0 \ \text{ e } \ j \geq 1 \\ \max\limits_{0 \leq k \leq j} \Big(\text{Lucros}[i][k] + \mathcal{L}(i-1, j-k) \Big) & i \geq 1 \ \text{ e } \ j \geq 1 \end{cases}$$

Lucros	**0**	**1**	**2**	**3**	**4**	**5**	(Número de Caixotes)
Loja 0	0.0	1.5	3.5	4.5	6.0	6.5	
Loja 1	0.0	2.5	5.0	5.5	5.5	5.5	
Loja 2	0.0	2.0	3.0	5.5	6.0	6.0	

$\mathcal{L}$	**0**	**1**	**2**	**3**	**4**	**5**	(Número de Caixotes)
lojas 0	0.0	1.5	3.5	4.5	6.0	6.5	
lojas 0,**1**	0.0	2.5	5.0	6.5	8.5	9.5	
lojas 0,1,**2**	0.0	2.5	5.0	7.0	8.5	10.5	

Programação Dinâmica da Função $\mathcal{L}$

$$\begin{array}{rcllcll}
\mathcal{L}[2][4] & = & \max(& \underline{\text{Lucros}[2][0] + \mathcal{L}[1][4]} & , & \text{Lucros}[2][1] + \mathcal{L}[1][3] & , \\
& & & \text{Lucros}[2][2] + \mathcal{L}[1][2] & , & \text{Lucros}[2][3] + \mathcal{L}[1][1] & , \\
& & & \text{Lucros}[2][4] + \mathcal{L}[1][0] & & &) \\
& = & \max(& \underline{0.00 + 8.50} & , & 2.00 + 6.50 & , \\
& & & 3.00 + 5.00 & , & 5.50 + 2.50 & , \\
& & & 6.00 + 0.00 & & &) \\
& = & \max(& \underline{8.50} & , & 8.50 & , \\
& & & 8.00 & , & 8.00 & , \\
& & & 6.00 & & &)
\end{array}$$

O que deve ser guardado para se poder construir uma distribuição ótima?

$$
\begin{array}{rcl}
\mathcal{L}[2][4] & = & \max\big(\underline{\text{Lucros}[2][0] + \mathcal{L}[1][4]} \;,\; \text{Lucros}[2][1] + \mathcal{L}[1][3] \;, \\
& & \quad \text{Lucros}[2][2] + \mathcal{L}[1][2] \;,\; \text{Lucros}[2][3] + \mathcal{L}[1][1] \;, \\
& & \quad \text{Lucros}[2][4] + \mathcal{L}[1][0] \;\big) \\
& = & \max\big(\underline{0.00 + 8.50} \;,\; 2.00 + 6.50 \;, \\
& & \quad 3.00 + 5.00 \;,\; 5.50 + 2.50 \;, \\
& & \quad 6.00 + 0.00 \;\big) \\
& = & \max\big(\underline{8.50} \;,\; 8.50 \;, \\
& & \quad 8.00 \;,\; 8.00 \;, \\
& & \quad 6.00 \;\big)
\end{array}
$$

$$\mathcal{D}[2][4] = 0$$

$\mathcal{D}[2][4]$ é o número de caixotes a enviar para a **Loja 2**, quando há **4** caixotes para distribuir pelas **lojas** $0, 1, 2$. (É o valor do "k que maximiza".)

O que deve ser guardado para se poder construir uma distribuição ótima?

$\mathcal{L}$	**0**	**1**	**2**	**3**	**4**	**5**	(Número de Caixotes)
lojas 0	0.0	1.5	3.5	4.5	6.0	6.5	
lojas 0,**1**	0.0	2.5	5.0	6.5	8.5	9.5	
lojas 0,1,**2**	0.0	2.5	5.0	7.0	8.5	10.5	

$\mathcal{D}$	**0**	**1**	**2**	**3**	**4**	**5**	(Número de Caixotes)
lojas 0	0	1	2	3	4	5	
lojas 0,**1**	0	1	2	2	2	2	
lojas 0,1,**2**	0	0	0	1	0	1	

Construção da Distribuição Ótima

$\mathcal{D}$	**0**	**1**	**2**	**3**	**4**	**5**	(Número de Caixotes)
lojas 0	0	1	2	3	4	5	
lojas 0,**1**	0	1	2	2	2	2	
lojas 0,1,**2**	0	0	0	1	0	1	

$\mathcal{D}[2][5] = 1$: 1 caixote para a Loja 2 (Restam $5 - 1 = 4$ caixotes.)

$\mathcal{D}[1][4] = 2$: 2 caixotes para a Loja 1 (Restam $4 - 2 = 2$ caixotes.)

$\mathcal{D}[0][2] = 2$: 2 caixotes para a Loja 0

```
// The values in profit first column (0 crates) are ignored.
Pair<Double,int[]> strawberries( double[][] profit )
{
    int numShops = profit.length;
    int numCols = profit[0].length;
    double[][] maxProfitTab = new double[numShops][numCols];
    int[][] distrTab = new int[numShops][numCols];
    compMaxProfit(profit, maxProfitTab, distrTab);

    double maxProfit = maxProfitTab[numShops − 1][nunCols − 1];

    int[] optDistr = new int[numShops];
    compDistr(distrTab, numShops − 1, numCols − 1, optDistr);

    return new PairClass<Double,int[]>(maxProfit, optDistr);
}
```

```
void compMaxProfit( double[][] profit, double[][] maxProfit,
    int[][] distr )
{
    int numShops = profit.length;
    int numCols = profit[0].length;
    // Column 0 — 0 crates.
    for ( int i = 0;  i < numShops;  i++ )
    {  maxProfit[i][0] = 0;
       distr[i][0] = 0;
    }
    // Row 0 — Only one shop.
    for ( int j = 1;  j < numCols;  j++ )
    {  maxProfit[0][j] = profit[0][j];
       distr[0][j] = j;
    }
    // Remaining cells, filled by rows.
    ..........................................
```

```
    // Remaining cells, filled by rows.
    for ( int i = 1;  i < numShops  i++ )
        for ( int j = 1;  j < numCols;  j++ )
        {
            maxProfit[i][j] = maxProfit[i − 1][j];
            distr[i][j] = 0;
            for ( int k = 1;  k ≤ j;  k++ )
            {
                double value = profit[i][k] + maxProfit[i − 1][j − k];
                if ( value > maxProfit[i][j] )
                {
                    maxProfit[i][j] = value;
                    distr[i][j] = k;
                }
            }
        }
}
```

Complexidade de compMaxProfit

$$(L = \text{profit.length}, \;\; C = \text{profit[0].length} - 1)$$

Primeiro Ciclo $\Theta(L)$

Segundo Ciclo $\Theta(C)$

Terceiro Ciclo

$$\sum_{i=1}^{L-1} \sum_{j=1}^{C} \sum_{k=1}^{j} 1 \quad = \quad \sum_{i=1}^{L-1} \sum_{j=1}^{C} j \quad = \quad \sum_{i=1}^{L-1} \frac{C(C+1)}{2}$$

$$= (L-1)\,\frac{C(C+1)}{2} \quad = \quad \Theta(L\,C^2)$$

Total $\Theta(L\,C^2)$

```
void compDistr( int[][] distrTab, int shop, int crates, int[] optDistr )
{
    if ( shop >= 0 )
    {
        int cratesToShop = distrTab[shop][crates];
        optDistr[shop] = cratesToShop;
        compDistr(distrTab, shop - 1, crates - cratesToShop, optDistr);
    }
}
```

Complexidades

$(L = \text{profit.length},\ \ C = \text{profit[0].length} - 1)$

Temporal

compMaxProfit $\Theta(L\,C^2)$

compDistr (Chamada inicial $(\_, L-1, \_, \_)$) $\Theta(\ L\ )$

$\Theta(L)$ chamadas, cada uma $\Theta(1)$.

strawberries $\Theta(L\,C^2)$

Espacial

strawberries $\Theta(\ L\,C\ )$

Problema

da

Maior Subsequência Comum

(Longest Common Subsequence)

Maior Subsequência Comum

$$b \quad d \quad c \quad \underline{a \quad b \quad a}$$ Subsequência: $a\,b\,a$

$$\underline{a \quad b} \quad c \quad b \quad d \quad \underline{a} \quad b$$ Comprimento: 3

$$\underline{b} \quad d \quad \underline{c \quad a \quad b} \quad a$$ Subsequência: $b\,c\,a\,b$

$$a \quad \underline{b \quad c} \quad b \quad d \quad \underline{a \quad b}$$ Comprimento: 4

Dadas duas sequências

$$X = < x_1 \;\; x_2 \;\; x_3 \;\; \cdots \;\; x_m > \qquad (m \geq 1)$$

$$Y = < y_1 \;\; y_2 \;\; y_3 \;\; \cdots \;\; y_n > \qquad (n \geq 1),$$

calcular uma das subsequências comuns a X e Y de maior comprimento.

Resolução do Problema (1)

$$\begin{array}{lllllll} x_1 & x_2 & x_3 & \cdots & x_m & & (m \geq 1) \\ y_1 & y_2 & y_3 & \cdots & y_n & & (n \geq 1) \end{array}$$

Uma maior subsequência comum a X_i, Y_j, $\mathcal{S}(i,j)$, **e o respetivo comprimento**, $\mathcal{C}(i,j)$

$$\begin{aligned} X_i &= x_1\, x_2 \cdots x_i &= &< x_1\, x_2 \cdots x_{i-1} > x_i \\ Y_j &= y_1\, y_2 \cdots y_j &= &< y_1\, y_2 \cdots y_{j-1} > y_j \end{aligned}$$

- Se $i = 0$ ou $j = 0$,
- Se $i \geq 1$, $j \geq 1$ e $x_i = y_j$,
- Se $i \geq 1$, $j \geq 1$ e $x_i \neq y_j$,

Resolução do Problema (2)

$$\begin{array}{lllllll} x_1 & x_2 & x_3 & \cdots & x_m & & (m \geq 1) \\ y_1 & y_2 & y_3 & \cdots & y_n & & (n \geq 1) \end{array}$$

Uma maior subsequência comum a X_i, Y_j, $\mathcal{S}(i,j)$, **e o respetivo comprimento**, $\mathcal{C}(i,j)$

$$\begin{array}{lclcl} X_i & = & x_1\, x_2 \cdots x_i & = & < x_1\, x_2 \cdots x_{i-1} > x_i \\ Y_j & = & y_1\, y_2 \cdots y_j & = & < y_1\, y_2 \cdots y_{j-1} > y_j \end{array}$$

- Se $i = 0$ ou $j = 0$, $\mathcal{S}(i,j) = \lambda$ e $\mathcal{C}(i,j) = 0$;
- Se $i \geq 1$, $j \geq 1$ e $x_i = y_j$,
- Se $i \geq 1$, $j \geq 1$ e $x_i \neq y_j$,

Resolução do Problema (3)

$$x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_m \qquad (m \geq 1)$$
$$y_1 \quad y_2 \quad y_3 \quad \cdots \quad y_n \qquad (n \geq 1)$$

Uma maior subsequência comum a X_i, Y_j, $\mathcal{S}(i,j)$, **e o respetivo comprimento**, $\mathcal{C}(i,j)$

$$\begin{aligned} X_i &= x_1\, x_2 \cdots x_i &= &< x_1\, x_2 \cdots x_{i-1} > x_i \\ Y_j &= y_1\, y_2 \cdots y_j &= &< y_1\, y_2 \cdots y_{j-1} > y_j \end{aligned}$$

- Se $i = 0$ ou $j = 0$, $\quad \mathcal{S}(i,j) = \lambda$ e $\mathcal{C}(i,j) = 0$;
- Se $i \geq 1$, $j \geq 1$ e $x_i = y_j$, $\quad \mathcal{S}(i,j) = \mathcal{S}(i-1, j-1)\, x_i$
  e $\quad \mathcal{C}(i,j) = \mathcal{C}(i-1, j-1) + 1$;
- Se $i \geq 1$, $j \geq 1$ e $x_i \neq y_j$,

Resolução do Problema (4)

$$\begin{array}{lllll} x_1 & x_2 & x_3 & \cdots & x_m \qquad (m \geq 1) \\ y_1 & y_2 & y_3 & \cdots & y_n \qquad (n \geq 1) \end{array}$$

Uma maior subsequência comum a X_i, Y_j, $\mathcal{S}(i,j)$, **e o respetivo comprimento**, $\mathcal{C}(i,j)$

$$\begin{array}{lllll} X_i & = & x_1\, x_2 \cdots x_i & = & <x_1\, x_2 \cdots x_{i-1}> x_i \\ Y_j & = & y_1\, y_2 \cdots y_j & = & <y_1\, y_2 \cdots y_{j-1}> y_j \end{array}$$

- Se $i = 0$ ou $j = 0$, $\qquad \mathcal{S}(i,j) = \lambda$ e $\mathcal{C}(i,j) = 0$;
- Se $i \geq 1$, $j \geq 1$ e $x_i = y_j$, $\qquad \mathcal{S}(i,j) = \mathcal{S}(i-1, j-1)\, x_i$
  e $\qquad \mathcal{C}(i,j) = \mathcal{C}(i-1, j-1) + 1$;
- Se $i \geq 1$, $j \geq 1$ e $x_i \neq y_j$, $\qquad \mathcal{S}(i,j) = \max\Big(\ \mathcal{S}(i-1, j),\ \mathcal{S}(i, j-1)\ \Big)$
  e $\qquad \mathcal{C}(i,j) = \max\Big(\ \mathcal{C}(i-1, j),\ \mathcal{C}(i, j-1)\ \Big)$.

Programação Dinâmica da Função $\mathcal{C}$

$$\mathcal{C}(i,j) = \begin{cases} 0 & i = 0 \text{ ou } j = 0 \\ 1 + \mathcal{C}(i-1, j-1) & i \geq 1,\ j \geq 1 \text{ e } x_i = y_j \\ \max\Big(\ \mathcal{C}(i-1, j),\ \mathcal{C}(i, j-1)\ \Big) & i \geq 1,\ j \geq 1 \text{ e } x_i \neq y_j \end{cases}$$

			a	**b**	**c**	**b**	**d**	**a**	**b**
	$\mathcal{C}$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	0	0	0	0	0	0	0	0	0
b	**1**	0	0	1	1	1	1	1	1
d	**2**	0	0	1	1	1	2	2	2
c	**3**	0	0	1	2	2	2	2	2
a	**4**	0	1	1	2	2	2	3	3
b	**5**	0	1	2	2	3	3	3	4
a	**6**	0	1	2	2	3	3	4	4

O que deve ser guardado para se poder construir a subsequência?

O caso que se aplicou para calcular $\mathcal{C}(i,j)$. $\mathcal{R}(i,j)$ tem 1, 2 ou 3.

$$\mathcal{C}(i,j) = \begin{cases} 0 & i = 0 \text{ ou } j = 0 \\ \underbrace{1 + \mathcal{C}(i-1, j-1)}_{(1)} & i \geq 1,\ j \geq 1 \text{ e } x_i = y_j \\ \max\left(\underbrace{\mathcal{C}(i-1, j)}_{(2)}, \underbrace{\mathcal{C}(i, j-1)}_{(3)} \right) & i \geq 1,\ j \geq 1 \text{ e } x_i \neq y_j \end{cases}$$

- Se $i = 0$ ou $j = 0$, não é necessário guardar informação;
- Se $i \geq 1$, $j \geq 1$ e $x_i = y_j$, $\mathcal{R}(i,j) = 1$;
- Se $i \geq 1$, $j \geq 1$, $x_i \neq y_j$ e $\mathcal{C}(i-1,j) \geq \mathcal{C}(i,j-1)$, $\mathcal{R}(i,j) = 2$;
- Se $i \geq 1$, $j \geq 1$, $x_i \neq y_j$ e $\mathcal{C}(i-1,j) < \mathcal{C}(i,j-1)$, $\mathcal{R}(i,j) = 3$.

Construção da Subsequência

		a	b	c	b	d	a	b
$\mathcal{R}$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	–	–	–	–	–	–	–	–
b 1	–	2	1	3	1	3	3	1
d 2	–	2	2	2	2	1	3	3
c 3	–	2	2	1	3	2	2	2
a 4	–	1	2	2	2	2	1	3
b 5	–	2	1	2	1	3	2	1
a 6	–	1	2	2	2	2	1	2

Problema	**Pos**	**Regra**	**Subseq** **0 1 2 3**
$(6,7)$	3	$\mathcal{R}[6][7] = 2$	
$(5,7)$	3	$\mathcal{R}[5][7] = 1$	**b**
$(4,6)$	2	$\mathcal{R}[4][6] = 1$	**a b**
$(3,5)$	1	$\mathcal{R}[3][5] = 2$	**a b**
$(2,5)$	1	$\mathcal{R}[2][5] = 1$	**d a b**
$(1,4)$	0	$\mathcal{R}[1][4] = 1$	**b d a b**
$(0,3)$	-1		

```
// The values in seqX[0] and seqY[0] are ignored.
char[] LCS( char[] seqX, char[] seqY )
{
    int[][] maxLength = new int[seqX.length][seqY.length];
    byte[][] lastCase = new byte[seqX.length][seqY.length];
    compMaxLength(seqX, seqY, maxLength, lastCase);

    int xLength = seqX.length − 1;
    int yLength = seqY.length − 1;
    int lcsLength = maxLength[xLength][yLength];

    char[] lcs = new char[lcsLength];
    compLCS(lastCase, seqX, xLength, yLength, lcs, lcsLength − 1);
    return lcs;
}
```

```
void compMaxLength( char[] seqX, char[] seqY, int[][] maxLength,
    byte[][] lastCase )
{
    // Row 0 — seqX is empty.
    for ( int j = 0;   j < seqY.length;   j++ )
        maxLength[0][j] = 0;
    // Column 0 — seqY is empty.
    for ( int i = 1;   i < seqX.length;   i++ )
        maxLength[i][0] = 0;
    // Remaining cells, filled by rows.
    for ( int i = 1;   i < seqX.length;   i++ )
        for ( int j = 1;   j < seqY.length;   j++ )
            ..............................................
}
```

```
// Remaining cells, filled by rows.
for ( int i = 1;  i < seqX.length;  i++ )
    for ( int j = 1;  j < seqY.length;  j++ )
        if ( seqX[i] == seqY[j] )
        {   maxLength[i][j] = 1 + maxLength[i-1][j-1];
            lastCase[i][j] = 1;
        }
        else if ( maxLength[i-1][j] ≥ maxLength[i][j-1] )
        {   maxLength[i][j] = maxLength[i-1][j];
            lastCase[i][j] = 2;
        }
        else
        {   maxLength[i][j] = maxLength[i][j-1];
            lastCase[i][j] = 3;
        }
```

```
void compLCS( char[][] lastCase, char[] seqX, int row, int col,
    char[] lcs, int pos )
{
    if ( row > 0  &&  col > 0 )
        switch ( lastCase[row][col] )
        {
            case 1:  lcs[pos] = seqX[row];
                     compLCS(lastCase, seqX, row − 1, col − 1, lcs, pos − 1);
                     break;
            case 2:  compLCS(lastCase, seqX, row − 1, col, lcs, pos);
                     break;
            case 3:  compLCS(lastCase, seqX, row, col − 1, lcs, pos);
                     break;
        }
}
```

Complexidades

$(m = \text{seqX.length} - 1, \;\; n = \text{seqY.length} - 1)$

Temporal

compMaxLength	$\Theta(\; m\,n \;)$
compLCS (Chamada inicial $(\_, \_, m, n, \_, \_)$)	$O(m+n)$
LCS	$\Theta(\; m\,n \;)$

Espacial

LCS	$\Theta(\; m\,n \;)$

Problema do Produto de uma Cadeia de Matrizes

(Matrix-Chain Multiplication)

Produto de Duas Matrizes

$$\begin{bmatrix} \\ a_{i1} & a_{i2} & \cdots & a_{ik} \\ \\ \end{bmatrix} \begin{bmatrix} b_{1j} \\ b_{2j} \\ \cdot \\ \cdot \\ \cdot \\ b_{kj} \end{bmatrix} = \begin{bmatrix} \\ c_{ij} \\ \\ \end{bmatrix}$$

$$m \times k \qquad\qquad k \times n \qquad\qquad m \times n$$

$$c_{ij} = \sum_{h=1}^{k} a_{ih} b_{hj}$$

$$\textbf{Produto2Matrizes}(m, k, n) = \Theta(m \times k \times n)$$

Produto de uma Cadeia de Matrizes

$$\begin{array}{ccccccc} M_1 & \times & M_2 & \times & M_3 & \times & M_4 \\ 2 \times 50 & & 50 \times 100 & & 100 \times 200 & & 200 \times 30 \end{array}$$

$$\begin{array}{rcr} ((M_1 \times M_2) \times M_3) \times M_4 & \longrightarrow & 62\,000 \text{ produtos} \\ (M_1 \times (M_2 \times M_3)) \times M_4 & \longrightarrow & 1\,032\,000 \text{ produtos} \\ (M_1 \times M_2) \times (M_3 \times M_4) & \longrightarrow & 616\,000 \text{ produtos} \\ M_1 \times ((M_2 \times M_3) \times M_4) & \longrightarrow & 1\,303\,000 \text{ produtos} \\ M_1 \times (M_2 \times (M_3 \times M_4)) & \longrightarrow & 753\,000 \text{ produtos} \end{array}$$

Qual é a ordem pela qual se devem multiplicar as matrizes, de forma a **minimizar** o custo total da operação?

Resolução do Problema (1)

$$\begin{array}{ccccccccccccc} & M_1 & \times & M_2 & \times & M_3 & \times & \cdots & \times & M_n & \\ d_0 & & d_1 & & d_2 & & d_3 & \cdots & d_{n-1} & & d_n \end{array}$$

Núm. de produtos para calcular $M_i \times \cdots \times M_j$ (com $i \leq j$)

- Se $i = j$,

- Se $i < j$,

Resolução do Problema (2)

$$\begin{array}{ccccccccccccc} & M_1 & \times & M_2 & \times & M_3 & \times & \cdots & \times & M_n & \\ d_0 & & d_1 & & d_2 & & d_3 & \cdots & d_{n-1} & & d_n \end{array}$$

Núm. de produtos para calcular $M_i \times \cdots \times M_j$ (com $i \leq j$)

- Se $i = j$, $\quad \text{prod}(i, j) = \text{prod}(i, i) = 0$;
- Se $i < j$,

Resolução do Problema (3)

$$\begin{array}{ccccccccccccc} & M_1 & \times & M_2 & \times & M_3 & \times & \cdots & \times & M_n & \\ d_0 & & d_1 & & d_2 & & d_3 & \cdots & d_{n-1} & & d_n \end{array}$$

Núm. de produtos para calcular $M_i \times \cdots \times M_j$ (com $i \leq j$)

- Se $i = j$, $\quad \mathrm{prod}(i, j) = \mathrm{prod}(i, i) = 0$;
- Se $i < j$, a <u>última operação</u> é $(M_i \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_j)$, para algum $k = i, i+1, \ldots, j-1$, e

$$\mathrm{prod}(i, j) = \mathrm{prod}(i, k) + \mathrm{prod}(k+1, j) + d_{i-1} d_k d_j.$$

Resolução do Problema (4)

$$
\begin{array}{ccccccccccc}
 & M_1 & \times & M_2 & \times & M_3 & \times & \cdots & \times & M_n & \\
d_0 & & d_1 & & d_2 & & d_3 & \cdots & d_{n-1} & & d_n
\end{array}
$$

Núm. de produtos para calcular $M_i \times \cdots \times M_j$ (com $i \leq j$)

- Se $i = j$, $\quad \text{prod}(i,j) = \text{prod}(i,i) = 0$;
- Se $i < j$, a <u>última operação</u> é $(M_i \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_j)$, para algum $k = i, i+1, \ldots, j-1$, e

$$\text{prod}(i,j) = \text{prod}(i,k) + \text{prod}(k+1,j) + d_{i-1} d_k d_j.$$

Núm. mínimo de produtos para calcular $M_i \times \cdots \times M_j$ (com $i \leq j$)

$$\mathcal{P}(i,j) = \begin{cases} 0 & i = j \\ \min\limits_{i \leq k < j} \Big(\mathcal{P}(i,k) + \mathcal{P}(k+1,j) + d_{i-1} d_k d_j \Big) & i < j \end{cases}$$

Programação Dinâmica da Função $\mathcal{P}$

$$\mathcal{P}(i,j) = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \Big(\mathcal{P}(i,k) + \mathcal{P}(k+1,j) + d_{i-1} d_k d_j \Big) & i < j \end{cases}$$

$$\begin{array}{ccccccccc} & M_1 & \times & M_2 & \times & M_3 & \times & M_4 & \\ 2 & & 50 & & 100 & & 200 & & 30 \end{array}$$

$\mathcal{P}_{11} = 0$	$\mathcal{P}_{22} = 0$	$\mathcal{P}_{33} = 0$	$\mathcal{P}_{44} = 0$
$\mathcal{P}_{12} = 10\,000$	$\mathcal{P}_{23} = 1\,000\,000$	$\mathcal{P}_{34} = 600\,000$	
$\mathcal{P}_{13} = 50\,000$	$\mathcal{P}_{24} = 750\,000$		
$\mathcal{P}_{14} = 62\,000$			

O que deve ser guardado?

$\mathcal{P}_{11} = 0$	$\mathcal{P}_{22} = 0$	$\mathcal{P}_{33} = 0$	$\mathcal{P}_{44} = 0$
$\mathcal{P}_{12} = 10\,000$	$\mathcal{P}_{23} = 1\,000\,000$	$\mathcal{P}_{34} = 600\,000$	
$\mathcal{P}_{13} = 50\,000$	$\mathcal{P}_{24} = 750\,000$		
$\mathcal{P}_{14} = 62\,000$			

$$\begin{aligned}
\mathcal{P}[1][3] &= \min(\mathcal{P}[1][1] + \mathcal{P}[2][3] + 2 \times 50 \times 200 \quad , \\
&\qquad \underline{\mathcal{P}[1][2] + \mathcal{P}[3][3] + 2 \times 100 \times 200} \quad) \\
&= \min(\quad 1\,020\,000 \quad , \quad \underline{50\,000} \quad)
\end{aligned}$$

O que deve ser guardado?

$\mathcal{P}_{11} = 0$	$\mathcal{P}_{22} = 0$	$\mathcal{P}_{33} = 0$	$\mathcal{P}_{44} = 0$
$\mathcal{P}_{12} = 10\,000$	$\mathcal{P}_{23} = 1\,000\,000$	$\mathcal{P}_{34} = 600\,000$	
$\mathcal{P}_{13} = 50\,000$	$\mathcal{P}_{24} = 750\,000$		
$\mathcal{P}_{14} = 62\,000$			

$$\begin{aligned}
\mathcal{P}[1][3] &= \min(\ \mathcal{P}[1][1] + \mathcal{P}[2][3] + 2 \times 50 \times 200\ , \\
&\qquad\quad \underline{\mathcal{P}[1][2] + \mathcal{P}[3][3] + 2 \times 100 \times 200}\ ) \\
&= \min(\ 1\,020\,000\ ,\ \underline{50\,000}\ ) \\
\mathcal{U}[1][3] &= 2
\end{aligned}$$

$\mathcal{U}[1][3]$ é o índice da última matriz da esquerda quando se efetua o último produto de duas matrizes para calcular $M_1 \times \cdots \times M_3$. (É o valor do "$k$ que minimiza".)

Determinação da Ordem (Expressão)

$\mathcal{U}_{11} = -$	$\mathcal{U}_{22} = -$	$\mathcal{U}_{33} = -$	$\mathcal{U}_{44} = -$
$\mathcal{U}_{12} = 1$	$\mathcal{U}_{23} = 2$	$\mathcal{U}_{34} = 3$	
$\mathcal{U}_{13} = 2$	$\mathcal{U}_{24} = 2$		
$\mathcal{U}_{14} = 3$			

E(1,4)	$\mathcal{U}[1][4] = 3$	$(E(1,3) \times E(4,4))$
E(1,3)	$\mathcal{U}[1][3] = 2$	$((E(1,2) \times E(3,3)) \times E(4,4))$
E(1,2)	$\mathcal{U}[1][2] = 1$	$(((E(1,1) \times E(2,2)) \times E(3,3)) \times E(4,4))$
E(1,1)		$(((1 \times E(2,2)) \times E(3,3)) \times E(4,4))$
E(2,2)		$(((1 \times 2) \times E(3,3)) \times E(4,4))$
E(3,3)		$(((1 \times 2) \times 3) \times E(4,4))$
E(4,4)		$(((1 \times 2) \times 3) \times 4)$

```
// Matrix i has dims[i − 1] rows and dims[i] columns.
Pair<Long,List<String>> matrixChainMult( int[] dims )
{
   long[][] minProdTab = new long[dims.length][dims.length];
   int[][] lastLeft = new int[dims.length][dims.length];
   compMinProd(dims, minProdTab, lastLeft);

   int numMatrices = dims.length − 1;
   long minProd = minProdTab[1][numMatrices];

   DLL<String> expr = compOrder(lastLeft, 1, numMatrices);
   return new PairClass<Long,List<String>>(minProd, expr);
}
```

DLL<String> abrevia DoublyLinkedList<String>

```
void compMinProd( int[] dims, long[][] minProd, int[][] lastLeft )
{
    // Base case: 1 matrix.
    for ( int i = 1;  i < dims.length;  i++ )
        minProd[i][i] = 0;

    // Recursive case: d is the difference between the indices.
    for ( int d = 1;  d < dims.length - 1;  d++ )
        // i is the left index.
        for ( int i = 1;  i < dims.length - d;  i++ )
        {
            int j = i + d;   // j is the right index.
            int fixedFacts = dims[i - 1] * dims[j];
            ..............................................
        }
}
```

```
// Recursive case: d is the difference between the indices.
for ( int d = 1;  d < dims.length - 1;  d++ )
  for ( int i = 1;  i < dims.length - d;  i++ )  // i is the left index.
  {  int j = i + d;   // j is the right index.
     int fixedFacts = dims[i - 1] * dims[j];
     minProd[i][j] = minProd[i + 1][j] + fixedFacts * dims[i];
     lastLeft[i][j] = i;
     for ( int k = i + 1;  k < j;  k++ )
     {  long value = minProd[i][k] + minProd[k + 1][j] +
                          fixedFacts * dims[k];
        if ( value < minProd[i][j] )
        {  minProd[i][j] = value;
           lastLeft[i][j] = k;
        }
     }
  }
```

Complexidade de compMinProd

$(n = \text{dims.length} - 1)$

Primeiro Ciclo (caso base) $\Theta(n)$

Segundo Ciclo (caso geral)

$$\sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{k=i}^{j-1} 1 \quad = \quad \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} (j - 1 - i + 1) \quad = \quad \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} d$$

$$= \sum_{d=1}^{n-1} (n-d)d \quad = \quad \sum_{d=1}^{n-1} (nd - d^2) \quad = \quad n \sum_{d=1}^{n-1} d \; - \; \sum_{d=1}^{n-1} d^2$$

$$= n \, \frac{(n-1)\, n}{2} \; - \; \frac{(n-1)\, n\, (2n-1)}{6}$$

$$= (\frac{1}{2} n^3 + \cdots) \; - \; (\frac{1}{3} n^3 + \cdots) \quad = \quad \frac{1}{6} n^3 + \cdots \quad = \Theta(n^3)$$

```
DLL<String> compOrder( int[][] lastLeft, int firstPos, int lastPos )
{
   DLL<String> exp = new DLL<String>();
   if ( firstPos == lastPos )
         exp.addLast( Integer.toString(firstPos) );
   else
   {
      int leftPos = lastLeft[firstPos][lastPos];
      DLL<S> leftExp = compOrder(lastLeft, firstPos, leftPos);
      DLL<S> rightExp = compOrder(lastLeft, leftPos+1, lastPos);
      exp.addLast( "(" );        exp.append(leftExp);
      exp.addLast( "x" );        exp.append(rightExp);
      exp.addLast( ")" );
   }
   return exp;
}
```

DLL<S> abrevia DoublyLinkedList<String>

Complexidades

($n =$ dims.length – 1)

Temporal

compMinProd	$\Theta(n^3)$
compOrder (Chamada inicial ($\_, 1, n$))	$\Theta(\ n\ )$
$\Theta(n)$ chamadas, cada uma $\Theta(1)$.	
matrixChainMult	$\Theta(n^3)$

Espacial

matrixChainMult	$\Theta(n^2)$