

# Técnicas de Desenho de Algoritmos

Ana Paula Tomás

Desenho e Análise de Algoritmos

Novembro 2019

# Técnicas de desenho de algoritmos

- Pesquisa exaustiva (*exhaustive search*)
- Divisão-e-conquista (*Divide-and-conquer*)
- **Estratégias ávidas**, gananciosas, gulosas (*greedy*)
- **Programação Dinâmica** (*dynamic programming*)
- ...

# Programação Dinâmica (DP)

- Obtém a solução à custa de **soluções de subproblemas** (ou de problemas relacionados).
- A **construção** é muitas vezes realizada **por fases** (como nos algoritmos de Floyd-Warshall, Bellman-Ford, Dijkstra, Prim).
- **Cada subproblema só é resolvido uma vez** e a sua solução é **memorizada** para utilização futura, se necessária.
- DP torna-se particularmente **eficiente** quando a **partilha de subproblemas** entre os subproblemas **é significativa** (não se reduz a “divide-and-conquer”).
- Em **problemas de otimização**: utilizado quando as soluções ótimas têm **subestrutura ótima** (por exemplo, caminho mínimo de  $s$  para  $t$  num grafo), mas não só. Habitualmente, começamos por definir uma **recorrência** para caracterizar **o valor ótimo** e **uma estratégia ótima** em função dos valores e das estratégias para os subproblemas.

# Algoritmo de Floyd-Warshall - aplicação de DP

**Problema:** Distâncias mínimas para todos os pares de nós

Dado um grafo  $G = (V, E, d)$ , com  $d(u, v) > 0$ , para todo  $(u, v) \in E$ , e  $|V| = n$ , determinar o comprimento do caminho mínimo de  $s$  para  $t$ , para **todos os pares**  $(s, t) \in V \times V$ ,  $s \neq t$ .

O algoritmo de Floyd-Warshall resolve o problema em  $\Theta(n^3)$ .

*Supõe-se que os nós do grafo estão numerados de 1 a  $n$*

Inicialmente:  $D_{ij} = 0$ ;  $D_{ij} = d(i, j)$ , se  $i \neq j$  e  $(i, j) \in E$ ; se não,  $D_{ij} = \infty$ .

ALGORITMOFLOYD-WARSHALL( $D, n$ )

Para  $k \leftarrow 1$  até  $n$  fazer

  Para  $i \leftarrow 1$  até  $n$  fazer

    Para  $j \leftarrow 1$  até  $n$  fazer

      Se  $D[i, j] > D[i, k] + D[k, j]$  então  $D[i, j] \leftarrow D[i, k] + D[k, j]$ ;

# Algoritmo de Floyd-Warshall - aplicação de DP

**Problema:** Distâncias mínimas para todos os pares de nós

Dado um grafo  $G = (V, E, d)$ , com  $d(u, v) > 0$ , para todo  $(u, v) \in E$ , e  $|V| = n$ , determinar o comprimento do caminho mínimo de  $s$  para  $t$ , para **todos os pares**  $(s, t) \in V \times V, s \neq t$ .

O **algoritmo de Floyd-Warshall** resolve o problema em  $\Theta(n^3)$ .

*Supõe-se que os nós do grafo estão numerados de 1 a  $n$*

Inicialmente:  $D_{ii} = 0$ ;  $D_{ij} = d(i, j)$ , se  $i \neq j$  e  $(i, j) \in E$ ; se não,  $D_{ij} = \infty$ .

**ALGORITMOFLOYD-WARSHALL**( $D, n$ )

Para  $k \leftarrow 1$  até  $n$  fazer

    Para  $i \leftarrow 1$  até  $n$  fazer

        Para  $j \leftarrow 1$  até  $n$  fazer

            Se  $D[i, j] > D[i, k] + D[k, j]$  então  $D[i, j] \leftarrow D[i, k] + D[k, j]$ ;

# Justificação da correção do algoritmo de Floyd-Warshall

Seja  $G = (V, E, d)$  um grafo dirigido finito, com  $d(e) \in \mathbb{R}^+$ , para todo  $e \in E$ .  
 Suponhamos que os nós estão **numerados de 1 a  $n = |V|$**

Seja  $D_{ij}^{(k)}$  a distância mínima de  $i$  para  $j$  em  $G$  **se os percursos só puderem ter os nós  $1, 2, \dots, k$  como nós intermédios**, com  $k \geq 0$ , fixo.

- Para todo  $(i, j) \in V \times V$ , o valor de  $D_{ij}^{(k)}$  **define-se recursivamente** assim:

$$D_{ij}^{(k)} = \min(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}), \text{ se } k \geq 1$$

$$D_{ij}^{(0)} = \begin{cases} d(i, j), & \text{se } i \neq j \wedge (i, j) \in E \\ \infty & \text{se } i \neq j \wedge (i, j) \notin E \\ 0 & \text{se } i = j \end{cases}$$

- O algoritmo de Floyd-Warshall baseia-se **nessa recorrência** e no facto de  $D_{ij}^{(k+1)} \leq D_{ij}^{(k)}$ , o que permite *dispensar a construção de matrizes auxiliares*.
- A **matriz das distâncias mínimas é  $D_{ij}^{(n)}$** , sendo  $n = |V|$ . Notar que, nesse caso, **qualquer nó de  $V$  pode ser nó intermédio no caminho mínimo**.

# Justificação da correção do algoritmo de Floyd-Warshall

Seja  $G = (V, E, d)$  um grafo dirigido finito, com  $d(e) \in \mathbb{R}^+$ , para todo  $e \in E$ .  
 Suponhamos que os nós estão **numerados de 1 a  $n = |V|$**

Seja  $D_{ij}^{(k)}$  a distância mínima de  $i$  para  $j$  em  $G$  **se os percursos só puderem ter os nós  $1, 2, \dots, k$  como nós intermédios**, com  $k \geq 0$ , fixo.

- Para todo  $(i, j) \in V \times V$ , o valor de  $D_{ij}^{(k)}$  **define-se recursivamente** assim:

$$D_{ij}^{(k)} = \min(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}), \text{ se } k \geq 1$$

$$D_{ij}^{(0)} = \begin{cases} d(i, j), & \text{se } i \neq j \wedge (i, j) \in E \\ \infty & \text{se } i \neq j \wedge (i, j) \notin E \\ 0 & \text{se } i = j \end{cases}$$

- O algoritmo de Floyd-Warshall baseia-se **nessa recorrência** e no facto de  $D_{ij}^{(k+1)} \leq D_{ij}^{(k)}$ , o que permite *dispensar a construção de matrizes auxiliares*.
- A **matriz das distâncias mínimas é  $D_{ij}^{(n)}$** , sendo  $n = |V|$ . Notar que, nesse caso, **qualquer nó de  $V$  pode ser nó intermédio no caminho mínimo**.

# Justificação da correção do algoritmo de Floyd-Warshall

Seja  $G = (V, E, d)$  um grafo dirigido finito, com  $d(e) \in \mathbb{R}^+$ , para todo  $e \in E$ .  
 Suponhamos que os nós estão **numerados de 1 a  $n = |V|$**

Seja  $D_{ij}^{(k)}$  a distância mínima de  $i$  para  $j$  em  $G$  **se os percursos só puderem ter os nós  $1, 2, \dots, k$  como nós intermédios**, com  $k \geq 0$ , fixo.

- Para todo  $(i, j) \in V \times V$ , o valor de  $D_{ij}^{(k)}$  **define-se recursivamente** assim:

$$D_{ij}^{(k)} = \min(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}), \text{ se } k \geq 1$$

$$D_{ij}^{(0)} = \begin{cases} d(i, j), & \text{se } i \neq j \wedge (i, j) \in E \\ \infty & \text{se } i \neq j \wedge (i, j) \notin E \\ 0 & \text{se } i = j \end{cases}$$

- O algoritmo de Floyd-Warshall baseia-se **nessa recorrência** e no facto de  $D_{ij}^{(k+1)} \leq D_{ij}^{(k)}$ , o que permite *dispensar a construção de matrizes auxiliares*.
- A **matriz das distâncias mínimas é  $D_{ij}^{(n)}$** , sendo  $n = |V|$ . Notar que, nesse caso, **qualquer nó de  $V$  pode ser nó intermédio no caminho mínimo**.



# Justificação da correção do algoritmo de Floyd-Warshall

Seja  $G = (V, E, d)$  um grafo dirigido finito, com  $d(e) \in \mathbb{R}^+$ , para todo  $e \in E$ .  
 Suponhamos que os nós estão **numerados de 1 a  $n = |V|$**

Seja  $D_{ij}^{(k)}$  a distância mínima de  $i$  para  $j$  em  $G$  **se os percursos só puderem ter os nós  $1, 2, \dots, k$  como nós intermédios**, com  $k \geq 0$ , fixo.

- Para todo  $(i, j) \in V \times V$ , o valor de  $D_{ij}^{(k)}$  **define-se recursivamente** assim:

$$D_{ij}^{(k)} = \min(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}), \text{ se } k \geq 1$$

$$D_{ij}^{(0)} = \begin{cases} d(i, j), & \text{se } i \neq j \wedge (i, j) \in E \\ \infty & \text{se } i \neq j \wedge (i, j) \notin E \\ 0 & \text{se } i = j \end{cases}$$

- O algoritmo de Floyd-Warshall baseia-se **nessa recorrência** e no facto de  $D_{ij}^{(k+1)} \leq D_{ij}^{(k)}$ , o que permite *dispensar a construção de matrizes auxiliares*.
- A **matriz das distâncias mínimas é  $D_{ij}^{(n)}$** , sendo  $n = |V|$ . Notar que, nesse caso, **qualquer nó de  $V$  pode ser nó intermédio no caminho mínimo**.

# Como caracterizar o caminho mínimo?

**Problema:** Caminhos mínimos para todos os pares de nós

Dado um grafo  $G = (V, E, d)$ , com  $d(u, v) > 0$ , para todo  $(u, v) \in E$ , encontrar um **caminho mínimo** de  $s$  para  $t$ , para todos os pares  $(s, t) \in V \times V$ ,  $s \neq t$ .

Assume-se que os nós estão numerados de 1 a  $n = |V|$ . Seja  $D_{ij}^{(k)}$  a distância mínima de  $i$  para  $j$  em  $G$  **se os percursos só puderem ter os nós  $1, 2, \dots, k$  como nós intermédios**, com  $k \geq 0$ , fixo. Seja  $P_{ij}^{(k)}$  **o nó que precede o nó  $j$  num caminho mínimo de  $i$  para  $j$  nessas condições**. Para todo  $(i, j) \in V \times V$ , o valor de  $P_{ij}^{(k)}$  **define-se recursivamente** assim:

$$P_{ij}^{(k)} = \begin{cases} P_{ij}^{(k-1)} & \text{se } D_{ij}^{(k)} = D_{ij}^{(k-1)} \\ P_{kj}^{(k-1)} & \text{caso contrário (} D_{ij}^{(k)} = D_{ik}^{(k-1)} + D_{kj}^{(k-1)} \text{ e } D_{ij}^{(k)} < D_{ij}^{(k-1)}) \end{cases}$$

$$P_{ij}^{(0)} = \begin{cases} i & \text{se } (i, j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

NB: Em alternativa, podíamos definir  $P_{ij}^{(k)}$  como o nó que segue  $i$  ou como um nó intermédio no caminho.

# Como caracterizar o caminho mínimo?

## Problema: Caminhos mínimos para todos os pares de nós

Dado um grafo  $G = (V, E, d)$ , com  $d(u, v) > 0$ , para todo  $(u, v) \in E$ , encontrar um **caminho mínimo** de  $s$  para  $t$ , para todos os pares  $(s, t) \in V \times V$ ,  $s \neq t$ .

Assume-se que os nós estão numerados de 1 a  $n = |V|$ . Seja  $D_{ij}^{(k)}$  a distância mínima de  $i$  para  $j$  em  $G$  **se os percursos só puderem ter os nós  $1, 2, \dots, k$  como nós intermédios**, com  $k \geq 0$ , fixo. Seja  $P_{ij}^{(k)}$  o **nó que precede o nó  $j$  num caminho mínimo de  $i$  para  $j$  nessas condições**. Para todo  $(i, j) \in V \times V$ , o valor de  $P_{ij}^{(k)}$  **define-se recursivamente** assim:

$$P_{ij}^{(k)} = \begin{cases} P_{ij}^{(k-1)} & \text{se } D_{ij}^{(k)} = D_{ij}^{(k-1)} \\ P_{kj}^{(k-1)} & \text{caso contrário } (D_{ij}^{(k)} = D_{ik}^{(k-1)} + D_{kj}^{(k-1)} \text{ e } D_{ij}^{(k)} < D_{ij}^{(k-1)}) \end{cases}$$

$$P_{ij}^{(0)} = \begin{cases} i & \text{se } (i, j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

NB: Em alternativa, podíamos definir  $P_{ij}^{(k)}$  como o nó que segue  $i$  ou como um nó intermédio no caminho.

# Algoritmo de Floyd-Warshall com cálculo dos caminhos

**ALGORITMO FLOYD-WARSHALL**( $D, P, n$ )

Para  $k \leftarrow 1$  até  $n$  fazer

  Para  $i \leftarrow 1$  até  $n$  fazer

    Para  $j \leftarrow 1$  até  $n$  fazer

      Se  $D[i, j] > D[i, k] + D[k, j]$  então

$D[i, j] \leftarrow D[i, k] + D[k, j];$

$P[i, j] \leftarrow P[k, j];$

**As matrizes  $D$  e  $P$  devem estar já inicializadas na chamada** se, como acima, não se incluir a inicialização na função:

$D[i, i] = 0; D[i, j] = d(i, j)$  se  $(i, j) \in E$  e, caso contrário,  $D[i, j] = \infty$ .

$P[i, i] = 0; P[i, j] = i$  se  $(i, j) \in E$  e, caso contrário,  $P[i, j] = 0$ .

# Construir expressão regular para AF – Método de Kleene

Dado um autómato finito  $A = (S, \Sigma, \delta, s_1, F)$ , com estados numerados de 1 a  $n$ , seja  $r_{ij}^{(k)}$  a expressão que descreve a linguagem determinada pelos percursos de  $i$  para  $j$  que passam quando muito por **estados intermédios etiquetados com números não superiores a  $k$** .

$$r_{ii}^{(0)} = \begin{cases} \varepsilon & \text{sse não existe qualquer lacete em } i \\ \varepsilon + a_1 \dots + a_p & \text{sse os lacetes em } i \text{ estão etiquetados com } a_1, \dots, a_p \end{cases}$$

$$r_{ij}^{(0)} = \begin{cases} \emptyset & \text{sse não existe qualquer arco } (i, j) \\ a_1 + \dots + a_p & \text{sse } a_1, \dots, a_p \text{ etiquetam os arcos } (i, j) \end{cases}$$

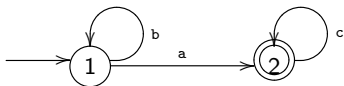
Define-se agora  $r_{ij}^{(k)}$ , para  $k \geq 1$ , recursivamente assim:

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} + r_{ik}^{(k-1)}(r_{kk}^{(k-1)})^*r_{kj}^{(k-1)}$$

onde  $\star$  é o (habitual) fecho de Kleene. A expressão que define a **linguagem reconhecida pelo autómato** é dada por:  $\sum_{s \in F} r_{1s}^{(n)}$

# Método de Kleene para obter expressões regulares para AFs

Muito trabalhoso...



$$r_{11}^{(0)} = \varepsilon + b \quad r_{22}^{(0)} = \varepsilon + c \quad r_{12}^{(0)} = a \quad r_{21}^{(0)} = \emptyset$$

$$r_{11}^{(1)} = r_{11}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^*r_{11}^{(0)} = \varepsilon + b + (\varepsilon + b)(\varepsilon + b)^*(\varepsilon + b) = b^*$$

$$r_{22}^{(1)} = r_{22}^{(0)} + r_{21}^{(0)}(r_{11}^{(0)})^*r_{12}^{(0)} = \varepsilon + c + \emptyset(\varepsilon + b)^*a = \varepsilon + c$$

$$r_{12}^{(1)} = r_{12}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^*r_{12}^{(0)} = a + (\varepsilon + b)(\varepsilon + b)^*a = b^*a$$

$$r_{21}^{(1)} = r_{21}^{(0)} + r_{21}^{(0)}(r_{11}^{(0)})^*r_{11}^{(0)} = \emptyset$$

$$r_{11}^{(2)} = r_{11}^{(1)} + r_{12}^{(1)}(r_{22}^{(1)})^*r_{21}^{(1)} = r_{11}^{(1)} = b^*$$

$$r_{12}^{(2)} = r_{12}^{(1)} + r_{12}^{(1)}(r_{22}^{(1)})^*r_{22}^{(1)} = b^*a + b^*a(\varepsilon + c)^*(\varepsilon + c) = b^*ac^*$$

$$r_{22}^{(2)} = r_{22}^{(1)} + r_{22}^{(1)}(r_{22}^{(1)})^*r_{22}^{(1)} = c^*$$

$$r_{21}^{(2)} = \emptyset$$

Conclusão: a expressão que descreve a linguagem aceite pelo AF é  $r_{12}^{(2)}$  ou seja,  $b^*ac^*$ .

Se 1 e 2 fossem estados finais seria  $r_{11}^{(2)} + r_{12}^{(2)} = b^* + b^*ac^*$ .

# Cálculo do fecho transitivo $R^+$ de uma relação binária $R$

## Recordar que:

- Um grafo dirigido  $G = (V, E)$  representa uma **relação binária**  $R$  definida no conjunto  $V$ . O conjunto de ramos corresponde ao conjunto de pares ordenados que constituem  $R$ . Por definição,  $R \subseteq V \times V$ .
- $R$  é **transitiva** se  $((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$ , para todo  $(x, y, z)$ .
- O **fecho transitivo de  $R$**  denota-se por  $R^+$  e é a menor relação binária definida em  $V$  que é transitiva e contém  $R$ . **Menor** para  $\subseteq$ .
- Usando a **composta de relações**, define-se  $R^1 = R$  e  $R^{i+1} = R^i R = R R^i$ . Por definição de composição,  $u R^i R v$  se existe  $w \in V$  tal que  $u R^i w \wedge w R v$ .

Consequentemente:

- $(x, y) \in R^i$  sse existir um percurso de  $x$  para  $y$  com  $i$  ramos no grafo de  $R$ , para  $i \geq 1$ . É conhecido que  $R^+ = \bigcup_{i=1}^n R^i$ , com  $n = |V|$ .
- $(x, y) \in R^+$  sse existir um percurso de  $x$  para  $y$  no grafo de  $R$ .

# Cálculo do fecho transitivo $R^+$ de uma relação binária $R$

## Recordar que:

- Um grafo dirigido  $G = (V, E)$  representa uma **relação binária**  $R$  definida no conjunto  $V$ . O conjunto de ramos corresponde ao conjunto de pares ordenados que constituem  $R$ . Por definição,  $R \subseteq V \times V$ .
- $R$  é **transitiva** se  $((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$ , para todo  $(x, y, z)$ .
- O **fecho transitivo de  $R$**  denota-se por  $R^+$  e é a menor relação binária definida em  $V$  que é transitiva e contém  $R$ . **Menor** para  $\subseteq$ .
- Usando a **composta de relações**, define-se  $R^1 = R$  e  $R^{i+1} = R^i R = R R^i$ . Por definição de composição,  $u R^i R v$  se existe  $w \in V$  tal que  $u R^i w \wedge w R v$ .

Consequentemente:

- $(x, y) \in R^i$  sse existir um percurso de  $x$  para  $y$  com  $i$  ramos no grafo de  $R$ , para  $i \geq 1$ . É conhecido que  $R^+ = \bigcup_{i=1}^n R^i$ , com  $n = |V|$ .
- $(x, y) \in R^+$  sse existir um percurso de  $x$  para  $y$  no grafo de  $R$ .



# Cálculo do fecho transitivo $R^+$ de uma relação binária $R$

## Recordar que:

- Um grafo dirigido  $G = (V, E)$  representa uma **relação binária**  $R$  definida no conjunto  $V$ . O conjunto de ramos corresponde ao conjunto de pares ordenados que constituem  $R$ . Por definição,  $R \subseteq V \times V$ .
- $R$  é **transitiva** se  $((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$ , para todo  $(x, y, z)$ .
- O **fecho transitivo de  $R$**  denota-se por  $R^+$  e é a menor relação binária definida em  $V$  que é transitiva e contém  $R$ . **Menor** para  $\subseteq$ .
- Usando a **composta de relações**, define-se  $R^1 = R$  e  $R^{i+1} = R^i R = R R^i$ . Por definição de composição,  $u R^i R v$  se existe  $w \in V$  tal que  $u R^i w \wedge w R v$ .

Consequentemente:

- $(x, y) \in R^i$  sse existir um percurso de  $x$  para  $y$  com  $i$  ramos no grafo de  $R$ , para  $i \geq 1$ . É conhecido que  $R^+ = \bigcup_{i=1}^n R^i$ , com  $n = |V|$ .
- $(x, y) \in R^+$  sse existir um percurso de  $x$  para  $y$  no grafo de  $R$ .

# Cálculo do fecho transitivo $R^+$ de uma relação binária $R$

## Recordar que:

- Um grafo dirigido  $G = (V, E)$  representa uma **relação binária**  $R$  definida no conjunto  $V$ . O conjunto de ramos corresponde ao conjunto de pares ordenados que constituem  $R$ . Por definição,  $R \subseteq V \times V$ .
- $R$  é **transitiva** se  $((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$ , para todo  $(x, y, z)$ .
- O **fecho transitivo de  $R$**  denota-se por  $R^+$  e é a menor relação binária definida em  $V$  que é transitiva e contém  $R$ . **Menor** para  $\subseteq$ .
- Usando a **composta de relações**, define-se  $R^1 = R$  e  $R^{i+1} = R^i R = R R^i$ . Por definição de composição,  $u R^i R v$  se existe  $w \in V$  tal que  $u R^i w \wedge w R v$ .

Consequentemente:

- $(x, y) \in R^i$  sse existir um percurso de  $x$  para  $y$  com  $i$  ramos no grafo de  $R$ , para  $i \geq 1$ . É conhecido que  $R^+ = \bigcup_{i=1}^n R^i$ , com  $n = |V|$ .
- $(x, y) \in R^+$  sse existir um percurso de  $x$  para  $y$  no grafo de  $R$ .

# Cálculo do fecho transitivo $R^+$ de uma relação binária $R$

## Recordar que:

- Um grafo dirigido  $G = (V, E)$  representa uma **relação binária**  $R$  definida no conjunto  $V$ . O conjunto de ramos corresponde ao conjunto de pares ordenados que constituem  $R$ . Por definição,  $R \subseteq V \times V$ .
- $R$  é **transitiva** se  $((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$ , para todo  $(x, y, z)$ .
- O **fecho transitivo de  $R$**  denota-se por  $R^+$  e é a menor relação binária definida em  $V$  que é transitiva e contém  $R$ . **Menor** para  $\subseteq$ .
- Usando a **composta de relações**, define-se  $R^1 = R$  e  $R^{i+1} = R^i R = R R^i$ . Por definição de composição,  $u R^i R v$  se existe  $w \in V$  tal que  $u R^i w \wedge w R v$ .

Consequentemente:

- $(x, y) \in R^i$  sse existir um percurso de  $x$  para  $y$  com  $i$  ramos no grafo de  $R$ , para  $i \geq 1$ . É conhecido que  $R^+ = \bigcup_{i=1}^n R^i$ , com  $n = |V|$ .
- $(x, y) \in R^+$  sse existir um percurso de  $x$  para  $y$  no grafo de  $R$ .

# Cálculo do fecho transitivo $R^+$ de uma relação binária $R$

## Recordar que:

- Um grafo dirigido  $G = (V, E)$  representa uma **relação binária**  $R$  definida no conjunto  $V$ . O conjunto de ramos corresponde ao conjunto de pares ordenados que constituem  $R$ . Por definição,  $R \subseteq V \times V$ .
- $R$  é **transitiva** se  $((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$ , para todo  $(x, y, z)$ .
- O **fecho transitivo de  $R$**  denota-se por  $R^+$  e é a menor relação binária definida em  $V$  que é transitiva e contém  $R$ . **Menor** para  $\subseteq$ .
- Usando a **composta de relações**, define-se  $R^1 = R$  e  $R^{i+1} = R^i R = R R^i$ . Por definição de composição,  $u R^i R v$  se existe  $w \in V$  tal que  $u R^i w \wedge w R v$ .

Consequentemente:

- $(x, y) \in R^i$  sse existir um percurso de  $x$  para  $y$  com  $i$  ramos no grafo de  $R$ , para  $i \geq 1$ . É conhecido que  $R^+ = \bigcup_{i=1}^n R^i$ , com  $n = |V|$ .
- $(x, y) \in R^+$  sse existir um percurso de  $x$  para  $y$  no grafo de  $R$ .

# Algoritmo de Warshall para cálculo do fecho transitivo

- A **matriz da relação binária**  $R$  é uma **matriz de booleanos** dada por

$$M_{ij} = \begin{cases} 1 & \text{se } (i, j) \in R \\ 0 & \text{se } (i, j) \notin R \end{cases}$$

- Como no algoritmo de **Floyd-Warshall**, seja  $M_{ij}^{(k)} = 1$  se existir algum percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  **que, quando muito, use nós numerados até  $k$  como nós intermédios**. Supomos que  $V = \{1, \dots, n\}$ .
- Definimos  $M_{ij}^{(0)} = M_{ij}$ . Tem-se  $(i, j) \in R^+$  sse  $M_{ij}^+ = 1$ , sendo  $M_{ij}^+ = M_{ij}^{(n)}$ .

## ALGORITMO WARSHALL( $M, n$ )

```

1 | Para  $k \leftarrow 1$  até  $n$  fazer
2 |   Para  $i \leftarrow 1$  até  $n$  fazer
3 |     Para  $j \leftarrow 1$  até  $n$  fazer
4 |        $M[i, j] \leftarrow M[i, j] \vee (M[i, k] \wedge M[k, j]);$ 

```

Na linha 4 explora propriedades de  $R^+$ . Mais eficiente do que  $M^{(k)}[i, j] \leftarrow M^{(k-1)}[i, j] \vee (M^{(k-1)}[i, k] \wedge M^{(k-1)}[k, j]);$

# Algoritmo de Warshall para cálculo do fecho transitivo

- A **matriz da relação binária**  $R$  é uma **matriz de booleanos** dada por

$$M_{ij} = \begin{cases} 1 & \text{se } (i, j) \in R \\ 0 & \text{se } (i, j) \notin R \end{cases}$$

- Como no algoritmo de **Floyd-Warshall**, seja  $M_{ij}^{(k)} = 1$  se existir algum percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  **que, quando muito, use nós numerados até  $k$  como nós intermédios**. Supomos que  $V = \{1, \dots, n\}$ .
- Definimos  $M_{ij}^{(0)} = M_{ij}$ . Tem-se  $(i, j) \in R^+$  sse  $M_{ij}^+ = 1$ , sendo  $M_{ij}^+ = M_{ij}^{(n)}$ .

## ALGORITMO WARSHALL( $M, n$ )

```

1 | Para  $k \leftarrow 1$  até  $n$  fazer
2 |   Para  $i \leftarrow 1$  até  $n$  fazer
3 |     Para  $j \leftarrow 1$  até  $n$  fazer
4 |        $M[i, j] \leftarrow M[i, j] \vee (M[i, k] \wedge M[k, j]);$ 

```

Na linha 4 explora propriedades de  $R^+$ . Mais eficiente do que  $M^{(k)}[i, j] \leftarrow M^{(k-1)}[i, j] \vee (M^{(k-1)}[i, k] \wedge M^{(k-1)}[k, j]);$

# Algoritmo de Warshall para cálculo do fecho transitivo

- A **matriz da relação binária**  $R$  é uma **matriz de booleanos** dada por

$$M_{ij} = \begin{cases} 1 & \text{se } (i, j) \in R \\ 0 & \text{se } (i, j) \notin R \end{cases}$$

- Como no algoritmo de **Floyd-Warshall**, seja  $M_{ij}^{(k)} = 1$  se existir algum percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  **que, quando muito, use nós numerados até  $k$  como nós intermédios**. Supomos que  $V = \{1, \dots, n\}$ .
- Definimos  $M_{ij}^{(0)} = M_{ij}$ . Tem-se  $(i, j) \in R^+$  sse  $M_{ij}^+ = 1$ , sendo  $M_{ij}^+ = M_{ij}^{(n)}$ .

## ALGORITMO WARSHALL( $M, n$ )

```

1 | Para  $k \leftarrow 1$  até  $n$  fazer
2 |   Para  $i \leftarrow 1$  até  $n$  fazer
3 |     Para  $j \leftarrow 1$  até  $n$  fazer
4 |        $M[i, j] \leftarrow M[i, j] \vee (M[i, k] \wedge M[k, j]);$ 

```

Na linha 4 explora propriedades de  $R^+$ . Mais eficiente do que  $M^{(k)}[i, j] \leftarrow M^{(k-1)}[i, j] \vee (M^{(k-1)}[i, k] \wedge M^{(k-1)}[k, j]);$

# Algoritmo de Warshall para cálculo do fecho transitivo

- A **matriz da relação binária**  $R$  é uma **matriz de booleanos** dada por

$$M_{ij} = \begin{cases} 1 & \text{se } (i, j) \in R \\ 0 & \text{se } (i, j) \notin R \end{cases}$$

- Como no algoritmo de **Floyd-Warshall**, seja  $M_{ij}^{(k)} = 1$  se existir algum percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  **que, quando muito, use nós numerados até  $k$  como nós intermédios**. Supomos que  $V = \{1, \dots, n\}$ .
- Definimos  $M_{ij}^{(0)} = M_{ij}$ . Tem-se  $(i, j) \in R^+$  sse  $M_{ij}^+ = 1$ , sendo  $M_{ij}^+ = M_{ij}^{(n)}$ .

## ALGORITMO WARSHALL( $M, n$ )

```

1 | Para  $k \leftarrow 1$  até  $n$  fazer
2 |   Para  $i \leftarrow 1$  até  $n$  fazer
3 |     Para  $j \leftarrow 1$  até  $n$  fazer
4 |        $M[i, j] \leftarrow M[i, j] \vee (M[i, k] \wedge M[k, j]);$ 

```

Na linha 4 explora propriedades de  $R^+$ . Mais eficiente do que  $M^{(k)}[i, j] \leftarrow M^{(k-1)}[i, j] \vee (M^{(k-1)}[i, k] \wedge M^{(k-1)}[k, j]);$



# Algoritmo de Warshall para cálculo do fecho transitivo

- A **matriz da relação binária**  $R$  é uma **matriz de booleanos** dada por

$$M_{ij} = \begin{cases} 1 & \text{se } (i, j) \in R \\ 0 & \text{se } (i, j) \notin R \end{cases}$$

- Como no algoritmo de **Floyd-Warshall**, seja  $M_{ij}^{(k)} = 1$  se existir algum percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  **que, quando muito, use nós numerados até  $k$  como nós intermédios**. Supomos que  $V = \{1, \dots, n\}$ .
- Definimos  $M_{ij}^{(0)} = M_{ij}$ . Tem-se  $(i, j) \in R^+$  sse  $M_{ij}^+ = 1$ , sendo  $M_{ij}^+ = M_{ij}^{(n)}$ .

## ALGORITMO WARSHALL( $M, n$ )

```

1 | Para  $k \leftarrow 1$  até  $n$  fazer
2 |   Para  $i \leftarrow 1$  até  $n$  fazer
3 |     Para  $j \leftarrow 1$  até  $n$  fazer
4 |        $M[i, j] \leftarrow M[i, j] \vee (M[i, k] \wedge M[k, j]);$ 

```

Na linha 4 explora propriedades de  $R^+$ . Mais eficiente do que  $M^{(k)}[i, j] \leftarrow M^{(k-1)}[i, j] \vee (M^{(k-1)}[i, k] \wedge M^{(k-1)}[k, j]);$

# Contagem de percursos em grafos

**Problema:** calcular o número de percursos de  $v_i$  para  $v_j$ , para todos os pares de nós  $(v_i, v_j)$  de um grafo  $G = (V, E)$ . Assumir que  $V = \{0, \dots, n-1\}$ . O resultado deve ser guardado numa matriz  $M$ , com  $M[i, j] = -1$  se existir uma infinidade de percursos de  $v_i$  para  $v_j$ .

## Resolução:

Seja  $C_{ij}^k$  o número de percursos de  $i$  para  $j$  que apenas podem ter como nós intermédios os numerados até  $k$ , com  $k \geq -1$  fixo.  $C_{ij}^k$  define-se pela recorrência:

$$C_{ij}^{-1} = \begin{cases} 1, & \text{se } (i, j) \in E \\ 0, & \text{se } (i, j) \notin E \end{cases}$$

$$C_{ij}^k = C_{ik}^{k-1} \times (C_{kk}^{k-1})^* \times C_{kj}^{k-1} + C_{ij}^{k-1}$$

em que  $\times$  e  $+$  (extensão das operações habituais a  $\mathbb{R}_0^+ \cup \{\infty\}$ ) e  $\star$  satisfazem:

$$0^* = 1$$

$$\infty \times 0 = 0 \times \infty = 0$$

$$\infty \times y = y \times \infty = \infty, \text{ se } y \neq 0$$

$$y^* = \infty, \text{ se } y \neq 0$$

$$\infty + y = y + \infty = \infty, \text{ para todo } y$$

# Contagem de percursos em grafos

**Problema:** calcular o número de percursos de  $v_i$  para  $v_j$ , para todos os pares de nós  $(v_i, v_j)$  de um grafo  $G = (V, E)$ . Assumir que  $V = \{0, \dots, n-1\}$ . O resultado deve ser guardado numa matriz  $M$ , com  $M[i, j] = -1$  se existir uma infinidade de percursos de  $v_i$  para  $v_j$ .

## Resolução:

Seja  $C_{ij}^k$  o número de percursos de  $i$  para  $j$  que apenas podem ter como nós intermédios os numerados até  $k$ , com  $k \geq -1$  fixo.  $C_{ij}^k$  define-se pela recorrência:

$$C_{ij}^{-1} = \begin{cases} 1, & \text{se } (i, j) \in E \\ 0, & \text{se } (i, j) \notin E \end{cases}$$

$$C_{ij}^k = C_{ik}^{k-1} \times (C_{kk}^{k-1})^* \times C_{kj}^{k-1} + C_{ij}^{k-1}$$

em que  $\times$  e  $+$  (extensão das operações habituais a  $\mathbb{R}_0^+ \cup \{\infty\}$ ) e  $\star$  satisfazem:

$$0^* = 1$$

$$\infty \times 0 = 0 \times \infty = 0$$

$$\infty \times y = y \times \infty = \infty, \text{ se } y \neq 0$$

$$y^* = \infty, \text{ se } y \neq 0$$

$$\infty + y = y + \infty = \infty, \text{ para todo } y$$

# Contagem de percursos em grafos

**Problema:** calcular o número de percursos de  $v_i$  para  $v_j$ , para todos os pares de nós  $(v_i, v_j)$  de um grafo  $G = (V, E)$ . Assumir que  $V = \{0, \dots, n-1\}$ . O resultado deve ser guardado numa matriz  $M$ , com  $M[i, j] = -1$  se existir uma infinidade de percursos de  $v_i$  para  $v_j$ .

## Resolução:

Seja  $C_{ij}^k$  o número de percursos de  $i$  para  $j$  que apenas podem ter como nós intermédios os numerados até  $k$ , com  $k \geq -1$  fixo.  $C_{ij}^k$  define-se pela **recorrência**:

$$C_{ij}^{-1} = \begin{cases} 1, & \text{se } (i, j) \in E \\ 0, & \text{se } (i, j) \notin E \end{cases}$$

$$C_{ij}^k = C_{ik}^{k-1} \times (C_{kk}^{k-1})^* \times C_{kj}^{k-1} + C_{ij}^{k-1}$$

em que  $\times$  e  $+$  (extensão das operações habituais a  $\mathbb{R}_0^+ \cup \{\infty\}$ ) e  $\star$  satisfazem:

$$0^* = 1$$

$$\infty \times 0 = 0 \times \infty = 0$$

$$\infty \times y = y \times \infty = \infty, \text{ se } y \neq 0$$

$$y^* = \infty, \text{ se } y \neq 0$$

$$\infty + y = y + \infty = \infty, \text{ para todo } y$$

# Contagem de percursos – Implementação em C

Se  $M_{ij}^{k-1} = -1$  ou se  $M_{ik}^{k-1} \times M_{kj}^{k-1} \neq 0$  e algum dos valores  $M_{ik}^{k-1}$ ,  $M_{kj}^{k-1}$  e  $M_{kk}^{k-1}$  for -1, então  $M_{ij}^k = -1$ . Nos outros casos,  $M_{ij}^k = M_{ik}^{k-1} \times M_{kj}^{k-1} + M_{ij}^{k-1}$ .

```
void contacaminhos(int n,int M[][MAX])
{ int aux[MAX][MAX], k, i, j;

  for(k=0; k < n ; k++) {
    // copia M para aux
    for (i=0; i < n; i++)
      for (j=0; j < n; j++) aux[i][j] = M[i][j];
    // atualiza a contagem
    for (i=0; i < n; i++)
      for (j=0; j < n; j++)
        if (aux[i][j] != -1) {
          if (aux[i][k]*aux[k][j])
            if(aux[k][k] || aux[i][k]== -1 || aux[k][j]== -1) M[i][j] = -1;
            else M[i][j] += aux[i][k]*aux[k][j];
        }
  }
}
```

## Contagem de percursos – Implementação em C

Se  $M_{ij}^{k-1} = -1$  ou se  $M_{ik}^{k-1} \times M_{kj}^{k-1} \neq 0$  e algum dos valores  $M_{ik}^{k-1}$ ,  $M_{kj}^{k-1}$  e  $M_{kk}^{k-1}$  for -1, então  $M_{ij}^k = -1$ . Nos outros casos,  $M_{ij}^k = M_{ik}^{k-1} \times M_{kj}^{k-1} + M_{ij}^{k-1}$ .

```
void contacaminhos(int n,int M[][MAX])
{ int aux[MAX][MAX], k, i, j;

  for(k=0; k < n ; k++) {
    // copia M para aux
    for (i=0; i < n; i++)
      for (j=0; j < n; j++) aux[i][j] = M[i][j];
    // atualiza a contagem
    for (i=0; i < n; i++)
      for (j=0; j < n; j++)
        if (aux[i][j] != -1) {
          if (aux[i][k]*aux[k][j])
            if(aux[k][k] || aux[i][k]== -1 || aux[k][j]== -1) M[i][j] = -1;
            else M[i][j] += aux[i][k]*aux[k][j];
        }
  }
}
```

# Caixotes de Morangos - aplicação de DP



O dono de uma pequena cadeia de ( $L \geq 1$ ) mercearias adquiriu ( $C \geq 1$ ) caixotes de morangos e tem que decidir quantos caixotes enviar para cada uma das suas lojas, de forma a maximizar o lucro. Devido às características específicas 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. É conhecido o lucro do envio de  $n$  caixotes para cada uma das lojas, para cada  $n \in [0, C]$ . Naturalmente, é nulo se não enviar nenhum caixote. 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. Como efetuar a distribuição?

(Margarida Mamede, UNL, adaptado)

# Caixotes de Morangos

Exemplo de dados:

3	5	
1.50	2.50	2.00
3.50	5.00	3.00
4.50	5.50	5.50
6.00	5.50	6.00
6.50	5.50	6.00

Neste exemplo, tem  $L = 3$  lojas e  $C = 5$  caixotes.

Na coluna  $j$  tem os lucros  $v_{ij}$  do envio de  $i$  caixotes para a loja  $j$ , com  $i = 1, 2, \dots, C$ , e  $j = 1, 2, \dots, L$ .

Admitimos ainda que  $v_{0j} = 0$ , para todo  $j$ .

Seja  $z_{k,j}$  o lucro ótimo se enviar no total  $k$  caixotes para as  $j$  primeiras lojas, com  $k$  e  $j$  fixos. Então,  $z_{k,j}$  é definido pela recorrência:

$$\begin{aligned}
 z_{0,j} &= 0, \text{ para } 1 \leq j \leq L \\
 z_{k,1} &= v_{k,1}, \text{ para } 1 \leq k \leq C \\
 z_{k,j} &= \max_{0 \leq t \leq k} (v_{k-t,j} + z_{t,j-1}), \text{ para } 1 \leq k \leq C, \text{ e } 2 \leq j \leq L,
 \end{aligned}$$

Para  $j \geq 2$ , calculam-se os lucros das soluções que enviam  $k - t$  caixotes à loja  $j$  e distribuem *otimamente* os restantes  $t$  pelas lojas numeradas até  $j - 1$ , para  $0 \leq t \leq k$ .

A solução de maior valor define  $z_{k,j}$ .



# Caixotes de Morangos

Exemplo de dados:

3	5	
1.50	2.50	2.00
3.50	5.00	3.00
4.50	5.50	5.50
6.00	5.50	6.00
6.50	5.50	6.00

Neste exemplo, tem  $L = 3$  lojas e  $C = 5$  caixotes.

Na coluna  $j$  tem os lucros  $v_{ij}$  do envio de  $i$  caixotes para a loja  $j$ , com  $i = 1, 2, \dots, C$ , e  $j = 1, 2, \dots, L$ .

Admitimos ainda que  $v_{0j} = 0$ , para todo  $j$ .

Seja  $z_{k,j}$  o lucro ótimo se enviar no total  $k$  caixotes para as  $j$  primeiras lojas, com  $k$  e  $j$  fixos. Então,  $z_{k,j}$  é definido pela recorrência:

$$\begin{aligned}
 z_{0,j} &= 0, \text{ para } 1 \leq j \leq L \\
 z_{k,1} &= v_{k,1}, \text{ para } 1 \leq k \leq C \\
 z_{k,j} &= \max_{0 \leq t \leq k} (v_{k-t,j} + z_{t,j-1}), \text{ para } 1 \leq k \leq C, \text{ e } 2 \leq j \leq L,
 \end{aligned}$$

Para  $j \geq 2$ , calculam-se os lucros das soluções que enviam  $k - t$  caixotes à loja  $j$  e distribuem *otimamente* os restantes  $t$  pelas lojas numeradas até  $j - 1$ , para  $0 \leq t \leq k$ .

A solução de maior valor define  $z_{k,j}$ .

# Caixotes de Morangos

Exemplo de dados:

3	5	
1.50	2.50	2.00
3.50	5.00	3.00
4.50	5.50	5.50
6.00	5.50	6.00
6.50	5.50	6.00

Neste exemplo, tem  $L = 3$  lojas e  $C = 5$  caixotes.

Na coluna  $j$  tem os lucros  $v_{ij}$  do envio de  $i$  caixotes para a loja  $j$ , com  $i = 1, 2, \dots, C$ , e  $j = 1, 2, \dots, L$ .

Admitimos ainda que  $v_{0j} = 0$ , para todo  $j$ .

Seja  $z_{k,j}$  o lucro ótimo se enviar no total  $k$  caixotes para as  $j$  primeiras lojas, com  $k$  e  $j$  fixos. Então,  $z_{k,j}$  é definido pela recorrência:

$$\begin{aligned}
 z_{0,j} &= 0, \text{ para } 1 \leq j \leq L \\
 z_{k,1} &= v_{k,1}, \text{ para } 1 \leq k \leq C \\
 z_{k,j} &= \max_{0 \leq t \leq k} (v_{k-t,j} + z_{t,j-1}), \text{ para } 1 \leq k \leq C, \text{ e } 2 \leq j \leq L,
 \end{aligned}$$

Para  $j \geq 2$ , calculam-se os lucros das soluções que enviam  $k - t$  caixotes à loja  $j$  e distribuem *otimamente os restantes*  $t$  pelas lojas numeradas até  $j - 1$ , para  $0 \leq t \leq k$ . A solução de maior valor define  $z_{k,j}$ .

# Caixotes de Morangos - Exemplo

3 5

1.50 2.50 2.00

3.50 5.00 3.00

4.50 5.50 5.50

6.00 5.50 6.00

6.50 5.50 6.00

$$z_{0,j} = 0, \text{ para } 1 \leq j \leq L$$

$$z_{k,1} = v_{k,1}, \text{ para } 1 \leq k \leq C$$

$$z_{k,j} = \max_{0 \leq t \leq k} (v_{k-t,j} + z_{t,j-1}), \text{ para } 1 \leq k \leq C, \text{ e } 2 \leq j \leq L,$$

caixas lojas	0	1	2	3	4	5
$L_1$	0.00 0 : $L_1$	1.50 1 : $L_1$	3.50 2 : $L_1$	4.50 3 : $L_1$	6.00 4 : $L_1$	6.50 5 : $L_1$
$L_1, L_2$	0.00 0 : $L_1$ 0 : $L_2$	2.50 0 : $L_1$ 1 : $L_2$	5.00 0 : $L_1$ 2 : $L_2$	6.50 1 : $L_1$ 2 : $L_2$	8.50 2 : $L_1$ 2 : $L_2$	9.50 3 : $L_1$ 2 : $L_2$
$L_1, L_2, L_3$	0.00 0 : $L_1$ 0 : $L_2$ 0 : $L_3$	2.50 0 : $L_1$ 1 : $L_2$ 0 : $L_3$	5.00 0 : $L_1$ 2 : $L_2$ 0 : $L_3$	7.00 0 : $L_1$ 2 : $L_2$ 1 : $L_3$	8.50 1 : $L_1$ 2 : $L_2$ 1 : $L_3$	10.50 0 : $L_1$ 2 : $L_2$ 3 : $L_3$

## Caixotes de Morangos (cont.)

$$z_{0,j} = 0, \text{ para } 1 \leq j \leq L$$

$$z_{k,1} = v_{k,1}, \text{ para } 1 \leq k \leq C$$

$$z_{k,j} = \max_{0 \leq t \leq k} (v_{k-t,j} + z_{t,j-1}), \text{ para } 1 \leq k \leq C, \text{ e } 2 \leq j \leq L$$

- **Algoritmos baseados em programação dinâmica podem gastar muita memória.** É necessário evitar, se possível, gastos de memória excessivos.
- Neste caso, **não precisamos de uma matriz  $(C + 1) \times L$**  para guardar  $z_{k,j}$ , pois  $z_{k,j}$  só depende dos valores de  $z_{t,j-1}$ , para  $t \leq k$ .
- **Bastariam dois arrays com  $C + 1$  posições**, para guardar os valores de  $z_{k,j-1}$  e de  $z_{k,j}$ , para todo  $k$ .
- Se analisarmos com mais cuidado, podemos concluir que, de facto, **basta um array  $Z[\cdot]$** , sendo  $Z[k] = z_{k,j}$ , pois  $z_{k,j}$  só depende dos valores de  $z_{t,j-1}$ , para  $t \leq k$ . Para tal, **na atualização de  $Z[k]$  para um novo  $j$** , tem de se **começar pelo valor mais alto de  $k$** , tomando  $k = C, C - 1, \dots, 2, 1$ .

# Caixotes de Morangos – *DP construção "bottom-up"*

CAIXOTESMORANGOS( $V, L, C, Z$ )

```

0 | Z[0] ← 0;
1 | Para k ← 1 até C fazer Z[k] ← V[k, 1];
2 | Para j ← 2 até L fazer
3 |   Para k ← C até 1 com decremento de 1 fazer
4 |     Para t ← 0 até k - 1 fazer /* NB: inicialmente Z[k] é já V[0, j] + Z[k] */
5 |       Se V[k - t, j] + Z[t] > Z[k] então
6 |         Z[k] ← V[k - t, j] + Z[t];

```

## Complexidade:

Passando  $V$  e  $Z$  por referência e  $C$  e  $L$  por valor, a **complexidade temporal** é  $\Theta(LC^2)$  e a **espacial** (adicional) é  $\Theta(C)$ . "Adicional" porque não contabiliza o espaço  $\Theta(LC)$  ocupado pela matriz de dados  $V$ , mas apenas  $Z$ .

*Justificação (sucinta):* A complexidade temporal do **ciclo 4-6** é  $\Theta(k)$ . Logo, para o **ciclo 3-6** é  $\Theta(\sum_{k=1}^C k) = \Theta(C(C+1)/2) = \Theta(C^2)$  e, portanto, para o **ciclo 2-6** é  $\Theta(LC^2)$ . Assim, o **bloco 1-6** tem complexidade  $\Theta(C + LC^2) = \Theta(LC^2)$ .

Se os dados fossem lidos de um ficheiro e se desse  $V^T$  (a transposta de  $V$ ) em vez de  $V$ , o espaço total podia ser  $\Theta(C)$ .

Ver problema da aula prática: Caixotes de Morangos II (não passar  $V$ ; ler lucro da loja  $j$  dentro da função e atualizar  $Z$ )

# Caixotes de Morangos – *DP construção "bottom-up"*

CAIXOTESMORANGOS( $V, L, C, Z$ )

```

0 | Z[0] ← 0;
1 | Para k ← 1 até C fazer Z[k] ← V[k, 1];
2 | Para j ← 2 até L fazer
3 |   Para k ← C até 1 com decremento de 1 fazer
4 |     Para t ← 0 até k - 1 fazer /* NB: inicialmente Z[k] é já V[0, j] + Z[k] */
5 |       Se V[k - t, j] + Z[t] > Z[k] então
6 |         Z[k] ← V[k - t, j] + Z[t];

```

## Complexidade:

Passando  $V$  e  $Z$  por referência e  $C$  e  $L$  por valor, a **complexidade temporal** é  $\Theta(LC^2)$  e a **espacial** (adicional) é  $\Theta(C)$ . "Adicional" porque não contabiliza o espaço  $\Theta(LC)$  ocupado pela matriz de dados  $V$ , mas apenas  $Z$ .

*Justificação (sucinta):* A complexidade temporal do **ciclo 4-6** é  $\Theta(k)$ . Logo, para o **ciclo 3-6** é  $\Theta(\sum_{k=1}^C k) = \Theta(C(C+1)/2) = \Theta(C^2)$  e, portanto, para o **ciclo 2-6** é  $\Theta(LC^2)$ . Assim, o **bloco 1-6** tem complexidade  $\Theta(C + LC^2) = \Theta(LC^2)$ .

Se os dados fossem lidos de um ficheiro e se desse  $V^T$  (a transposta de  $V$ ) em vez de  $V$ , o espaço total podia ser  $\Theta(C)$ .

Ver problema da aula prática: Caixotes de Morangos II (não passar  $V$ ; ler lucro da loja  $j$  dentro da função e atualizar  $Z$ )

# Problemas de trocos com número de moedas limitado

- O problema "Não lhes dê troco" usa uma **estratégia ávida** (*greedy*) para dar o troco, que nem sempre permite obter o montante pretendido.
- De quantas formas conseguiria obter uma quantia  $Q$  dada se não usar essa estratégia? Seja  $d_k$  o número de moedas disponíveis de valor  $v_k$ , para  $1 \leq k \leq m$ . Admita-se que  $v_k < v_{k+1}$ , para todo  $k < m$ .
- Se puder usar apenas moedas de valor  $v_1, \dots, v_k$ , o número de formas  $N_{q,k}$  de obter  $q$  pode ser definido recursivamente assim:

$$N_{0,k} = 1, \text{ para } 1 \leq k \leq m \text{ (não dar moeda nenhuma se } q = 0)$$

$$N_{q,1} = \begin{cases} 1 & \text{se } q > 0 \wedge q \% v_1 = 0 \wedge d_1 \geq \frac{q}{v_1} \\ 0 & \text{se } q > 0 \wedge (q \% v_1 \neq 0 \vee d_1 < \frac{q}{v_1}) \end{cases}$$

$$N_{q,k} = \sum_{r=0}^{\min(d_k, \lfloor q/v_k \rfloor)} N_{q-rv_k, k-1}, \text{ para todo } q > 0 \text{ e } 1 < k \leq m.$$

O valor procurado é  $N_{Q,m}$ . Dependendo de  $Q$  e dos valores das moedas disponíveis, pode acontecer que nem todos os pares  $(q, k)$ , com  $q \leq Q$ , precisem de ser calculados.

# Problemas de trocos com número de moedas limitado

- O problema "Não lhes dê troco" usa uma **estratégia ávida** (*greedy*) para dar o troco, que nem sempre permite obter o montante pretendido.
- De quantas formas conseguiria obter uma quantia  $Q$  dada se não usar essa estratégia? Seja  $d_k$  o número de moedas disponíveis de valor  $v_k$ , para  $1 \leq k \leq m$ . Admita-se que  $v_k < v_{k+1}$ , para todo  $k < m$ .
- Se puder usar apenas moedas de valor  $v_1, \dots, v_k$ , o número de formas  $N_{q,k}$  de obter  $q$  pode ser definido recursivamente assim:

$$N_{0,k} = 1, \text{ para } 1 \leq k \leq m \text{ (não dar moeda nenhuma se } q = 0 \text{)}$$

$$N_{q,1} = \begin{cases} 1 & \text{se } q > 0 \wedge q \% v_1 = 0 \wedge d_1 \geq \frac{q}{v_1} \\ 0 & \text{se } q > 0 \wedge (q \% v_1 \neq 0 \vee d_1 < \frac{q}{v_1}) \end{cases}$$

$$N_{q,k} = \sum_{r=0}^{\min(d_k, \lfloor q/v_k \rfloor)} N_{q-rv_k, k-1}, \text{ para todo } q > 0 \text{ e } 1 < k \leq m.$$

O valor procurado é  $N_{Q,m}$ . Dependendo de  $Q$  e dos valores das moedas disponíveis, pode acontecer que nem todos os pares  $(q, k)$ , com  $q \leq Q$ , precisem de ser calculados.



# Problemas de trocos com número de moedas limitado

- O problema "Não lhes dê troco" usa uma **estratégia ávida** (*greedy*) para dar o troco, que nem sempre permite obter o montante pretendido.
- De quantas formas conseguiria obter uma quantia  $Q$  dada se não usar essa estratégia? Seja  $d_k$  o número de moedas disponíveis de valor  $v_k$ , para  $1 \leq k \leq m$ . Admita-se que  $v_k < v_{k+1}$ , para todo  $k < m$ .
- Se puder usar apenas moedas de valor  $v_1, \dots, v_k$ , o número de formas  $N_{q,k}$  de obter  $q$  pode ser definido recursivamente assim:

$$N_{0,k} = 1, \text{ para } 1 \leq k \leq m \text{ (não dar moeda nenhuma se } q = 0 \text{)}$$

$$N_{q,1} = \begin{cases} 1 & \text{se } q > 0 \wedge q \% v_1 = 0 \wedge d_1 \geq \frac{q}{v_1} \\ 0 & \text{se } q > 0 \wedge (q \% v_1 \neq 0 \vee d_1 < \frac{q}{v_1}) \end{cases}$$

$$N_{q,k} = \sum_{r=0}^{\min(d_k, \lfloor q/v_k \rfloor)} N_{q-rv_k, k-1}, \text{ para todo } q > 0 \text{ e } 1 < k \leq m.$$

O valor procurado é  $N_{Q,m}$ . Dependendo de  $Q$  e dos valores das moedas disponíveis, pode acontecer que nem todos os pares  $(q, k)$ , com  $q \leq Q$ , precisem de ser calculados.

# Problemas de trocos com número de moedas limitado

- O problema "Não lhes dê troco" usa uma **estratégia ávida** (*greedy*) para dar o troco, que nem sempre permite obter o montante pretendido.
- De quantas formas conseguiria obter uma quantia  $Q$  dada se não usar essa estratégia? Seja  $d_k$  o número de moedas disponíveis de valor  $v_k$ , para  $1 \leq k \leq m$ . Admita-se que  $v_k < v_{k+1}$ , para todo  $k < m$ .
- Se puder usar apenas moedas de valor  $v_1, \dots, v_k$ , o número de formas  $N_{q,k}$  de obter  $q$  pode ser definido recursivamente assim:

$$N_{0,k} = 1, \text{ para } 1 \leq k \leq m \text{ (não dar moeda nenhuma se } q = 0)$$

$$N_{q,1} = \begin{cases} 1 & \text{se } q > 0 \wedge q \% v_1 = 0 \wedge d_1 \geq \frac{q}{v_1} \\ 0 & \text{se } q > 0 \wedge (q \% v_1 \neq 0 \vee d_1 < \frac{q}{v_1}) \end{cases}$$

$$N_{q,k} = \sum_{r=0}^{\min(d_k, \lfloor q/v_k \rfloor)} N_{q-rv_k, k-1}, \text{ para todo } q > 0 \text{ e } 1 < k \leq m.$$

O valor procurado é  $N_{Q,m}$ . Dependendo de  $Q$  e dos valores das moedas disponíveis, pode acontecer que nem todos os pares  $(q, k)$ , com  $q \leq Q$ , precisem de ser calculados.

# Problemas de trocos com número de moedas limitado

- O problema "Não lhes dê troco" usa uma **estratégia ávida** (*greedy*) para dar o troco, que nem sempre permite obter o montante pretendido.
- De quantas formas conseguiria obter uma quantia  $Q$  dada se não usar essa estratégia? Seja  $d_k$  o número de moedas disponíveis de valor  $v_k$ , para  $1 \leq k \leq m$ . Admita-se que  $v_k < v_{k+1}$ , para todo  $k < m$ .
- Se puder usar apenas moedas de valor  $v_1, \dots, v_k$ , o número de formas  $N_{q,k}$  de obter  $q$  pode ser definido recursivamente assim:

$$N_{0,k} = 1, \text{ para } 1 \leq k \leq m \text{ (não dar moeda nenhuma se } q = 0)$$

$$N_{q,1} = \begin{cases} 1 & \text{se } q > 0 \wedge q \% v_1 = 0 \wedge d_1 \geq \frac{q}{v_1} \\ 0 & \text{se } q > 0 \wedge (q \% v_1 \neq 0 \vee d_1 < \frac{q}{v_1}) \end{cases}$$

$$N_{q,k} = \sum_{r=0}^{\min(d_k, \lfloor q/v_k \rfloor)} N_{q-rv_k, k-1}, \text{ para todo } q > 0 \text{ e } 1 < k \leq m.$$

O valor procurado é  $N_{Q,m}$ . Dependendo de  $Q$  e dos valores das moedas disponíveis, pode acontecer que nem todos os pares  $(q, k)$ , com  $q \leq Q$ , precisem de ser calculados.

# Problemas de trocos com número de moedas limitado

- O problema "Não lhes dê troco" usa uma **estratégia ávida** (*greedy*) para dar o troco, que nem sempre permite obter o montante pretendido.
- De quantas formas conseguiria obter uma quantia  $Q$  dada se não usar essa estratégia? Seja  $d_k$  o número de moedas disponíveis de valor  $v_k$ , para  $1 \leq k \leq m$ . Admita-se que  $v_k < v_{k+1}$ , para todo  $k < m$ .
- Se puder usar apenas moedas de valor  $v_1, \dots, v_k$ , o número de formas  $N_{q,k}$  de obter  $q$  pode ser definido recursivamente assim:

$$N_{0,k} = 1, \text{ para } 1 \leq k \leq m \text{ (não dar moeda nenhuma se } q = 0)$$

$$N_{q,1} = \begin{cases} 1 & \text{se } q > 0 \wedge q \% v_1 = 0 \wedge d_1 \geq \frac{q}{v_1} \\ 0 & \text{se } q > 0 \wedge (q \% v_1 \neq 0 \vee d_1 < \frac{q}{v_1}) \end{cases}$$

$$N_{q,k} = \sum_{r=0}^{\min(d_k, \lfloor q/v_k \rfloor)} N_{q-rv_k, k-1}, \text{ para todo } q > 0 \text{ e } 1 < k \leq m.$$

O valor procurado é  $N_{Q,m}$ . Dependendo de  $Q$  e dos valores das moedas disponíveis, pode acontecer que nem todos os pares  $(q, k)$ , com  $q \leq Q$ , precisem de ser calculados.

# Problemas de trocos com número de moedas limitado

## Abordagem "Top-Down" com memoização

```

CONTASOLS( $v, d, q, k$ )    /* chamar CONTASOLS( $v, d, Q, m$ ) para obter  $N_{Q,m}$  */
1  | Se  $q = 0$  então retorna 1;
2  | Se  $k = 1$  então
3  |   Se  $q \% v[1] \neq 0 \vee d[1] < q/v[1]$  então retorna 0;
4  |   retorna 1;
5  | Se  $N[q, k]$  já calculado então retorna  $N[q, k]$ ;
6  |  $rmax \leftarrow \min(d[k], \lfloor q/v[k] \rfloor)$ ;
7  |  $conta \leftarrow 0$ ;
8  | Para  $r \leftarrow 0$  até  $rmax$  fazer
9  |    $conta \leftarrow conta + \text{CONTASOLS}(v, d, q - r * v[k], k - 1)$ ;
10 |  $N[q, k] \leftarrow conta$ ; /* memoriza para uso futuro se necessário */
11 | retorna  $conta$ ;
  
```

Implementação: Definir a tabela  $N$  por dicionário (hash-table)

Dicionários/Tabelas de dispersão/Arrays associativos – coleção de pares (Chave, Valor).

Java: *Map*, *HashMap*, *TreeMap* C++: *std::unordered\_map*, *std::map*

# Problemas de trocos com número de moedas limitado

## Abordagem "Top-Down" com memoização

```

CONTASOLS( $v, d, q, k$ )    /* chamar CONTASOLS( $v, d, Q, m$ ) para obter  $N_{Q,m}$  */
1  | Se  $q = 0$  então retorna 1;
2  | Se  $k = 1$  então
3  |   Se  $q \% v[1] \neq 0 \vee d[1] < q/v[1]$  então retorna 0;
4  |   retorna 1;
5  | Se  $N[q, k]$  já calculado então retorna  $N[q, k]$ ;
6  |  $rmax \leftarrow \min(d[k], \lfloor q/v[k] \rfloor)$ ;
7  |  $conta \leftarrow 0$ ;
8  | Para  $r \leftarrow 0$  até  $rmax$  fazer
9  |    $conta \leftarrow conta + \text{CONTASOLS}(v, d, q - r * v[k], k - 1)$ ;
10 |  $N[q, k] \leftarrow conta$ ; /* memoriza para uso futuro se necessário */
11 | retorna  $conta$ ;

```

**Implementação:** Definir a tabela  $N$  por **dicionário** (*hash-table*)

Dicionários/Tabelas de dispersão/Arrays associativos – coleção de pares (**Chave, Valor**).

Java: *Map*, *HashMap*, *TreeMap* C++: *std::unordered\_map*, *std::map*

# Problemas de trocos com número de moedas ilimitado



**Problema:** Supondo que se tem um número **não limitado** de moedas de valores **200, 100, 50, 20, 10, 5, 2, e 1**, qual é o **número mínimo** de moedas necessário para formar uma quantia  $Q$ ?

- Abordagem de programação dinâmica é **ineficiente**.
- Prova-se que a **estratégia greedy** que consiste em começar por **usar a moeda de valor mais alto**  $v_k \leq Q$  **o número máximo de vezes que puder** (isto é,  $n_k = \lfloor Q/v_k \rfloor$  vezes) e aplicar a mesma estratégia para obter a quantia  $Q - n_k v_k$  restante, determina a **solução ótima**, em  $O(m)$ , sendo  $m$  o número de tipos de moedas existentes.

**Atenção!** Para garantir  $O(m)$ , é necessário usar  $Q - n_k v_k$  em vez de dar **uma** moeda  $v_k$  e aplicar a estratégia a  $Q - v_k$ . Note que  $O(Q)$  é  $O(2^{\log_2 Q})$  e, portanto, é exponencial no tamanho da representação de  $Q$  (input) em binário

(assumido no modelo RAM para análise assintótica).

# Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se  $\{200, 100, 50, 20, 10, 5, 2, 1\}$ :

- Seja  $x^*$  uma solução ótima para a quantia  $Q$ . Seja  $x_v^*$  é o número de moedas que usa de valor  $v$ .
  - Se  $x_{100}^* > 1$ , a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto,  $x_{100}^* \leq 1$ .  
Analogamente se conclui que:  $x_{50}^* \leq 1$ ,  $x_{10}^* \leq 1$ , e  $x_1^* \leq 1$ .
  - Se  $x_{20}^* > 2$  então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto,  $x_{20}^* \leq 2$ . Analogamente,  $x_2^* \leq 2$ .
  - Não pode ter simultaneamente  $x_2^* = 2$  e  $x_1^* = 1$ , pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto  $2x_2^* + x_1^* \leq 4$ .  
Também não tem simultaneamente  $x_{20}^* = 2$  e  $x_{10}^* = 1$ .
  - Como  $2x_2^* + x_1^* \leq 4$ ,  $x_5^* \leq 1$  e  $x_{10}^* \leq 1$  então  $5x_5^* + 2x_2^* + x_1^* \leq 9$  e  $10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 19$ . Analogamente, se deduz que  $20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 49$ ,  $50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 99$ .  
 $100x_{100}^* + 50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 199$ .
- Tem-se  $\sum_{i=1}^k v_i x_{v_i}^* < v_{k+1}$ , para todo  $k$ . Portanto,  $x^*$  é a solução greedy. □

NB: A estratégia greedy apresentada não seria correta para, por exemplo,  $V = \{1, 300, 1000\}$ ,  $Q = 1200$ .



# Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se  $\{200, 100, 50, 20, 10, 5, 2, 1\}$ :

- Seja  $x^*$  uma solução ótima para a quantia  $Q$ . Seja  $x_v^*$  é o número de moedas que usa de valor  $v$ .
  - Se  $x_{100}^* > 1$ , a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto,  $x_{100}^* \leq 1$ .  
Analogamente se conclui que:  $x_{50}^* \leq 1$ ,  $x_{10}^* \leq 1$ , e  $x_1^* \leq 1$ .
  - Se  $x_{20}^* > 2$  então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto,  $x_{20}^* \leq 2$ . Analogamente,  $x_2^* \leq 2$ .
  - Não pode ter simultaneamente  $x_2^* = 2$  e  $x_1^* = 1$ , pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto  $2x_2^* + x_1^* \leq 4$ .  
Também não tem simultaneamente  $x_{20}^* = 2$  e  $x_{10}^* = 1$ .
  - Como  $2x_2^* + x_1^* \leq 4$ ,  $x_5^* \leq 1$  e  $x_{10}^* \leq 1$  então  $5x_5^* + 2x_2^* + x_1^* \leq 9$  e  $10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 19$ . Analogamente, se deduz que  $20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 49$ ,  $50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 99$ .  
 $100x_{100}^* + 50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 199$ .
- Tem-se  $\sum_{i=1}^k v_i x_{v_i}^* < v_{k+1}$ , para todo  $k$ . Portanto,  $x^*$  é a solução greedy. □

NB: A estratégia greedy apresentada não seria correta para, por exemplo,  $V = \{1, 300, 1000\}$ ,  $Q = 1200$ .

# Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se  $\{200, 100, 50, 20, 10, 5, 2, 1\}$ :

- Seja  $x^*$  uma solução ótima para a quantia  $Q$ . Seja  $x_v^*$  é o número de moedas que usa de valor  $v$ .
  - Se  $x_{100}^* > 1$ , a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto,  $x_{100}^* \leq 1$ .  
Analogamente se conclui que:  $x_{50}^* \leq 1$ ,  $x_{10}^* \leq 1$ , e  $x_1^* \leq 1$ .
  - Se  $x_{20}^* > 2$  então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto,  $x_{20}^* \leq 2$ . Analogamente,  $x_2^* \leq 2$ .
  - Não pode ter simultaneamente  $x_2^* = 2$  e  $x_1^* = 1$ , pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto  $2x_2^* + x_1^* \leq 4$ .  
Também não tem simultaneamente  $x_{20}^* = 2$  e  $x_{10}^* = 1$ .
  - Como  $2x_2^* + x_1^* \leq 4$ ,  $x_5^* \leq 1$  e  $x_{10}^* \leq 1$  então  $5x_5^* + 2x_2^* + x_1^* \leq 9$  e  $10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 19$ . Analogamente, se deduz que  $20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 49$ ,  $50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 99$ .  
 $100x_{100}^* + 50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 199$ .
- Tem-se  $\sum_{i=1}^k v_i x_{v_i}^* < v_{k+1}$ , para todo  $k$ . Portanto,  $x^*$  é a solução greedy.  $\square$

NB: A estratégia greedy apresentada não seria correta para, por exemplo,  $V = \{1, 300, 1000\}$ ,  $Q = 1200$ .

# Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se  $\{200, 100, 50, 20, 10, 5, 2, 1\}$ :

- Seja  $x^*$  uma solução ótima para a quantia  $Q$ . Seja  $x_v^*$  é o número de moedas que usa de valor  $v$ .
  - Se  $x_{100}^* > 1$ , a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto,  $x_{100}^* \leq 1$ .  
Analogamente se conclui que:  $x_{50}^* \leq 1$ ,  $x_{10}^* \leq 1$ , e  $x_1^* \leq 1$ .
  - Se  $x_{20}^* > 2$  então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto,  $x_{20}^* \leq 2$ . Analogamente,  $x_2^* \leq 2$ .
  - Não pode ter simultaneamente  $x_2^* = 2$  e  $x_1^* = 1$ , pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto  $2x_2^* + x_1^* \leq 4$ .  
Também não tem simultaneamente  $x_{20}^* = 2$  e  $x_{10}^* = 1$ .
  - Como  $2x_2^* + x_1^* \leq 4$ ,  $x_5^* \leq 1$  e  $x_{10}^* \leq 1$  então  $5x_5^* + 2x_2^* + x_1^* \leq 9$  e  $10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 19$ . Analogamente, se deduz que  $20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 49$ ,  $50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 99$ .  
 $100x_{100}^* + 50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 199$ .
- Tem-se  $\sum_{i=1}^k v_i x_{v_i}^* < v_{k+1}$ , para todo  $k$ . Portanto,  $x^*$  é a solução greedy.  $\square$

NB: A estratégia greedy apresentada não seria correta para, por exemplo,  $V = \{1, 300, 1000\}$ ,  $Q = 1200$ .

# Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se  $\{200, 100, 50, 20, 10, 5, 2, 1\}$ :

- Seja  $x^*$  uma solução ótima para a quantia  $Q$ . Seja  $x_v^*$  é o número de moedas que usa de valor  $v$ .
- Se  $x_{100}^* > 1$ , a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto,  $x_{100}^* \leq 1$ .  
Analogamente se conclui que:  $x_{50}^* \leq 1$ ,  $x_{10}^* \leq 1$ , e  $x_1^* \leq 1$ .
- Se  $x_{20}^* > 2$  então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto,  $x_{20}^* \leq 2$ . Analogamente,  $x_2^* \leq 2$ .
- Não pode ter simultaneamente  $x_2^* = 2$  e  $x_1^* = 1$ , pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto  $2x_2^* + x_1^* \leq 4$ .  
Também não tem simultaneamente  $x_{20}^* = 2$  e  $x_{10}^* = 1$ .
- Como  $2x_2^* + x_1^* \leq 4$ ,  $x_5^* \leq 1$  e  $x_{10}^* \leq 1$  então  $5x_5^* + 2x_2^* + x_1^* \leq 9$  e  $10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 19$ . Analogamente, se deduz que  $20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 49$ ,  $50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 99$ .  
 $100x_{100}^* + 50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 199$ .

Tem-se  $\sum_{i=1}^k v_i x_{v_i}^* < v_{k+1}$ , para todo  $k$ . Portanto,  $x^*$  é a solução greedy. □

NB: A estratégia greedy apresentada não seria correta para, por exemplo,  $V = \{1, 300, 1000\}$ ,  $Q = 1200$ . ⏪ ⏩ ⏴ ⏵ 🔍 ↻

# Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se  $\{200, 100, 50, 20, 10, 5, 2, 1\}$ :

- Seja  $x^*$  uma solução ótima para a quantia  $Q$ . Seja  $x_v^*$  é o número de moedas que usa de valor  $v$ .
  - Se  $x_{100}^* > 1$ , a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto,  $x_{100}^* \leq 1$ .  
Analogamente se conclui que:  $x_{50}^* \leq 1$ ,  $x_{10}^* \leq 1$ , e  $x_1^* \leq 1$ .
  - Se  $x_{20}^* > 2$  então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto,  $x_{20}^* \leq 2$ . Analogamente,  $x_2^* \leq 2$ .
  - Não pode ter simultaneamente  $x_2^* = 2$  e  $x_1^* = 1$ , pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto  $2x_2^* + x_1^* \leq 4$ .  
Também não tem simultaneamente  $x_{20}^* = 2$  e  $x_{10}^* = 1$ .
  - Como  $2x_2^* + x_1^* \leq 4$ ,  $x_5^* \leq 1$  e  $x_{10}^* \leq 1$  então  $5x_5^* + 2x_2^* + x_1^* \leq 9$  e  $10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 19$ . Analogamente, se deduz que  $20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 49$ ,  $50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 99$ .  
 $100x_{100}^* + 50x_{50}^* + 20x_{20}^* + 10x_{10}^* + 5x_5^* + 2x_2^* + x_1^* \leq 199$ .
- Tem-se  $\sum_{i=1}^k v_i x_{v_i}^* < v_{k+1}$ , para todo  $k$ . Portanto,  $x^*$  é a solução greedy. □

NB: A estratégia greedy apresentada não seria correta para, por exemplo,  $V = \{1, 300, 1000\}$ ,  $Q = 1200$ .

# Algoritmo de Bellman-Ford

Seja  $G = (V, E, d)$  um grafo que **pode ter pesos negativos** nos ramos. O **algoritmo de Bellman-Ford** determina percursos com peso mínimo de um nó **origem**  $s$  para cada nó  $v \in V = \{1, 2, \dots, n\}$ .

- Um percurso mínimo de  $s$  para  $v$  não pode ter mais do que  $n - 1$  ramos, a menos que inclua **ciclos com peso negativo**. Mas, nesse caso, **não há** percurso mínimo de  $s$  para  $v$  pois, usando o ciclo, poderia diminuir o peso. . .
- O algoritmo inclui um passo (linhas 6–8) para **verificar se existem ciclos com peso negativo**. Se existirem, as distâncias finais não estariam corretas.

## ALGORITMO BELLMAN-FORD( $s, n$ )

1. Para cada  $v \in V$  fazer  $dist[v] \leftarrow \infty$
2.  $dist[s] \leftarrow 0$ ;
3. Para  $r \leftarrow 1$  até  $n - 1$  fazer
4.     Para cada  $(u, v) \in E$  fazer
5.         Se  $dist[v] > dist[u] + d(u, v)$  então  $dist[v] \leftarrow dist[u] + d(u, v)$
6. Para cada  $(u, v) \in E$  fazer
7.     Se  $dist[v] > dist[u] + d(u, v)$  então retorna false; /\*ciclos com peso negativo\*/
8. retorna true; /\*sem ciclos com peso negativo\*/

# Algoritmo de Bellman-Ford

Seja  $G = (V, E, d)$  um grafo que **pode ter pesos negativos** nos ramos. O algoritmo de Bellman-Ford determina percursos com peso mínimo de um nó **origem**  $s$  para cada nó  $v \in V = \{1, 2, \dots, n\}$ .

- Um percurso mínimo de  $s$  para  $v$  não pode ter mais do que  $n - 1$  ramos, a menos que inclua **ciclos com peso negativo**. Mas, nesse caso, **não há** percurso mínimo de  $s$  para  $v$  pois, usando o ciclo, poderia diminuir o peso...
- O algoritmo inclui um passo (linhas 6–8) para **verificar se existem ciclos com peso negativo**. Se existirem, as distâncias finais não estariam corretas.

## ALGORITMO BELLMAN-FORD( $s, n$ )

- Para cada  $v \in V$  fazer  $dist[v] \leftarrow \infty$
- $dist[s] \leftarrow 0$ ;
- Para  $r \leftarrow 1$  até  $n - 1$  fazer
- Para cada  $(u, v) \in E$  fazer
- Se  $dist[v] > dist[u] + d(u, v)$  então  $dist[v] \leftarrow dist[u] + d(u, v)$
- Para cada  $(u, v) \in E$  fazer
- Se  $dist[v] > dist[u] + d(u, v)$  então retorna false; /\*ciclos com peso negativo\*/
- retorna true; /\*sem ciclos com peso negativo\*/

# Algoritmo de Bellman-Ford

Seja  $G = (V, E, d)$  um grafo que **pode ter pesos negativos** nos ramos. O algoritmo de Bellman-Ford determina percursos com peso mínimo de um nó **origem**  $s$  para cada nó  $v \in V = \{1, 2, \dots, n\}$ .

- Um percurso mínimo de  $s$  para  $v$  não pode ter mais do que  $n - 1$  ramos, a menos que inclua **ciclos com peso negativo**. Mas, nesse caso, **não há** percurso mínimo de  $s$  para  $v$  pois, usando o ciclo, poderia diminuir o peso. . .
- O algoritmo inclui um passo (linhas 6–8) para **verificar se existem ciclos com peso negativo**. Se existirem, as distâncias finais não estariam corretas.

## ALGORITMO BELLMAN-FORD( $s, n$ )

- Para cada  $v \in V$  fazer  $dist[v] \leftarrow \infty$
- $dist[s] \leftarrow 0$ ;
- Para  $r \leftarrow 1$  até  $n - 1$  fazer
  - Para cada  $(u, v) \in E$  fazer
    - Se  $dist[v] > dist[u] + d(u, v)$  então  $dist[v] \leftarrow dist[u] + d(u, v)$
- Para cada  $(u, v) \in E$  fazer
  - Se  $dist[v] > dist[u] + d(u, v)$  então retorna false; /\*ciclos com peso negativo\*/
- retorna true; /\*sem ciclos com peso negativo\*/



# Algoritmo de Bellman-Ford

Seja  $G = (V, E, d)$  um grafo que **pode ter pesos negativos** nos ramos. O **algoritmo de Bellman-Ford** determina percursos com peso mínimo de um nó **origem**  $s$  para cada nó  $v \in V = \{1, 2, \dots, n\}$ .

- Um percurso mínimo de  $s$  para  $v$  não pode ter mais do que  $n - 1$  ramos, a menos que inclua **ciclos com peso negativo**. Mas, nesse caso, **não há** percurso mínimo de  $s$  para  $v$  pois, usando o ciclo, poderia diminuir o peso. . .
- O algoritmo inclui um passo (linhas 6–8) para **verificar se existem ciclos com peso negativo**. Se existirem, as distâncias finais não estariam corretas.

## ALGORITMO BELLMAN-FORD( $s, n$ )

- Para cada  $v \in V$  fazer  $dist[v] \leftarrow \infty$
- $dist[s] \leftarrow 0$ ;
- Para  $r \leftarrow 1$  até  $n - 1$  fazer
  - Para cada  $(u, v) \in E$  fazer
    - Se  $dist[v] > dist[u] + d(u, v)$  então  $dist[v] \leftarrow dist[u] + d(u, v)$
- Para cada  $(u, v) \in E$  fazer
  - Se  $dist[v] > dist[u] + d(u, v)$  então retorna false; /\*ciclos com peso negativo\*/
- retorna true; /\*sem ciclos com peso negativo\*/

# Algoritmo de Bellman-Ford

Seja  $G = (V, E, d)$  um grafo que **pode ter pesos negativos** nos ramos. O algoritmo de Bellman-Ford determina percursos com peso mínimo de um nó **origem**  $s$  para cada nó  $v \in V = \{1, 2, \dots, n\}$ .

- Um percurso mínimo de  $s$  para  $v$  não pode ter mais do que  $n - 1$  ramos, a menos que inclua **ciclos com peso negativo**. Mas, nesse caso, **não há** percurso mínimo de  $s$  para  $v$  pois, usando o ciclo, poderia diminuir o peso. . .
- O algoritmo inclui um passo (linhas 6–8) para **verificar se existem ciclos com peso negativo**. Se existirem, as distâncias finais não estariam corretas.

## ALGORITMO BELLMAN-FORD( $s, n$ )

1. Para cada  $v \in V$  fazer  $dist[v] \leftarrow \infty$
2.  $dist[s] \leftarrow 0$ ;
3. Para  $r \leftarrow 1$  até  $n - 1$  fazer
4.     Para cada  $(u, v) \in E$  fazer
5.         Se  $dist[v] > dist[u] + d(u, v)$  então  $dist[v] \leftarrow dist[u] + d(u, v)$
6. Para cada  $(u, v) \in E$  fazer
7.     Se  $dist[v] > dist[u] + d(u, v)$  então retorna false; /\*ciclos com peso negativo\*/
8. retorna true; /\*sem ciclos com peso negativo\*/

# Algoritmo de Bellman-Ford adaptado para todos os pares

Para  $d(e) \in \mathbb{R}^+$ , a matriz das distâncias mínimas  $\tilde{D}_{ij}^{(n-1)}$  para **todos os**  $(i, j)$  pode ser definida pela recorrência

$$\tilde{D}_{ij}^{(1)} = \begin{cases} d(i, j) & \text{se } i \neq j \text{ e } (i, j) \in E \\ \infty, & \text{se } i \neq j \text{ e } (i, j) \notin E \\ 0, & \text{se } i = j. \end{cases}$$

$$\begin{aligned} \tilde{D}_{ij}^{(r)} &= \min\{\tilde{D}_{ik}^{(r-1)} + \tilde{D}_{kj}^{(1)} \mid 1 \leq k \leq n\} \\ &= \min\{\tilde{D}_{ik}^{(1)} + \tilde{D}_{kj}^{(r-1)} \mid 1 \leq k \leq n\}, \quad \text{se } r \geq 2 \end{aligned}$$

$\tilde{D}_{ij}^{(r)}$  é a distância mínima de  $i$  para  $j$  se o percurso não puder ter mais do que  $r$  ramos, para cada  $r \geq 1$ .

# Algoritmo de Bellman-Ford adaptado para todos os pares

Para  $d(e) \in \mathbb{R}^+$ , a matriz das distâncias mínimas  $\tilde{D}_{ij}^{(n-1)}$  para **todos os**  $(i, j)$  pode ser definida pela recorrência

$$\begin{aligned}\tilde{D}_{ij}^{(1)} &= \begin{cases} d(i, j) & \text{se } i \neq j \text{ e } (i, j) \in E \\ \infty, & \text{se } i \neq j \text{ e } (i, j) \notin E \\ 0, & \text{se } i = j. \end{cases} \\ \tilde{D}_{ij}^{(r)} &= \min\{\tilde{D}_{ik}^{(r-1)} + \tilde{D}_{kj}^{(1)} \mid 1 \leq k \leq n\} \\ &= \min\{\tilde{D}_{ik}^{(1)} + \tilde{D}_{kj}^{(r-1)} \mid 1 \leq k \leq n\}, \text{ se } r \geq 2\end{aligned}$$

Calcula-se como um **produto de matrizes**  $\otimes$  em  $(\mathbb{R}, \min, +)$ , sendo um **método multiplicativo**. Os percursos mínimos têm subestrutura ótima e, por isso,  $\tilde{D}^{(r+s)} = \tilde{D}^{(r)} \otimes \tilde{D}^{(s)}$ , sendo dada por

$$(\tilde{D}^{(r)} \otimes \tilde{D}^{(s)})_{ij} = \min_{1 \leq k \leq n} (\tilde{D}_{ik}^{(r)} + \tilde{D}_{kj}^{(s)})$$

Para reduzir o número de multiplicações  $\otimes$  de  $\Theta(n)$  para  $\Theta(\log_2 n)$  podemos usar o **método binário para cálculo de potências**:  $x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n \% 2} = \prod_{t=0}^{\lfloor \log_2 n \rfloor} (x^{2^t})^{b_t}$ , onde  $b_t$  é o bit  $t$  da representação de  $n$  em binário. Por exemplo,  $\tilde{D}^{19} = \tilde{D}^{16} \otimes \tilde{D}^2 \otimes \tilde{D}$ .

# Algoritmo de Bellman-Ford adaptado para todos os pares

Para  $d(e) \in \mathbb{R}^+$ , a matriz das distâncias mínimas  $\tilde{D}_{ij}^{(n-1)}$  para **todos os**  $(i, j)$  pode ser definida pela recorrência

$$\begin{aligned}\tilde{D}_{ij}^{(1)} &= \begin{cases} d(i, j) & \text{se } i \neq j \text{ e } (i, j) \in E \\ \infty, & \text{se } i \neq j \text{ e } (i, j) \notin E \\ 0, & \text{se } i = j. \end{cases} \\ \tilde{D}_{ij}^{(r)} &= \min\{\tilde{D}_{ik}^{(r-1)} + \tilde{D}_{kj}^{(1)} \mid 1 \leq k \leq n\} \\ &= \min\{\tilde{D}_{ik}^{(1)} + \tilde{D}_{kj}^{(r-1)} \mid 1 \leq k \leq n\}, \text{ se } r \geq 2\end{aligned}$$

Calcula-se como um **produto de matrizes**  $\otimes$  em  $(\mathbb{R}, \min, +)$ , sendo um **método multiplicativo**. Os **percursos mínimos** têm subestrutura ótima e, por isso,  $\tilde{D}^{(r+s)} = \tilde{D}^{(r)} \otimes \tilde{D}^{(s)}$ , sendo dada por

$$(\tilde{D}^{(r)} \otimes \tilde{D}^{(s)})_{ij} = \min_{1 \leq k \leq n} (\tilde{D}_{ik}^{(r)} + \tilde{D}_{kj}^{(s)})$$

Para reduzir o número de multiplicações  $\otimes$  de  $\Theta(n)$  para  $\Theta(\log_2 n)$  podemos usar o **método binário para cálculo de potências**:  $x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n \% 2} = \prod_{t=0}^{\lfloor \log_2 n \rfloor} (x^{2^t})^{b_t}$ , onde  $b_t$  é o bit  $t$  da representação de  $n$  em binário. Por exemplo,  $\tilde{D}^{19} = \tilde{D}^{16} \otimes \tilde{D}^2 \otimes \tilde{D}$ .

# Algoritmo de Bellman-Ford adaptado para todos os pares

Para  $d(e) \in \mathbb{R}^+$ , a matriz das distâncias mínimas  $\tilde{D}_{ij}^{(n-1)}$  para **todos os**  $(i, j)$  pode ser definida pela recorrência

$$\tilde{D}_{ij}^{(1)} = \begin{cases} d(i, j) & \text{se } i \neq j \text{ e } (i, j) \in E \\ \infty, & \text{se } i \neq j \text{ e } (i, j) \notin E \\ 0, & \text{se } i = j. \end{cases}$$

$$\tilde{D}_{ij}^{(r)} = \min\{\tilde{D}_{ik}^{(r-1)} + \tilde{D}_{kj}^{(1)} \mid 1 \leq k \leq n\}$$

$$= \min\{\tilde{D}_{ik}^{(1)} + \tilde{D}_{kj}^{(r-1)} \mid 1 \leq k \leq n\}, \text{ se } r \geq 2$$

Calcula-se como um **produto de matrizes**  $\otimes$  em  $(\mathbb{R}, \min, +)$ , sendo um **método multiplicativo**. Os **percursos mínimos têm subestrutura ótima** e, por isso,  $\tilde{D}^{(r+s)} = \tilde{D}^{(r)} \otimes \tilde{D}^{(s)}$ , sendo dada por

$$(\tilde{D}^{(r)} \otimes \tilde{D}^{(s)})_{ij} = \min_{1 \leq k \leq n} (\tilde{D}_{ik}^{(r)} + \tilde{D}_{kj}^{(s)})$$

Para reduzir o número de multiplicações  $\otimes$  de  $\Theta(n)$  para  $\Theta(\log_2 n)$  podemos usar o **método binário para cálculo de potências**:  $x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n \% 2} = \prod_{t=0}^{\lfloor \log_2 n \rfloor} (x^{2^t})^{b_t}$ , onde  $b_t$  é o bit  $t$  da representação de  $n$  em binário. Por exemplo,  $\tilde{D}^{19} = \tilde{D}^{16} \otimes \tilde{D}^2 \otimes \tilde{D}$ .

# Outra abordagem DP para cálculo do fecho transitivo

Podemos adotar a ideia do algoritmo de Bellman-Ford (adaptado) para obter a matrix do fecho transitivo  $R^+$ .

- Definimos  $\tilde{M}_{ij}^{(r)} = 1$  se existir algum percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  com até  $r$  ramos, para  $r \geq 1$ , fixo.
- Recorrência: para todos os pares  $(i, j)$  tem-se

$$\tilde{M}_{ij}^{(1)} = M_{ij}$$

$$\tilde{M}_{ij}^{(r)} = \tilde{M}_{ij}^{(r-1)} \vee \left( \bigvee_{k=1}^n (\tilde{M}_{ik}^{(r-1)} \wedge M_{kj}) \right), \quad \text{para } r \geq 2$$

- Podemos avaliar usando o método multiplicativo e adaptar o método binário para reduzir o número de multiplicações, pois

$$\tilde{M}^{(r+s)} = \tilde{M}^{(r)} \otimes \tilde{M}^{(s)}$$

onde o produto de matrizes  $\otimes$  é considerado em  $(\{0, 1\}, \vee, \wedge)$ .

- Pontos fixos:** se  $\tilde{M}^{(r)} = \tilde{M}^{(r+1)}$ , então  $\tilde{M}^{(r)} = M^+$ . Também, no algoritmo de Bellman-Ford adaptado, se  $\tilde{D}^{(r)} = \tilde{D}^{(r+1)}$ , então  $\tilde{D}^{(r)} = \tilde{D}^{(n)}$ .

# Outra abordagem DP para cálculo do fecho transitivo

Podemos adotar a ideia do algoritmo de Bellman-Ford (adaptado) para obter a matrix do fecho transitivo  $R^+$ .

- Definimos  $\tilde{M}_{ij}^{(r)} = 1$  se existir algum **percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  com até  $r$  ramos**, para  $r \geq 1$ , fixo.
- Recorrência: para todos os pares  $(i, j)$  tem-se

$$\tilde{M}_{ij}^{(1)} = M_{ij}$$

$$\tilde{M}_{ij}^{(r)} = \tilde{M}_{ij}^{(r-1)} \vee \left( \bigvee_{k=1}^n (\tilde{M}_{ik}^{(r-1)} \wedge M_{kj}) \right), \quad \text{para } r \geq 2$$

- Podemos avaliar usando o método multiplicativo e adaptar o método binário para reduzir o número de multiplicações, pois

$$\tilde{M}^{(r+s)} = \tilde{M}^{(r)} \otimes \tilde{M}^{(s)}$$

onde o produto de matrizes  $\otimes$  é considerado em  $(\{0, 1\}, \vee, \wedge)$ .

- Pontos fixos:** se  $\tilde{M}^{(r)} = \tilde{M}^{(r+1)}$ , então  $\tilde{M}^{(r)} = M^+$ . Também, no algoritmo de Bellman-Ford adaptado, se  $\tilde{D}^{(r)} = \tilde{D}^{(r+1)}$ , então  $\tilde{D}^{(r)} = \tilde{D}^{(n)}$ .



# Outra abordagem DP para cálculo do fecho transitivo

Podemos adotar a ideia do algoritmo de Bellman-Ford (adaptado) para obter a matrix do fecho transitivo  $R^+$ .

- Definimos  $\tilde{M}_{ij}^{(r)} = 1$  se existir algum **percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  com até  $r$  ramos**, para  $r \geq 1$ , fixo.
- Recorrência: para todos os pares  $(i, j)$  tem-se

$$\tilde{M}_{ij}^{(1)} = M_{ij}$$

$$\tilde{M}_{ij}^{(r)} = \tilde{M}_{ij}^{(r-1)} \vee \left( \bigvee_{k=1}^n (\tilde{M}_{ik}^{(r-1)} \wedge M_{kj}) \right), \quad \text{para } r \geq 2$$

- Podemos avaliar usando o método multiplicativo e adaptar o método binário para reduzir o número de multiplicações, pois

$$\tilde{M}^{(r+s)} = \tilde{M}^{(r)} \otimes \tilde{M}^{(s)}$$

onde o produto de matrizes  $\otimes$  é considerado em  $(\{0, 1\}, \vee, \wedge)$ .

- Pontos fixos:** se  $\tilde{M}^{(r)} = \tilde{M}^{(r+1)}$ , então  $\tilde{M}^{(r)} = M^+$ . Também, no algoritmo de Bellman-Ford adaptado, se  $\tilde{D}^{(r)} = \tilde{D}^{(r+1)}$ , então  $\tilde{D}^{(r)} = \tilde{D}^{(n)}$ .

## Outra abordagem DP para cálculo do fecho transitivo

Podemos adotar a ideia do algoritmo de Bellman-Ford (adaptado) para obter a matrix do fecho transitivo  $R^+$ .

- Definimos  $\tilde{M}_{ij}^{(r)} = 1$  se existir algum **percurso no grafo de  $R$  do nó  $i$  para o nó  $j$  com até  $r$  ramos**, para  $r \geq 1$ , fixo.
- Recorrência: para todos os pares  $(i, j)$  tem-se

$$\tilde{M}_{ij}^{(1)} = M_{ij}$$

$$\tilde{M}_{ij}^{(r)} = \tilde{M}_{ij}^{(r-1)} \vee \left( \bigvee_{k=1}^n (\tilde{M}_{ik}^{(r-1)} \wedge M_{kj}) \right), \quad \text{para } r \geq 2$$

- Podemos avaliar usando o método multiplicativo e adaptar o método binário para reduzir o número de multiplicações, pois

$$\tilde{M}^{(r+s)} = \tilde{M}^{(r)} \otimes \tilde{M}^{(s)}$$

onde o produto de matrizes  $\otimes$  é considerado em  $(\{0, 1\}, \vee, \wedge)$ .

- Pontos fixos:** se  $\tilde{M}^{(r)} = \tilde{M}^{(r+1)}$ , então  $\tilde{M}^{(r)} = M^+$ . Também, no algoritmo de Bellman-Ford adaptado, se  $\tilde{D}^{(r)} = \tilde{D}^{(r+1)}$ , então  $\tilde{D}^{(r)} = \tilde{D}^{(n)}$ .

# Algoritmo CYK para decidir se $x \in \mathcal{L}(G)$ , para GIC $G$ na forma normal de Chomsky, e $x \in \Sigma^*$

- Para  $G = (V, \Sigma, P, S)$  fixa, a complexidade temporal do algoritmo é  $O(|x|^3)$ , ou seja, é cúbica no comprimento da palavra que se quer analisar.
- Seja  $N[i, i + s]$  o conjunto de variáveis em  $V$  que geram a subpalavra  $x_i \dots x_{i+s}$  de  $x$ , isto é,  $N[i, i + s] = \{A \mid A \in V, A \Rightarrow_G^* x_i \dots x_{i+s}\}$ .
- **Algoritmo CYK:**
  - $N[i, i] := \{A \mid A \in V, A \rightarrow x_i\}$ , para  $1 \leq i \leq n$  e  $N[i, j] := \emptyset$ , para todo  $(i, j)$ , com  $i \neq j$ .
  - Para cada  $s$  entre 1 e  $n - 1$  fazer
    - Para cada  $i$  entre 1 e  $n - s$ , considerar  $N[i, k]$  e  $N[k + 1, i + s]$ , para todo  $k$  com  $i \leq k \leq (i + s) - 1$ . Se existir  $(A \rightarrow BC) \in P$  com  $B \in N[i, k]$  e  $C \in N[k + 1, i + s]$ , acrescentar  $A$  a  $N[i, i + s]$ .
  - A palavra  $x$  está em  $\mathcal{L}(G)$  se e só se  $S \in N[1, n]$ .

# Algoritmo CYK – aplicação de DP

É usual usar uma matriz, com  $N[t, t + s]$  na coluna  $t$  e linha  $\#s+1$ .

#n	$N[1, n]$					
#n-1	$N[1, n-1]$	$N[2, n]$				
⋮	⋮	⋮	⋮			
#3	$N[1, 3]$	$N[2, 4]$	⋯	$N[n-2, n]$		
#2	$N[1, 2]$	$N[2, 3]$	⋯	$N[n-2, n-1]$	$N[n-1, n]$	
#1	$N[1, 1]$	$N[2, 2]$	⋯	$N[n-2, n-2]$	$N[n-1, n-1]$	$N[n, n]$
	$x_1$	$x_2$	⋯	$x_{n-2}$	$x_{n-1}$	$x_n$

A entrada  $N[t, t + s]$  da tabela apresenta o conjunto das categorias possíveis para a subpalavra  $x_t \cdots x_{t+s}$  de  $x$ . Portanto, caracteriza as categorias das subpalavras indicadas na matrix seguinte:

#n	$x_1 \cdots x_n$					
#n-1	$x_1 \cdots x_{n-1}$	$x_2 \cdots x_n$				
⋮	⋮	⋮	⋮			
#3	$x_1 x_2 x_3$	$x_2 x_3 x_4$	⋯	$x_{n-2} x_{n-1} x_n$		
#2	$x_1 x_2$	$x_2 x_3$	⋯	$x_{n-2} x_{n-1}$	$x_{n-1} x_n$	
#1	$x_1$	$x_2$	⋯	$x_{n-2}$	$x_{n-1}$	$x_n$

# Algoritmo CYK – aplicação de DP

É usual usar uma matriz, com  $N[t, t + s]$  na coluna  $t$  e linha  $\#s+1$ .

#n	$N[1, n]$					
#n-1	$N[1, n-1]$	$N[2, n]$				
⋮	⋮	⋮	⋮			
#3	$N[1, 3]$	$N[2, 4]$	⋯	$N[n-2, n]$		
#2	$N[1, 2]$	$N[2, 3]$	⋯	$N[n-2, n-1]$	$N[n-1, n]$	
#1	$N[1, 1]$	$N[2, 2]$	⋯	$N[n-2, n-2]$	$N[n-1, n-1]$	$N[n, n]$
	$x_1$	$x_2$	⋯	$x_{n-2}$	$x_{n-1}$	$x_n$

A entrada  $N[t, t + s]$  da tabela apresenta o conjunto das categorias possíveis para a subpalavra  $x_t \cdots x_{t+s}$  de  $x$ . Portanto, caracteriza as categorias das subpalavras indicadas na matrix seguinte:

#n	$x_1 \cdots x_n$					
#n-1	$x_1 \cdots x_{n-1}$	$x_2 \cdots x_n$				
⋮	⋮	⋮	⋮			
#3	$x_1 x_2 x_3$	$x_2 x_3 x_4$	⋯	$x_{n-2} x_{n-1} x_n$		
#2	$x_1 x_2$	$x_2 x_3$	⋯	$x_{n-2} x_{n-1}$	$x_{n-1} x_n$	
#1	$x_1$	$x_2$	⋯	$x_{n-2}$	$x_{n-1}$	$x_n$

# Algoritmo CYK - Exemplo

Para GIC  $G$ , com  $V = \{E, T, F, E_1, E_2, T_1, T_2, T_3, M, S, X, Q, A, B\}$ , símbolo inicial  $E$ , e seguintes produções:

$$E \rightarrow TE_1 \mid TE_2 \mid FT_1 \mid FT_2 \mid AT_3 \mid n$$

$$T \rightarrow n \mid FT_1 \mid FT_2 \mid AT_3$$

$$F \rightarrow AT_3 \mid n$$

$$E_1 \rightarrow ME$$

$$E_2 \rightarrow SE$$

$$T_1 \rightarrow XT$$

$$T_2 \rightarrow QT$$

$$M \rightarrow +$$

$$S \rightarrow -$$

$$X \rightarrow *$$

$$Q \rightarrow /$$

$$A \rightarrow ($$

$$B \rightarrow )$$

$$T_3 \rightarrow EB$$

Tem-se  $(n + n) * n \in \mathcal{L}(G)$ ? Sim, porque o símbolo inicial  $E$  ocorre no topo da tabela:

#7	{T, E}						
#6	∅	∅					
#5	{F, T, E}	∅	∅				
#4	∅	{T <sub>3</sub> }	∅	∅			
#3	∅	{E}	∅	∅	∅		
#2	∅	∅	{E <sub>1</sub> }	{T <sub>3</sub> }	∅	{T <sub>1</sub> }	
#1	{A}	{E, T, F}	{M}	{E, T, F}	{B}	{X}	{E, T, F}
	(	n	+	n	)	*	n