Capítulo I

Programação Dinâmica

(Dynamic Programming)

Problema
dos
Números de Fibonacci

Números de Fibonacci

Qual é o valor de $\mathcal{F}(7)$?

$$\mathcal{F}(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ \mathcal{F}(n-1) + \mathcal{F}(n-2) & n \geq 2 \end{cases}$$

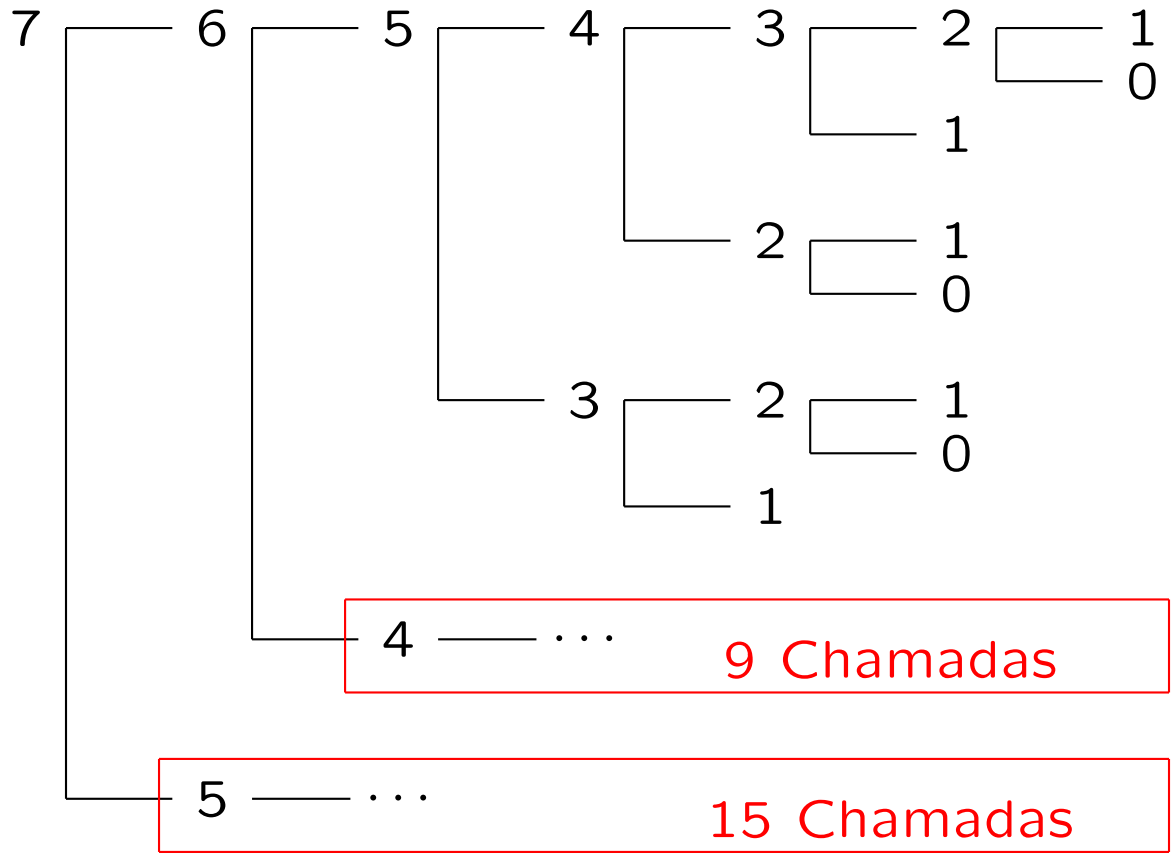
Solução Recursiva por Força Bruta

```
long fibRec( int n ) {
    int res;
    if ( n == 0  ||  n == 1 )
        res = n;
    else
        res = fibRec(n - 1) + fibRec(n - 2);
    return res;
}
```

Qual é a complexidade temporal de fibRec(n)?

Quantas chamadas recursivas, nCR(n), são efetuadas durante a execução de fibRec(n)?

Complexidade da Solução Força Bruta



Facto: nCR(n) é $O(\phi^n)$, onde $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ é o número de ouro.

Portanto, a complexidade temporal de fibRec(n) é exponencial.

Característica do Problema

O número de **sub-problemas distintos** a resolver é polinomial. Para calcular o valor de $\mathcal{F}(n)$ só precisamos dos valores de:

$$\mathcal{F}(n-1),\ \mathcal{F}(n-2),\ \ldots,\ \mathcal{F}(1),\ \mathcal{F}(0)$$

A função fibRec(n) executa **chamadas recursivas repetidas**.

Técnicas de Desenho de Algoritmos Aplicáveis

- **Função-Memória** (*memoization*): algoritmo recursivo (*top-down*)
- **Programação Dinâmica**: algoritmo iterativo (*bottom-up*)

Em ambas, cada sub-problema é resolvido, no máximo, uma vez, guardando-se a solução numa "tabela".

Solução com Função-Memória

```
long fibMem( int n ) {
    long[] tab = new long[n + 1];
    for ( int i = 0;   i ≤ n;   i++ )
        tab[i] = −1;
    return fibMem(tab, n);
}

long fibMem( long[] tab, int n ) {
    if ( tab[n] == −1 ) {
        if ( n == 0  ||  n == 1 )
            tab[n] = n;
        else
            tab[n] = fibMem(tab, n − 1) + fibMem(tab, n − 2);
    }
    return tab[n];
}
```

Complexidade temporal:

$\text{fibMem}(n) = \Theta(n)$

Complexidade espacial:

$\text{fibMem}(n) = \Theta(n)$

Solução com Programação Dinâmica

Aplicação Direta da Técnica

```
long fibDP( int n ) {
    long[] tab = new long[n + 1];
    // Position 0: Base case.
    tab[0] = 0;
    // Position 1: Base case.
    tab[1] = 1;
    // Recursive case.
    for ( int i = 2;   i ≤ n;   i++ )
        tab[i] = tab[i − 1] + tab[i − 2];
    return tab[n];
}
```

Complexidade temporal:

fibDP(n) = $\Theta(n)$

Complexidade espacial:

fibDP(n) = $\Theta(n)$

Solução com Programação Dinâmica

Versão Otimizada

```
long fibIter( int n ) {
    if ( n == 0  ||  n == 1 )
        return n;

    long penultimate;
    long last = 0;
    long current = 1;
    for ( int i = 2;   i ≤ n;   i++ ) {
        penultimate = last;
        last = current;
        current = penultimate + last;
    }
    return current;
}
```

Complexidade temporal:

$\text{fibIter}(n) = \Theta(n)$

Complexidade espacial:

$\text{fibIter}(n) = \Theta(1)$

Problema
dos
Caixotes de Morangos

Lucro Máximo

Caixotes de Morangos — Lucro (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 — Lucro (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?

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 $j = 0, \ldots, C$)	

$\mathcal{L}(i, j)$ **é o lucro máximo com o envio para as lojas** $0, 1, \ldots, i$ **de** j **caixotes:**

-
-
-

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 $j = 0, \ldots, C$)	

$\mathcal{L}(i, j)$ **é o lucro máximo com o envio para as lojas** $0, 1, \ldots, i$ **de** j **caixotes:**

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

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 $j = 0, \ldots, C$)

$\mathcal{L}(i, j)$ **é o 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 $j \geq 1$, $\mathcal{L}(i, j) =$ **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 $j = 0, \ldots, C$)

$\mathcal{L}(i,j)$ **é o 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 $j \geq 1$, $\mathcal{L}(i,j) =$ **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:

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

Resolução do Problema (5)

Qual é o valor de $\mathcal{L}(L-1, C)$?

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

No caso do exemplo, qual é o valor de $\mathcal{L}(2,5)$?

Luc	**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

Programação Dinâmica da Função $\mathcal{L}$ (1)

Passo 1: Obter memória para tabelar a função

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

Luc	**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

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

Programação Dinâmica da Função $\mathcal{L}$ (2)

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

(Caso $j = 0$, primeira coluna)

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

Luc	**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

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

Programação Dinâmica da Função $\mathcal{L}$ (3)

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

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

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

Luc	**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

$\mathcal{L}$	**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					

Programação Dinâmica da Função $\mathcal{L}$ (4)

Passo 3: Tratar o caso geral

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

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

Luc	**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

$\mathcal{L}$	**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					

Programação Dinâmica da Função $\mathcal{L}$ (5)

Passo 3: Tratar o caso geral

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

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

Luc	**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

$\mathcal{L}$	**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

Programação Dinâmica da Função $\mathcal{L}$ (6)

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

```
// The values in profit first column (0 crates) are ignored.
double strawberriesP( double[][] profit ) {
    int nShops = profit.length;
    int nCols = profit[0].length;
    double[][] maxProf = new double[nShops][nCols];
    // Column 0 — 0 crates.
    for ( int i = 0;   i < nShops;   i++ )
        maxProf[i][0] = 0;
    // Row 0 — Only one shop.
    for ( int j = 1;   j < nCols;   j++ )
        maxProf[0][j] = profit[0][j];
    // Remaining cells, filled by rows.
    ..........................................
```

```
    // Remaining cells, filled by rows.
    for ( int i = 1;   i < nShops   i++ )
        for ( int j = 1;   j < nCols;   j++ ) {
            maxProf[i][j] = maxProf[i - 1][j];
            for ( int k = 1;   k <= j;   k++ ) {
                double value = profit[i][k] + maxProf[i - 1][j - k];
                if ( value > maxProf[i][j] )
                    maxProf[i][j] = value;
            }
        }
    return maxProf[nShops - 1][nCols - 1];
}
```

Complexidade Temp. de strawberriesP

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

Primeiro Ciclo $\Theta(L)$

Segundo Ciclo $\Theta(C)$

Terceiro Ciclo

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

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

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

Complexidades

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

Temporal

strawberriesP $\quad\quad \Theta(L\,C^2)$

Espacial

strawberriesP $\quad\quad \Theta(\ L\,C\ )$

Matriz maxProf tem L linhas e $C + 1$ colunas.

Problema
dos
Caixotes de Morangos

Distribuição Ótima

Caixotes de Morangos — Distr.Ot. (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 — Distr.Ot. (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:

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

Como encontrar uma distribuição ótima (i.e. que maximize o lucro)?

Resolução do Problema

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 $j = 0, \ldots, C$)

$\mathcal{L}(i,j)$ **é o 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 $j \geq 1$, $\mathcal{L}(i,j) =$ **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:

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

Programação Dinâmica da Função $\mathcal{L}$ (1)

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

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 $\mathcal{L}$ (2)

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

O que deve ser guardado

para se poder construir uma distribuição ótima?

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

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

$\mathcal{D}[2][4]$ 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?

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

$\mathcal{D}(i,j)$ é o número de caixotes a enviar para a **Loja** i, quando há j caixotes para distribuir pelas **lojas** $0, 1, \ldots, i$.

- (**0 caixotes**) Se $j = 0$, $\mathcal{D}(i,j) = 0$;
- (**1 só loja**) Se $i = 0$ e $j \geq 1$, $\mathcal{D}(i,j) = j$;
- (**Caso geral**) Se $i \geq 1$ e $j \geq 1$, $\mathcal{D}(i,j) = k'$, onde k' é qualquer valor em $\{0, 1, \ldots, j\}$ tal que $\textbf{Lucros}[i][k'] + \mathcal{L}(i-1, j-k') = \mathcal{L}(i,j)$.

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

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

(Número de Caixotes)

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

(Número de Caixotes)

Construção da Distribuição Ótima

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

(Número de Caixotes)

Problema	$\mathcal{D}$	**Distribuição**	
		| | | |	
$(2,5)$	$\mathcal{D}[2][5] = 1$	| | | 1 |	Restam $5-1=4$ caixotes
$(1,4)$	$\mathcal{D}[1][4] = 2$	| | 2 | 1 |	Restam $4-2=2$ caixotes
$(0,2)$	$\mathcal{D}[0][2] = 2$	| 2 | 2 | 1 |	Restam $2-2=0$ caixotes
$(-1,0)$			

```
// The values in profit first column (0 crates) are ignored.
Pair<Double,int[]> strawberriesD( double[][] profit ) {
    int nShops = profit.length;
    int nCols = profit[0].length;
    double[][] maxProfTab = new double[nShops][nCols];
    int[][] distrTab = new int[nShops][nCols];
    compMaxProfit(profit, maxProfTab, distrTab);

    double maxProfit = maxProfTab[nShops - 1][nCols - 1];

    int[] optDistr = new int[nShops];
    compDistr(distrTab, nShops - 1, nCols - 1, optDistr);

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

```
void compMaxProfit( double[][] profit, double[][] maxProfit,
    int[][] distr ) {

    int nShops = profit.length;
    int nCols = profit[0].length;
    // Column 0 — 0 crates.
    for ( int i = 0;   i < nShops;   i++ ) {
        maxProfit[i][0] = 0;
        distr[i][0] = 0;
    }
    // Row 0 — Only one shop.
    for ( int j = 1;   j < nCols;   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 < nShops  i++ )
        for ( int j = 1;  j < nCols;  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 Temp. de compMaxProfit

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

Primeiro Ciclo $\Theta(L)$

Segundo Ciclo $\Theta(C)$

Terceiro Ciclo

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

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

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);
    }
}
```

```
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);
    }
}

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

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/passos, cada uma/um $\Theta(1)$.

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

Espacial

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

Matriz maxProfTab tem L linhas e $C+1$ colunas;

Matriz distrTab tem L linhas e $C+1$ colunas.

Problema
da
Maior Subsequência Comum

(Longest Common Subsequence)

Maior Subsequência Comum

b d c a b a — Subsequência: a b a

a b c b d a b — Comprimento: 3

b d c a b a — Subsequência: b c a b

a b c b d a b — Comprimento: 4

Dadas duas sequências:

$$X = (x_1 \ x_2 \ x_3 \ \cdots \ x_m) \quad (m \geq 1)$$

$$Y = (y_1 \ y_2 \ y_3 \ \cdots \ y_n) \quad (n \geq 1),$$

calcular o comprimento das maiores subsequências comuns a X e Y.

Resolução do Problema (1)

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

$\mathcal{C}(i,j)$ **é o comprimento das maiores subsequências comuns a** X_i e Y_j:

$$X_i = (x_1 \ x_2 \ x_3 \ \cdots \ x_i) \quad (i \geq 0)$$
$$Y_j = (y_1 \ y_2 \ y_3 \ \cdots \ y_j) \quad (j \geq 0)$$

Resolução do Problema (2)

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

$\mathcal{C}(i,j)$ **é o comprimento das maiores subsequências comuns a** X_i e Y_j:

$$X_i = (x_1 \ x_2 \ x_3 \ \cdots \ x_i) \quad (i \geq 0)$$
$$Y_j = (y_1 \ y_2 \ y_3 \ \cdots \ y_j) \quad (j \geq 0)$$

- 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 (3)

$$X = (x_1 \; x_2 \; x_3 \; \cdots \; x_m) \quad (m \geq 1)$$
$$Y = (y_1 \; y_2 \; y_3 \; \cdots \; y_n) \quad (n \geq 1)$$

$\mathcal{C}(i,j)$ **é o comprimento das maiores subsequências comuns a** X_i e Y_j:

$$X_i = (x_1 \; x_2 \; x_3 \; \cdots \; x_i) \quad (i \geq 0)$$
$$Y_j = (y_1 \; y_2 \; y_3 \; \cdots \; y_j) \quad (j \geq 0)$$

- Se $i = 0$ ou $j = 0$, $\mathcal{C}(i,j) = 0$;
- 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 (4)

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

$\mathcal{C}(i,j)$ **é o comprimento das maiores subsequências comuns a** X_i e Y_j:

$$X_i = (x_1 \ x_2 \ x_3 \ \cdots \ x_i) \quad (i \geq 0)$$
$$Y_j = (y_1 \ y_2 \ y_3 \ \cdots \ y_j) \quad (j \geq 0)$$

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

Resolução do Problema (5)

$$X = (x_1 \;\; x_2 \;\; x_3 \;\; \cdots \;\; x_m) \quad (m \geq 1)$$
$$Y = (y_1 \;\; y_2 \;\; y_3 \;\; \cdots \;\; y_n) \quad (n \geq 1)$$

$\mathcal{C}(i,j)$ **é o comprimento das maiores subsequências comuns a** X_i e Y_j:

$$X_i = (x_1 \;\; x_2 \;\; x_3 \;\; \cdots \;\; x_i) \quad (i \geq 0)$$
$$Y_j = (y_1 \;\; y_2 \;\; y_3 \;\; \cdots \;\; y_j) \quad (j \geq 0)$$

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

Resolução do Problema (6)

$$X = (\, x_1 \;\; x_2 \;\; x_3 \;\; \cdots \;\; x_m \,) \qquad (m \geq 1)$$
$$Y = (\, y_1 \;\; y_2 \;\; y_3 \;\; \cdots \;\; y_n \,) \qquad (n \geq 1)$$

Qual é o valor de $\mathcal{C}(m, n)$?

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

No caso do exemplo, qual é o valor de $\mathcal{C}(6, 7)$?

$$X = (\;\text{b}\;\; \text{d}\;\; \text{c}\;\; \text{a}\;\; \text{b}\;\; \text{a}\;)$$
$$Y = (\;\text{a}\;\; \text{b}\;\; \text{c}\;\; \text{b}\;\; \text{d}\;\; \text{a}\;\; \text{b}\;)$$

Programação Dinâmica da Função $\mathcal{C}$

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

			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

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

```
    // 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];
            else if ( maxLength[i - 1][j] >= maxLength[i][j - 1] )
                maxLength[i][j] = maxLength[i - 1][j];
            else
                maxLength[i][j] = maxLength[i][j - 1];

    return maxLength[seqX.length - 1][seqY.length - 1];
}
```

Complexidades

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

Temporal

LCS $\Theta(m\,n)$

Primeiro ciclo: $\Theta(n)$

Segundo ciclo: $\Theta(m)$

Terceiro ciclo: $\Theta(m\,n)$

Espacial

LCS $\Theta(m\,n)$

Matrix maxLength tem $m + 1$ linhas e $n + 1$ colunas.

Problema
da
Multiplicação de uma Cadeia
de Matrizes

(Matrix-Chain Multiplication)

Multiplicação de Duas Matrizes

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

$m \times k$ $\quad$ $k \times n$ $\quad$ $m \times n$

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

Multiplicação2Matrizes$(m, k, n) = \Theta(m \times n \times k)$

Multiplicação de Cadeia de Matrizes

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

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

Dadas as dimensões das matrizes:

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

qual é o número mínimo de produtos para calcular $M_1 \times \cdots \times M_n$?

Resolução do Problema (1)

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

$\mathcal{P}(i,j)$ **é o número mínimo de produtos para calcular** $M_i \times \cdots \times M_j$ (com $i \leq j$):

Resolução do Problema (2)

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

$\mathcal{P}(i,j)$ **é o número mínimo de produtos para calcular** $M_i \times \cdots \times M_j$ (com $i \leq j$):

- Se $i = j$,
- Se $i < j$,

Resolução do Problema (3)

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

$\mathcal{P}(i,j)$ **é o número mínimo de produtos para calcular** $M_i \times \cdots \times M_j$ (com $i \leq j$):

- Se $i = j$, $\mathcal{P}(i,j) = \mathcal{P}(i,i) = 0$;
- Se $i < j$,

Resolução do Problema (4)

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

$\mathcal{P}(i,j)$ **é o número mínimo de produtos para calcular** $M_i \times \cdots \times M_j$ (com $i \leq j$):

- Se $i = j$, $\mathcal{P}(i,j) = \mathcal{P}(i,i) = 0$;
- Se $i < j$, a última operação é $(M_i \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_j)$, para algum $k = i, i+1, \ldots, j-1$, efetuando-se $d_{i-1} d_j d_k$ produtos nessa última operação.

  Portanto:

$$\mathcal{P}(i,j) = \min_{i \leq k < j} \left(\mathcal{P}(i,k) + \mathcal{P}(k+1,j) + d_{i-1} d_j d_k \right)$$

Resolução do Problema (5)

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

Qual é o valor de $\mathcal{P}(1, n)$?

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

No caso do exemplo, qual é o valor de $\mathcal{P}(1, 4)$?

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

Programação Dinâmica da Função $\mathcal{P}$ (1)

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

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

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

Programação Dinâmica da Função $\mathcal{P}$ (2)

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

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

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

```
// Matrix i has dims[i − 1] rows and dims[i] columns.
long matrixChainMult( int[] dims ) {
    long[][] minProd = new long[dims.length][dims.length];
    // 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++ ) {
            // j is the right index.
            int j = i + d;
            ..........................................
```

```
    // 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];
            minProd[i][j] = minProd[i + 1][j] + fixedFacts * dims[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;
            }
        }
    return minProd[1][dims.length - 1];
}
```

Complex. Temp. de matrixChainMult

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

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

Segundo Ciclo (caso geral)

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

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

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

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

Complexidades

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

Temporal

matrixChainMult $\Theta(n^3)$

Espacial

matrixChainMult $\Theta(n^2)$

Matrix minLength tem $n + 1$ linhas e $n + 1$ colunas.

Problema

do

Robotruck

Robotruck (1)

There is a robotic truck that distributes mail packages to several locations in a factory. The robot sits at the end of a conveyer at the mail office and waits for packages to be loaded into its cargo area.

The robot has a **maximum load capacity**, which means that it may have to perform several round trips to complete its task. Provided that the maximum capacity is not exceeded, the robot can stop the conveyer at any time and start a round trip distributing the already collected packages. The **packages must be delivered in the incoming order**.

The **distance** of a round trip is **computed in a grid** by measuring the number of robot moves from the mail office, at location $(0, 0)$, to the location of delivery of the first package, the number of moves between package delivery locations, until the last package, and then the number of moves from the last location back to the mail office. The robot moves a cell at a time either horizontally or vertically in the factory plant grid.

Robotruck (2)

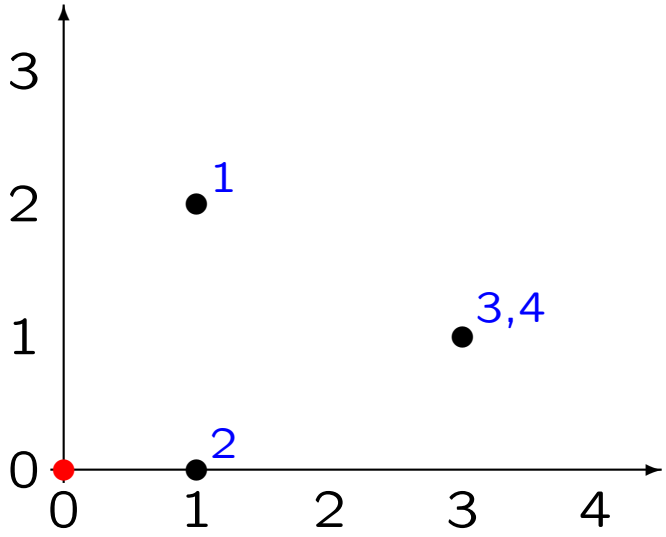
For example, consider four packages, to be delivered at the locations:

$$(1,2),\ \ (1,0),\ \ (3,1),\ \ (3,1).$$

By dividing these packages into 2 round trips of 2 packages each, the number of moves is $6+8=14$:

$$3+2+1=6 \quad \text{(first trip)}$$

$$4+0+4=8 \quad \text{(second trip)}$$



Notice that the two last packages are delivered at the same location and thus the number of moves between them is 0.

Robotruck (3)

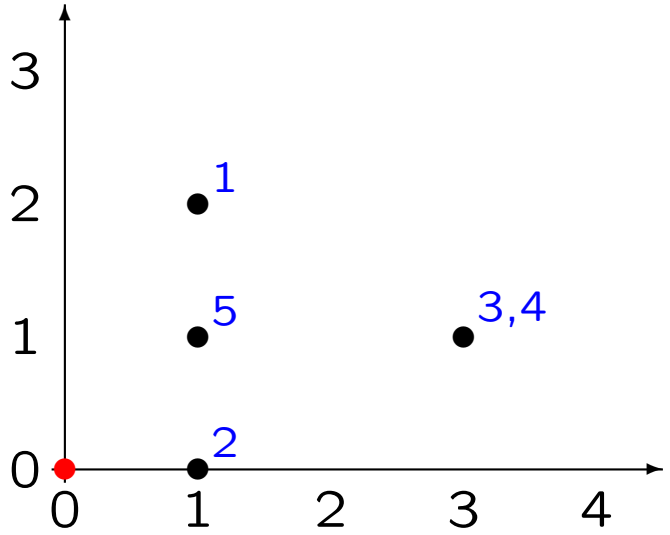
Given:

the robot maximum load capacity: C

the sequence of delivery locations: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)

the package weights: w_1, w_2, ..., w_n

($n \geq 1$, $x_i, y_i \geq 0$ and $1 \leq w_i \leq C$, for every $i = 1, \ldots, n$) compute the minimum distance the robot must travel to deliver all packages.



$C = 10$

Locais: $(1, 2)$, $(1, 0)$, $(3, 1)$, $(3, 1)$, $(1, 1)$

Pesos: 3, 2, 4, 4, 3

Resolução do Problema (1)

Capacidade do robot: C

Locais das entregas: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)

Pesos dos pacotes: w_1, w_2, ..., w_n

Distância entre Locais

$d((x, y), (x', y'))$ **é a distância entre os locais** (x, y) e (x', y'):

$$d((x, y), (x', y')) = |x - x'| + |y - y'|$$

Resolução do Problema (2)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

Distância de *round trip*

$r(i, j)$ **é a distância da viagem completa, da base até à base, para entregar os pacotes** $i, i+1, \ldots, j$ (com $i \leq j$):

$$\begin{aligned} r(i,j) \;=\; & d((0,0),\,(x_i,y_i)) \\ & + \sum_{k=i}^{j-1} d((x_k,y_k),\,(x_{k+1},y_{k+1})) \\ & + d((x_j,y_j),\,(0,0)) \end{aligned}$$

Resolução do Problema (3)

Capacidade do robot: C

Locais das entregas: (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n)

Pesos dos pacotes: w_1, w_2, $\ldots$, w_n

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

Resolução do Problema (4)

Capacidade do robot: C

Locais das entregas: (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n)

Pesos dos pacotes: w_1, w_2, $\ldots$, w_n

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $i = 1$, $\mathcal{D}(i) = r(1, 1)$;

- Se $i = 2$ e $w_1 + w_2 \leq C$,

Resolução do Problema (5)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $i = 1$, $\mathcal{D}(i) = r(1, 1)$;
- Se $i = 2$ e $w_1 + w_2 \leq C$, $\mathcal{D}(i) = r(1, 2)$;
- Se $i = 3$ e $w_1 + w_2 + w_3 \leq C$,

Resolução do Problema (6)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $i = 1$, $\mathcal{D}(i) = r(1, 1)$;
- Se $i = 2$ e $w_1 + w_2 \leq C$, $\mathcal{D}(i) = r(1, 2)$;
- Se $i = 3$ e $w_1 + w_2 + w_3 \leq C$, $\mathcal{D}(i) = r(1, 3)$;
- Se $\left(\sum_{k=1}^{i} w_k \right) \leq C$,

Resolução do Problema (7)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $i = 1$, $\mathcal{D}(i) = r(1, 1)$;
- Se $i = 2$ e $w_1 + w_2 \leq C$, $\mathcal{D}(i) = r(1, 2)$;
- Se $i = 3$ e $w_1 + w_2 + w_3 \leq C$, $\mathcal{D}(i) = r(1, 3)$;
- Se $\left(\sum_{k=1}^{i} w_k \right) \leq C$, $\mathcal{D}(i) = r(1, i)$;

Resolução do Problema (8)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k \right) > C$, quais os pacotes entregues na última viagem?

Resolução do Problema (9)

Capacidade do robot: C

Locais das entregas: (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n)

Pesos dos pacotes: w_1, w_2, $\ldots$, w_n

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k \right) > C$, quais os pacotes entregues na <u>última viagem</u>?
  - apenas o último pacote (uma alternativa que existe sempre)
    Distância mínima:

Resolução do Problema (10)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k\right) > C$, quais os pacotes entregues na última viagem?
  - apenas o último pacote (uma alternativa que existe sempre)
    Distância mínima: $r(i, i) + \mathcal{D}(i-1)$

Resolução do Problema (11)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k\right) > C$, quais os pacotes entregues na última viagem?
  - apenas o último pacote (uma alternativa que existe sempre)
    Distância mínima: $r(i, i) + \mathcal{D}(i-1)$
  - os 2 últimos pacotes, se a soma dos seus pesos não exceder C
    Distância mínima:

Resolução do Problema (12)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k\right) > C$, quais os pacotes entregues na última viagem?
  - apenas o último pacote (uma alternativa que existe sempre)
    Distância mínima: $r(i, i) + \mathcal{D}(i-1)$
  - os 2 últimos pacotes, se a soma dos seus pesos não exceder C
    Distância mínima: $r(i-1, i) + \mathcal{D}(i-2)$

Resolução do Problema (13)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k\right) > C$, quais os pacotes entregues na última viagem?
  - apenas o último pacote (uma alternativa que existe sempre)
    Distância mínima: $r(i, i) + \mathcal{D}(i-1)$
  - os 2 últimos pacotes, se a soma dos seus pesos não exceder C
    Distância mínima: $r(i-1, i) + \mathcal{D}(i-2)$
  - os 3 últimos pacotes, se a soma dos seus pesos não exceder C
    Distância mínina: $r(i-2, i) + \mathcal{D}(i-3)$

Resolução do Problema (14)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k\right) > C$, quais os pacotes entregues na última viagem?
  - apenas o último pacote (uma alternativa que existe sempre)
    Distância mínima: $r(i, i) + \mathcal{D}(i-1)$
  - os 2 últimos pacotes, se a soma dos seus pesos não exceder C
    Distância mínima: $r(i-1, i) + \mathcal{D}(i-2)$
  - os 3 últimos pacotes, se a soma dos seus pesos não exceder C
    Distância mínina: $r(i-2, i) + \mathcal{D}(i-3)$
  - e assim sucessivamente. Qual é a melhor alternativa?

Resolução do Problema (15)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

$\mathcal{D}(i)$ **é a distância mínima para entregar os pacotes** $1, \ldots, i$ $(i \geq 1)$:

- Se $\left(\sum_{k=1}^{i} w_k \right) > C$, a <u>última viagem</u> leva os pacotes $j, j+1, \ldots, i-1, i$, para algum j (entre 2 e i) tal que $w_j + w_{j+1} + \cdots + w_i \leq C$:

$$\begin{aligned} \mathcal{D}(i) = \min \ (\ & r(i,i) + \mathcal{D}(i-1) \ , \\ & r(i-1,i) + \mathcal{D}(i-2) \ , \\ & r(i-2,i) + \mathcal{D}(i-3) \ , \\ & \ldots \ , \\ & r(j,i) + \mathcal{D}(j-1) \ ) \end{aligned}$$

Resolução do Problema (16)

Capacidade do robot: C

Locais das entregas: $(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$

Pesos dos pacotes: $w_1,\ w_2,\ \ldots,\ w_n$

Qual é o valor de $\mathcal{D}(n)$?

$$\mathcal{D}(i) = \begin{cases} r(1,i) & \left(\sum_{k=1}^{i} w_k\right) \leq C \\ \min_{\{j \mid 1<j\leq i \text{ e } \left(\sum_{k=j}^{i} w_k\right)\leq C\}} \left(r(j,i) + \mathcal{D}(j-1) \right) & \text{caso contrário} \end{cases}$$

Para o exemplo seguinte, qual é o valor de $\mathcal{D}(5)$?

$C = 10$ Locais $(1,2)$, $(1,0)$, $(3,1)$, $(3,1)$, $(1,1)$ Pesos 3, 2, 4, 4, 3

Programação Dinâmica da Função $\mathcal{D}$ (1)

$$\mathcal{D}(i) = \begin{cases} r(1,i) & \left(\sum_{k=1}^{i} w_k\right) \leq C \\ \min_{\{j \mid 1<j\leq i \text{ e } \left(\sum_{k=j}^{i} w_k\right)\leq C\}} \left(r(j,i) + \mathcal{D}(j-1) \right) & \text{caso contrário} \end{cases}$$

$C = 10$ Locais $(1,2)$, $(1,0)$, $(3,1)$, $(3,1)$, $(1,1)$ Pesos 3, 2, 4, 4, 3

$\mathcal{D}$

	6	6	12		
0	1	2	3	4	5

$$\mathcal{D}[1] = r(1,1) = 3+3 = 6$$

$$\mathcal{D}[2] = r(1,2) = 3+2+1 = 6$$

$$\mathcal{D}[3] = r(1,3) = 3+2+3+4 = 12$$

Programação Dinâmica da Função $\mathcal{D}$ (2)

$$\mathcal{D}(i) = \begin{cases} r(1,i) & \left(\sum_{k=1}^{i} w_k\right) \leq C \\ \min_{\{j \mid 1<j\leq i \text{ e } \left(\sum_{k=j}^{i} w_k\right)\leq C\}} \left(r(j,i) + \mathcal{D}(j-1) \right) & \text{caso contrário} \end{cases}$$

$C = 10$ Locais $(1,2)$, $(1,0)$, $(3,1)$, $(3,1)$, $(1,1)$ Pesos 3, 2, 4, 4, 3

$\mathcal{D}$

	6	6	12	14	
0	1	2	3	4	5

$$\begin{aligned} \mathcal{D}[4] &= \min(\; r(4,4) + \mathcal{D}[3] \; , \; \underline{r(3,4) + \mathcal{D}[2]} \; , \; \underline{r(2,4) + \mathcal{D}[1]} \;) \\ &= \min(\quad 8 + 12 \quad , \quad \underline{8 + 6} \quad , \quad \underline{8 + 6} \quad) \end{aligned}$$

Programação Dinâmica da Função $\mathcal{D}$ (3)

$$\mathcal{D}(i) = \begin{cases} r(1,i) & \left(\sum_{k=1}^{i} w_k\right) \leq C \\ \min_{\{j \mid 1<j\leq i \text{ e } \left(\sum_{k=j}^{i} w_k\right)\leq C\}} \left(r(j,i) + \mathcal{D}(j-1) \right) & \text{caso contrário} \end{cases}$$

$C = 10$ Locais $(1,2)$, $(1,0)$, $(3,1)$, $(3,1)$, $(1,1)$ Pesos 3, 2, 4, 4, 3

$\mathcal{D}$

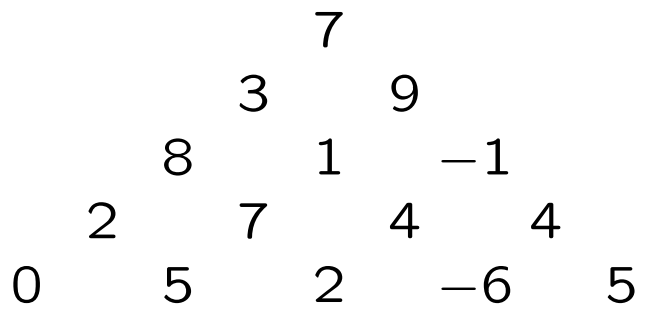
	6	6	12	14	18
0	1	2	3	4	5

$$\mathcal{D}[5] = \min(\underline{r(5,5) + \mathcal{D}[4]} , r(4,5) + \mathcal{D}[3])$$
$$= \min(\underline{4 + 14} , 8 + 12)$$

Problema
do
Jogo da Pirâmide

Jogo da Pirâmide (1)

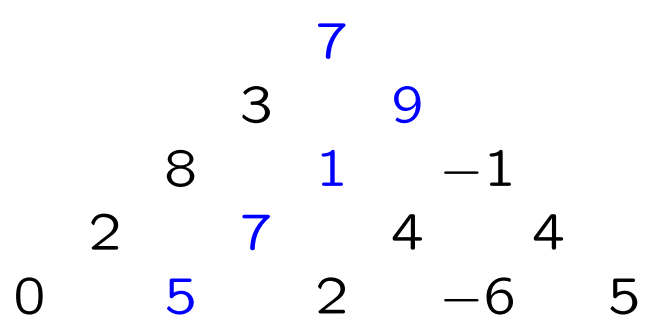
No jogo da Pirâmide, o jogador recebe uma pirâmide de números inteiros. O objetivo é efetuar um **percurso do topo** da pirâmide **até à base** que **maximize a soma** dos números do percurso.



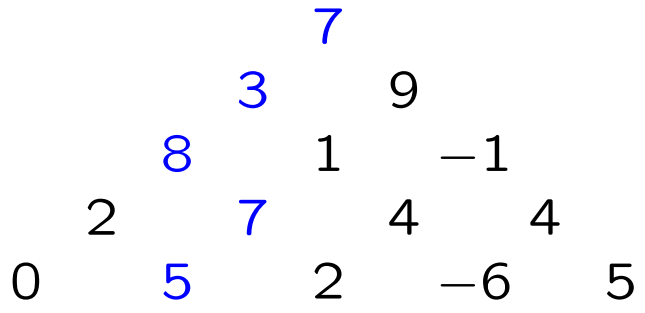
O percurso começa no topo da pirâmide. Em cada ponto do percurso (que não pertença à base da pirâmide), o jogador é obrigado a descer para o número que se encontra imediatamente à esquerda ou imediatamente à direita do ponto onde se encontra. O percurso termina assim que se chega à base da pirâmide.

Jogo da Pirâmide (2)

Considere os dois seguintes percursos na pirâmide (e os seus valores):



Valor 29



Valor 30

Se a pirâmide tiver n níveis ($n \geq 1$), está guardada numa tabela com n linhas e n colunas, como se ilustra na figura da direita.

	0	1	2	3	4
0	7				
1	3	9			
2	8	1	−1		
3	2	7	4	4	
4	0	5	2	−6	5

Dada uma pirâmide, qual é o valor máximo dos percursos do topo até à base?

Resolução do Problema (1)

Pirâmide com $n \geq 1$ níveis: **int**[n][n] P

$\mathcal{V}(i,j)$ **é o valor máximo dos percursos desde a posição** (i,j) **até à base** (com $i = 0, 1, \ldots, n-1$ e $j = 0, 1, \ldots, i$):

Resolução do Problema (2)

Pirâmide com $n \geq 1$ níveis: **int**[n][n] P

$\mathcal{V}(i,j)$ **é o valor máximo dos percursos desde a posição** (i,j) **até à base** (com $i = 0, 1, \ldots, n-1$ e $j = 0, 1, \ldots, i$):

- **(Posição pertence à base)** Se $i = n-1$, $\mathcal{V}(i,j) = P[i][j]$;

Resolução do Problema (3)

Pirâmide com $n \geq 1$ níveis: **int**[n][n] P

$\mathcal{V}(i,j)$ **é o valor máximo dos percursos desde a posição** (i,j) **até à base** (com $i = 0, 1, \ldots, n-1$ e $j = 0, 1, \ldots, i$):

- **(Posição pertence à base)** Se $i = n-1$, $\mathcal{V}(i,j) = P[i][j]$;

- (**Caso geral**) Se $i < n-1$, desce-se pela esquerda ou pela direita.

  Portanto:

$$\begin{aligned} \mathcal{V}(i,j) &= \max\Big(P[i][j] + \mathcal{V}(i+1,j),\ P[i][j] + \mathcal{V}(i+1,j+1) \Big) \\ &= P[i][j] + \max\Big(\mathcal{V}(i+1,j),\ \mathcal{V}(i+1,j+1) \Big). \end{aligned}$$

Resolução do Problema (4)

Pirâmide com $n \geq 1$ níveis: **int**[n][n] P

Qual é o valor de $\mathcal{V}(0,0)$?

$$\mathcal{V}(i,j) = \begin{cases} P[i][j] & i = n-1 \\ & \text{e } 0 \leq j \leq i \\ P[i][j] + \max\left(\mathcal{V}(i+1,j),\ \mathcal{V}(i+1,j+1) \right) & 0 \leq i \leq n-2 \\ & \text{e } 0 \leq j \leq i \end{cases}$$

Programação Dinâmica da Função $\mathcal{V}$ (1)

$$\mathcal{V}(i,j) = \begin{cases} P[i][j] & i = n-1 \\ & \text{e } 0 \leq j \leq i \\ P[i][j] + \max\left(\mathcal{V}(i+1,j),\ \mathcal{V}(i+1,j+1) \right) & 0 \leq i \leq n-2 \\ & \text{e } 0 \leq j \leq i \end{cases}$$

P	0	1	2	3	4
0	7				
1	3	9			
2	8	1	−1		
3	2	7	4	4	
4	0	5	2	−6	5

$\mathcal{V}$	0	1	2	3	4
0					
1					
2					
3					
4	0	5	2	−6	5

$$\mathcal{V}[4][1] = P[4][1]$$

Programação Dinâmica da Função $\mathcal{V}$ (2)

$$\mathcal{V}(i,j) = \begin{cases} P[i][j] & i = n-1 \\ & \text{e } 0 \leq j \leq i \\ P[i][j] + \max\left(\mathcal{V}(i+1,j),\ \mathcal{V}(i+1,j+1) \right) & 0 \leq i \leq n-2 \\ & \text{e } 0 \leq j \leq i \end{cases}$$

P	0	1	2	3	4
0	7				
1	3	9			
2	8	1	−1		
3	2	7	4	4	
4	0	5	2	−6	5

$\mathcal{V}$	0	1	2	3	4
0	30				
1	23	22			
2	20	13	8		
3	7	12	6	9	
4	0	5	2	−6	5

$$\begin{aligned} \mathcal{V}[2][1] &= P[2][1] + \max(\ \underline{\mathcal{V}[3][1]}\ ,\ \mathcal{V}[3][2]\ ) \\ &= 1 + \max(\ \underline{12}\ ,\ 6\ ) \end{aligned}$$