

Aus 16 & 17

## Algoritmos Greedy - O Problema da Mochila Fracionária

Input: • Mochila com capacidade  $\underline{W}$

- $n$  itens com pesos  $w_1, \dots, w_n$  e valores  $v_1, \dots, v_n$

Objetivo: Quais os itens que devemos colocar na mochila  
se quisermos transportar o maior valor possível, sendo  
que podemos transportar uma quantidade fracionária  
de cada item?

$$\begin{aligned} & \max \sum_{i=0}^n u_i \cdot \left( \frac{v_i}{w_i} \right) \\ & \forall i. 0 \leq x_i \leq 1 \\ & \sum_{i=0}^n u_i \leq \underline{W} \end{aligned} \quad \left. \right\} \quad \begin{aligned} & \text{O problema da mochila fracionária como} \\ & \text{um } \underline{\text{programa linear}} \end{aligned}$$

## Algoritmos Greedy - O Problema da Mochila Fracionária

Input: • Mochila com capacidade  $W$

- $n$  itens com pesos  $w_1, \dots, w_n$  e valores  $v_1, \dots, v_n$

Escolha Greedy: Coloca na mochila a maior quantidade possível do item com maior valor por unidade de peso

- Para cada item  $i$ , calcula  $v_i/w_i$
- Determinar o item  $k$  com maior valor  $v_k/w_k$

$$(x_k, W') = \begin{cases} \left(\frac{w_k}{W}, 0\right) & \text{se } W \leq w_k \\ (1, V - w_k) & \text{c.c.} \end{cases}$$

# Algoritmos Greedy - O Problema da Mochila Fracionária

FractionalKnapsack ( $w, v, \underline{w}$ )

let  $x[1..n]$  be a new array initialized to 0

for  $i = 1$  to  $n$

| if ( $w[i] < \underline{w}$ )

| |  $x[i] = 1;$

| |  $\underline{w} = \underline{w} - v[i]$

| else if  $w[i] \geq \underline{w}$

| |  $x[i] = w[i]/\underline{w};$

| | Return  $x$

Return  $x$

Observação:

Os produtos estão inicialmente ordenados  
por  $v_i/w_i$

$\Theta(n)$   $\Rightarrow$  os elementos já ordenados

$O(n \log n)$   $\Rightarrow$  os elementos não ordenados

## Algoritmos Greedy - O Problema da Mochila Fracionária

FractionalKnapsack ( $w, v, \underline{w}$ )

let  $x[1..n]$  be a new array initialized to 0

for  $i = 1$  to  $n$

| if ( $w[i] < \underline{w}$ )

| |  $x[i] = 1;$

| |  $\underline{w} = \underline{w} - v[i]$

| else //  $w[i] \geq \underline{w}$

| |  $x[i] = w[i]/\underline{w};$

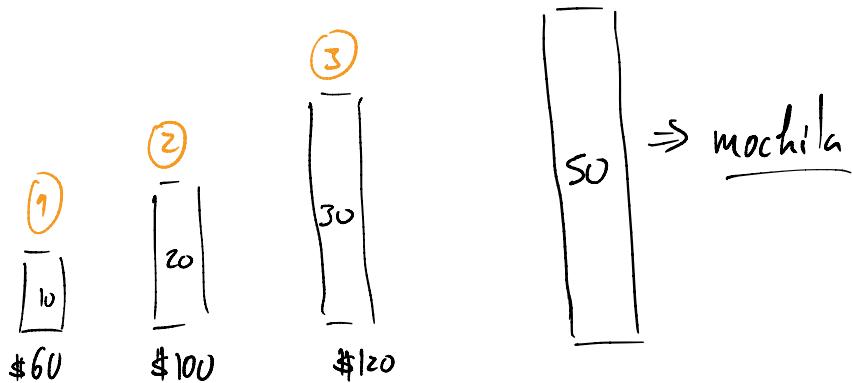
| | return  $x$

return  $x$

Complexidade:

$O(n)$

## Algoritmos Greedy - O Problema da Mochila Fracionária



$$\frac{60}{10} = 6 \quad \frac{100}{20} = 5 \quad \frac{120}{30} = 4$$

$$\cdot x_1 = 1, \quad W = 50 - 10 = 40$$

$$\cdot C = 1 \times 60 + 1 \times 100 + \frac{2}{3} \times 120$$

$$\cdot x_2 = 1, \quad W = 40 - 20 = 20$$

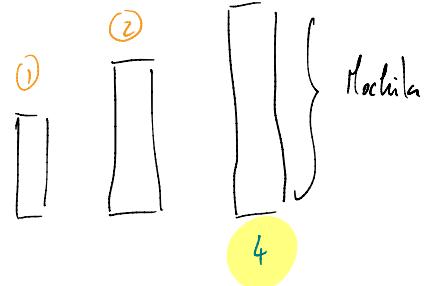
$$= 160 + 2 \times 40 = 160 + 80 = 240$$

$$\cdot x_3 = \frac{20}{30} = \frac{2}{3}, \quad W = 0$$

—

## Problema da Maçula Não Fracionária

- Não é permitido transportar quantidades fracionárias!



- Contar - exemplo:

$$\textcircled{1} \quad w_1 = 1$$

Escolha livre: Escolhamos o item ① mas depois o item ② já não cabe.

$$v_1 = 3$$

$$\textcircled{2} \quad w_2 = 4$$

$$v_2 = 4$$

## Problema da Mochila Não Fracionária

### • Problema da Mochila

- Com Repetição  $\Rightarrow$  Disponemos de quantidades ilimitadas de cada item
- Sem Repetição  $\Rightarrow$  Disponemos de uma unidade de cada item.

### • Problema da Mochila com Repetição

$$\vec{w} = [w_1, \dots, w_n]$$

$$\vec{v} = [v_1, \dots, v_n]$$

$k(w) \Rightarrow$  maior valor que conseguimos transformar numa mochila com capacidade  $w$

$$k(w) = \max \left\{ v_i + k(w - w_i) \mid w_i \leq w \right\}$$

$$k(w) = v_i + k(w - w_i)$$

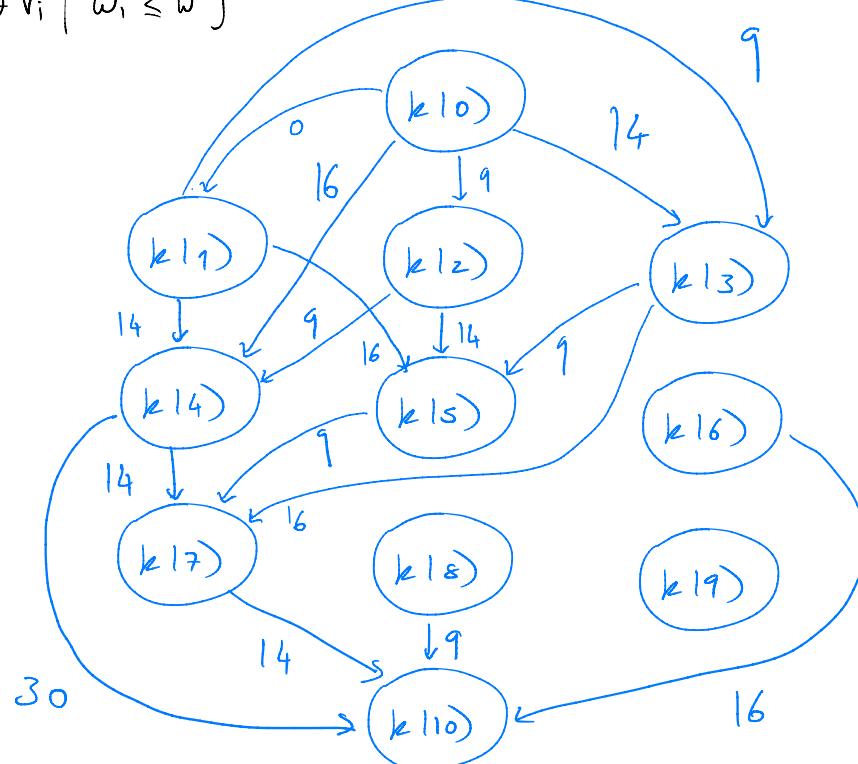
# Problema da Mochila Não Fracionária

- Problema da Mochila com Repetição

$k(w) \Rightarrow$  maior valor q conseguimos transportar numa mochila com capacidade  $w$

$$k(w) = \max \left\{ k(w - w_i) + v_i \mid w_i \leq w \right\}$$

Item	Peso	Valor	$w=10$
1	6	\$30	
2	3	\$14	
3	4	\$16	
4	2	\$9	



(incompleto)

- Precisamos encontrar...

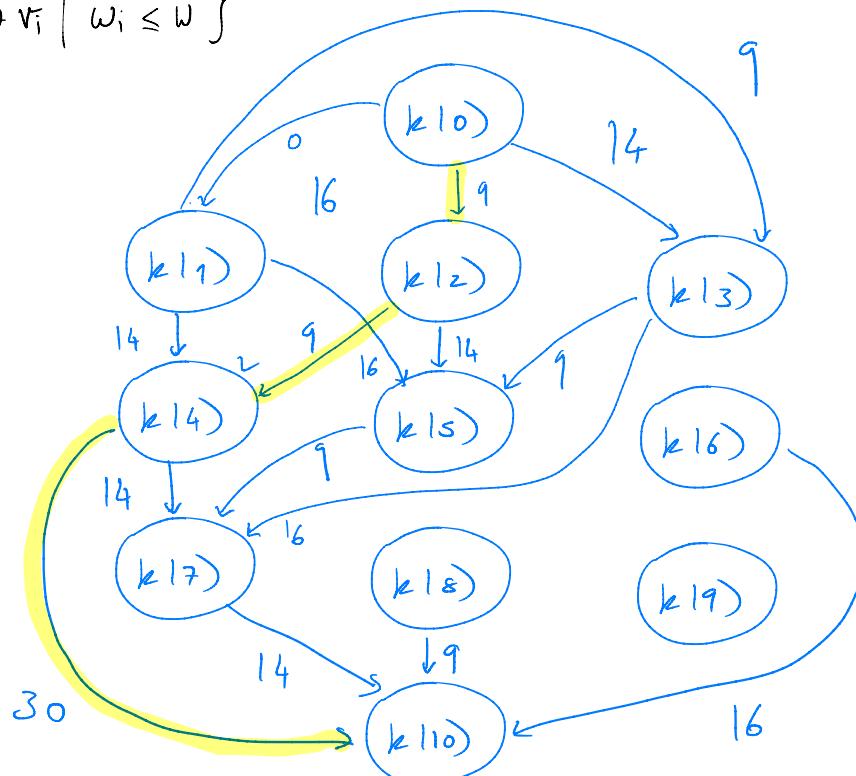
## Problema de Mochila NÃO Fracionária

- Problema de Machine com Repetição

$k(w) \Rightarrow$  maior valor e conseguimos transformar numa  
máquina com eficiência  $w$

$$k(w) = \max \left\{ k(w - w_i) + r_i \mid w_i \leq w \right\}$$

Item	Price	Value	<u>W = 10</u>
1	6	\$30	
2	3	\$14	
3	4	\$16	
4	2	\$9	



(incompleto)

- Queremos encontrar...  
o caminho mais longo no gráfico

## Problema de Machila NÃO Fracionária

- Problema de Machila com Repetição

- Implementação Naiva

Knapsack ( $w, \vec{v}, \vec{w}, n$ )

let  $k = 0$

for  $i=1$  to  $n$

if ( $\vec{v}[i] \leq w$ )

$k = \max(k, knapsack(w - \vec{w}[i], \vec{v}, \vec{w}, n) + \vec{v}[i])$

$$\underline{\underline{T(n)}} = O(n^V)$$

$$k(w) = \max \left\{ k(w - w_i) + v_i \mid w_i \leq w \right\}$$

$$T(w) = \begin{cases} n \times T(W-1) + O(1) & \text{se } W > 0 \\ O(1) & \text{se } W = 0 \end{cases}$$

$$T(w) = O(1) \quad \longrightarrow \quad 1$$

/ ... \  $n$

$$T(w-1) \quad O(1) \quad O(1) \quad \dots \quad = \quad n$$

$$/ \dots \backslash \quad / \dots \backslash \quad \dots \quad = \quad n^2$$

$$T(w-2) \quad O(1) \quad O(1) \quad O(1) \quad O(1)$$

$$\Delta \quad \Delta \quad \Delta \quad \Delta \quad \dots \quad = \quad n^V$$

## Problema de Machila NÃO Fracionária

- Problema de Machila com Repetição

- Implementação com Memoização

$$\text{KnapSack}(\underline{w}, \vec{v}, \vec{w}, n) \quad K(w) = \max \left\{ k(w - w_i) + v_i \mid w_i \leq w \right\}$$

let  $\vec{k}[0 \dots w]$  be a  $W+1$  size vector

$$\vec{k}[0] = 0$$

for  $w=1$  to  $\underline{w}$

$$\vec{k}[w] = 0$$

for  $i=1$  to  $n$

if  $\vec{w}[i] \leq w$

$$\vec{k}[w] = \max \left( \vec{k}[w - \vec{w}[i]] + \vec{v}[i], \vec{k}[w] \right)$$

## Problema de Machila NÃO Fracionária

- Problema de Machila com Repetição

- Implementação - Programação Dinâmica

Knapsack ( $W, \vec{v}, \vec{w}, n$ )

Let  $\vec{k}$  be an array of size  $W+1$

$$\vec{k}[0] = 0$$

for  $w=1$  to  $W$

$$\{ \vec{k}[w] = \max \left\{ \vec{k}[w - \vec{w}[i]] + \vec{v}[i] \mid \vec{w}[i] \leq w \right\} \}$$

return  $\vec{k}[w]$

$$k(w) = \max \left\{ k(w - w_i) + v_i \mid w_i \leq w \right\}$$

$$\left\{ \begin{array}{l} \vec{k}[w] = 0; \\ \text{for } i=1 \text{ to } n \\ \quad \text{if } (\vec{w}[i] \leq w) \\ \quad \vec{k}[w] = \max(\vec{k}[w - \vec{w}[i]] + \vec{v}[i], \vec{k}[w]) \end{array} \right.$$

## Problema da Mochila NÃO Fracionária

- Problema da Mochila com Repetição

- Implementação - Programação Dinâmica

$$K(w) = \max \left\{ K(w - w_i) + v_i \mid w_i \leq w \right\}$$

Knapsack ( $w, \vec{v}, \vec{w}, n$ )

Let  $\vec{k}$  be an array of size  $W+1$

$$\vec{k}[0] = 0$$

for  $w = 1$  to  $W$

$$| \vec{k}[w] = 0;$$

| for  $i = 1$  to  $n$

| | if ( $\vec{w}[i] \leq w$ )

$$| | | \vec{k}[w] = \max(\vec{k}[w - \vec{w}[i]] + \vec{v}[i], \vec{k}[w])$$

return  $\vec{k}[w]$

$O(n)$

$O(n, W)$

## Problema do Machila NÃO Fracionário

- Problema do Machila com Repetição

$$k(w) = \max \left\{ k(w - w_i) + v_i \mid w_i \leq w \right\}$$

- Implementação - Programação Dinâmica vs Memoization

Knapsack ( $w, \vec{v}, \vec{w}, n$ )

let  $\vec{k}$  be an array of size  $W+1$

$\vec{k}[0] = 0$

for  $w=1$  to  $W$

,  $\vec{k}[w] = 0$ ;

for  $i=1$  to  $n$

, if ( $\vec{w}[i] \leq w$ )

; ;  $\vec{k}[w] = \max(\vec{k}[w - \vec{w}[i]] + \vec{v}[i], \vec{k}[w])$

Knapsack' ( $w, \vec{k}, \vec{v}, \vec{w}, n$ )

; if ( $\vec{k}[w] \neq n$ ) return  $\vec{k}[w]$ ;

$R = 0$ ;

for  $i=1$  to  $n$

; if ( $\vec{w}[i] \leq w$ )

; ;  $R = \max(Knapsack'(w - \vec{w}[i], \vec{k}, \vec{v}, \vec{w}, n) + \vec{v}[i], R)$

$\vec{k}[w] = R$ ;

return  $R$

Knapsack ( $w, \vec{v}, \vec{w}, n$ )

; let  $\vec{k}$  be a new array with  $W+1$  positions

; return Knapsack' ( $w, \vec{k}, \vec{v}, \vec{w}, n$ )

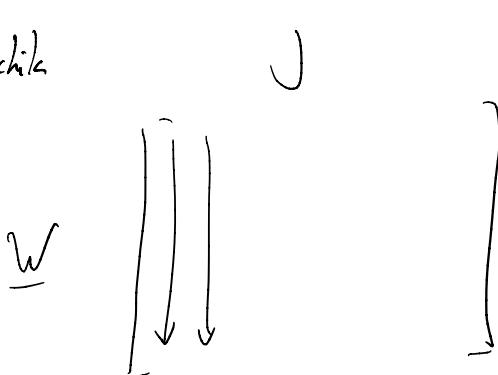
## Problema da Maquina NÃO Fracionária

- Problema da Maquina sem Repetição

\* Só podemos usar cada elemento exatamente uma vez!

- \*  $k(w, j) \Rightarrow$  maior valor  $\bar{y}$  conseguimos transportar usando uma maquina de capacidade  $w$  e os items  $1, \dots, j$

$$k(w, j) = \max(k(w - w_j, j-1) + v_j, k(w, j-1))$$



Knapsack NoRep ( $\underline{w}, \vec{v}, \vec{w}, n$ )

let  $k$  be a matrix  $(\underline{w}+1) \times (n+1)$

for  $j=0$  to  $n$

$k[0, j] = 0$

for  $w=0$  to  $\underline{w}$

$k[w, 0] = 0$

for  $j=1$  to  $n$   
for  $w=1$  to  $\underline{w}$   
 $k[w, j] = \max(k[w - \vec{w}_j, j-1] + \vec{v}_j, k[w, j-1])$   
return  $k[\underline{w}, n]$

## Maior Subsequência Comum

- Uma sequência  $\vec{z} = \langle z_1, \dots, z_n \rangle$  é uma **sobsequência** de uma outra sequência  $\vec{x} = \langle x_1, \dots, x_m \rangle$  se existir uma sequência de índices  $\langle i_1, \dots, i_n \rangle$  tal que:

$$\langle x_{i_1}, \dots, x_{i_n} \rangle = \langle z_1, \dots, z_n \rangle$$

- Dadas duas sequências  $\vec{x} = \langle x_1, \dots, x_n \rangle$  e  $\vec{y} = \langle y_1, \dots, y_m \rangle$ , dizemos que  $\vec{z} = \langle z_1, \dots, z_k \rangle$  é uma **sobsequência comum** de  $\vec{x}$  e  $\vec{y}$  se  $\vec{z}$  é uma subsequência de  $\vec{x}$  e de  $\vec{y}$ .

- $\vec{x} = \langle A, B, C, B, D, A, B \rangle$

- $\vec{y} = \langle B, D, C, A, B, A \rangle$

- $\vec{z} = \langle B, C, A, B \rangle$

- $\vec{z} = \langle B, C, B, A \rangle$

## Maior Subsequência Comum

- Uma sequência  $\vec{z} = \langle z_1, \dots, z_n \rangle$  é uma subsequência de uma outra sequência  $\vec{x} = \langle x_1, \dots, x_m \rangle$  se existir uma sequência de índices  $\langle i_1, \dots, i_n \rangle$  tal que:  
 $\langle x_{i_1}, \dots, x_{i_n} \rangle = \langle z_1, \dots, z_n \rangle$
- Dadas duas sequências  $\vec{x} = \langle x_1, \dots, x_n \rangle$  e  $\vec{y} = \langle y_1, \dots, y_m \rangle$ , dizemos que  $\vec{z} = \langle z_1, \dots, z_k \rangle$  é uma subsequência comum de  $\vec{x}$  e  $\vec{y}$  se  $\vec{z}$  é uma subsequência de  $\vec{x}$  e de  $\vec{y}$ .

- $\vec{x} = \langle A, B, C, B, D, A, B \rangle$
- $\vec{y} = \langle B, D, C, A, B, A \rangle$

- $\vec{z} = \langle B, C, A, B \rangle$
- $\vec{z}' = \langle B, C, B, A \rangle$
- $\vec{z}'' = \langle B, D, A, B \rangle$

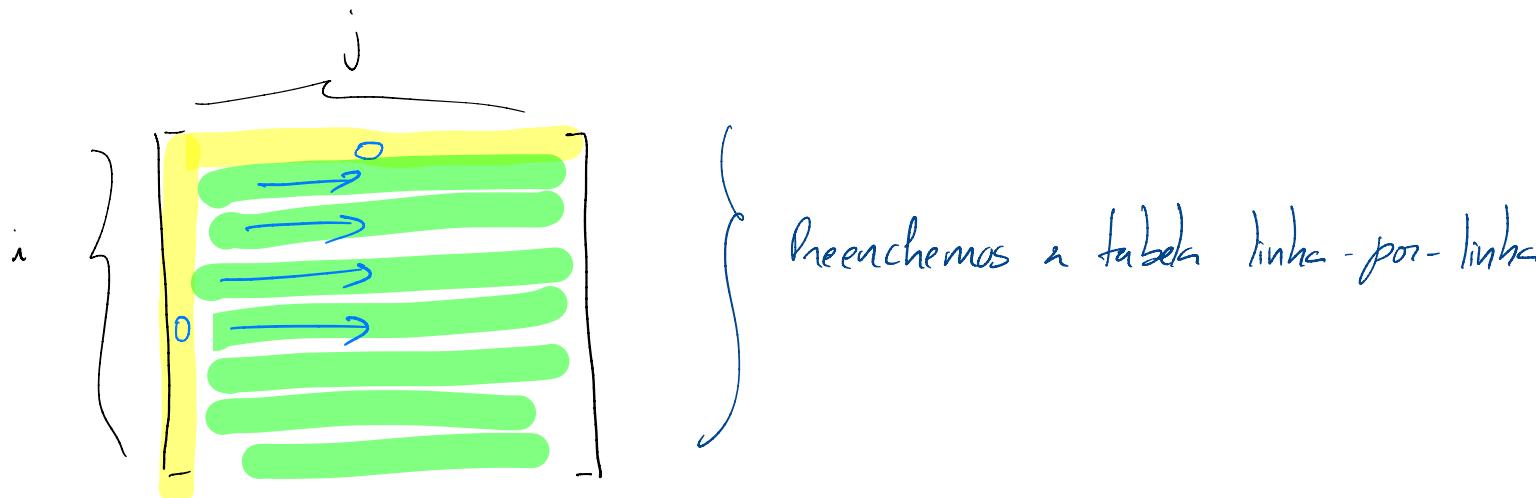
$\left. \begin{array}{l} \\ \\ \end{array} \right\}$  Algumas subsequências comuns

## Maior Subsequência Comum

Solução Recursiva       $\vec{x} = \langle x_1, \dots, x_n \rangle$  e  $\vec{y} = \langle y_1, \dots, y_m \rangle$

$C[i, j]$   $\Rightarrow$  tamanho da maior subsequência comum entre  
 $\langle x_1, \dots, x_i \rangle$  e  $\langle y_1, \dots, y_j \rangle$

$$C[i, j] = \begin{cases} 0 & \text{se } i=0 \text{ ou } j=0 \\ C[i-1, j-1] + 1 & \text{se } x_i = y_j \\ \max(C[i-1, j], C[i, j-1]) & \text{c.c.} \end{cases}$$



## Maior Subsequência Comum

### Solução Recursiva

$$\vec{x} = \langle x_1, \dots, x_n \rangle \text{ e } \vec{y} = \langle y_1, \dots, y_m \rangle$$

$C[i, j]$   $\Rightarrow$  tamanho da maior subsequência comum entre

$$\langle x_1, \dots, x_i \rangle \text{ e } \langle y_1, \dots, y_m \rangle$$

$$C[i, j] = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0 \\ C[i-1, j-1] + 1 & \text{se } x_i = y_j \\ \max(C[i-1, j], C[i, j-1]) & \text{c.c.} \end{cases}$$

$LCS(\vec{x}, \vec{y})$

let  $n = \vec{x}.length$

let  $m = \vec{y}.length$

let  $C[0, \dots, n; 0, \dots, m]$  be a new  $(n+1) \times (m+1)$  table

let  $B[0, \dots, n, 1, \dots, m]$  be a new  $n \times m$  table

for  $i = 0$  to  $n$

$$C[i, 0] = 0$$

for  $j = 0$  to  $m$

$$C[0, j] = 0$$

```

for i=1 to n
  for j=1 to m
    if  $\vec{x}[i] == \vec{y}[j]$ 
       $C[i, j] = C[i-1, j-1] + 1$ 
       $B[i, j] = "↖"$ 
    else
      if  $C[i-1, j] > C[i, j-1]$ 
         $C[i, j] = C[i-1, j]; B[i, j] = "↑"$ 
      else
         $C[i, j] = C[i, j-1]; B[i, j] = "←"$ 
  return C and B

```

## Maior Subsequência Comum - Exemplo

	O	B	D	C	A	B	A
O	0	0	0	0	0	0	0
A	0	0↑	0↑	0↑	1↖	1↖	1↑
B	0	1↖	1↖	1↖	1↑	2↑	2↖
C	0	1↑	1↑	2↑	2↖	2↑	2↑
B	0	1↖	1↑	2↑	2↑	3↑	3↖
D	0	1↑	2↑	2↑	2↑	3↑	3↑
A	0	1↑	2↑	2↑	3↖	3↑	4↑
B	0	1↖	2↑	2↑	3↑	4↑	4↑

## Maior Subsequência Comum - Exemplo

	O	B	D	C	A	B	A
O	0	0	0	0	0	0	0
A	0	0↑	0↑	0↑	1↑	1↖	1↑
B	0	1↖	1↖	1↖	1↑	2↑	2↖
C	0	1↑	1↑	2↑	2↖	2↑	2↑
B	0	1↖	1↑	2↑	2↑	3↑	3↖
D	0	1↑	2↑	2↑	2↑	3↑	3↑
A	0	1↑	2↑	2↑	3↖	3↑	4↑
B	0	1↖	2↑	2↑	3↑	4↑	4↑



## Maior Subsequência Comum - Exemplo

	0	B	D	C	A	B	A	
0	0	0	0	0	0	0	0	0
A	0	0↑	0↑	0↑	1↑	1↖	1↑	
B	0	1↖	1↖	1↖	1↑	2↑	2↖	
C	0	1↑	1↑	2↑	2↖	2↑	2↑	
B	0	1↑	1↑	2↑	2↑	3↑	3↖	
D	0	1↑	2↑	2↑	2↑	3↑	3↑	
A	0	1↑	2↑	2↑	3↖	3↑	4↑	
B	0	1↖	2↑	2↑	3↑	4↖	4↑	

PrintLCS(B, X, i, j)

if (i == 0) || (j == 0)  
return

if B[i, j] == "↑"  
print X[i]

else  
if B[i, j] == "↖"  
PrintLCS(B, X, i, j-1)  
else  
PrintLCS(B, X, i-1, j)

## Edit Distance

- Edit Distance: medida de semelhança entre strings.

Exemplo: sunny e snowy

• 

- Custo: n° de operações em que as duas strings diferem.

• Edit Distance: N° mínimo de operações q temos de aplicar para transformar uma string na outra. Q operações?

- Inserções (Insertions)

- Remoções (Deletions)

- Substituições (Substitutions)

## Edit Distance

- Soluzione Recursiva: edit distance entre  $\vec{x} = \langle x_1, \dots, x_n \rangle \in \mathcal{X}$  e  $\vec{y} = \langle y_1, \dots, y_m \rangle$

$$E[i, j] = \begin{cases} 0 & \text{se } i=0 \wedge j=0 \\ E[i-1, j-1] & \text{se } \vec{x}[i] = \vec{y}[j] \text{ e } i, j > 0 \\ \min(E[i-1, j] + 1, E[i, j-1] + 1, E[i-1, j-1] + 1) & \text{c.c.} \end{cases}$$

	S	W	O	W	Y
0	1 <sup>↑</sup>	2 <sup>↑</sup>	3 <sup>↑</sup>	4 <sup>↑</sup>	5 <sup>↑</sup>
S	1 <sup>↑</sup>	0 <sup>↑</sup>	1 <sup>↖</sup>	2 <sup>↖</sup>	3 <sup>↖</sup>
W	2 <sup>↑</sup>	1 <sup>↑</sup>	1 <sup>↖</sup>	2 <sup>↖</sup>	3 <sup>↖</sup>
O	3 <sup>↑</sup>	2 <sup>↑</sup>	1 <sup>↖</sup>	2 <sup>↖</sup>	3 <sup>↖</sup>
W	4 <sup>↑</sup>	3 <sup>↑</sup>	2 <sup>↖</sup>	2 <sup>↖</sup>	3 <sup>↖</sup>
Y	5 <sup>↑</sup>	4 <sup>↑</sup>	3 <sup>↑</sup>	3 <sup>↑</sup>	3 <sup>↖</sup>

	S	N	O	W	Y	
S	0↑	0↑	1↖	2↖	3↖	4↖
V	2↑	1↑	1↖	2↖	2↖	3↖
N	3↑	2↑	1↖	2↖	2↖	3↖
W	4↑	3↑	2↖	2↖	3↖	4↖
Y	5↑	4↑	3↑	3↑	3↑	3↑

	S	N	O	W	Y	
S	0↑	0↑	1↖	2↖	3↖	4↖
V	2↑	1↑	1↖	2↖	2↖	3↖
N	3↑	2↑	1↖	2↖	2↖	3↖
W	4↑	3↑	2↖	2↖	3↖	4↖
Y	5↑	4↑	3↑	3↑	3↑	3↑

	S	N	O	W	g	
I	0 ←	2 ←	3 ←	4 ←	5 ←	
S	1 ↑	0 ↑	1 ←	2 ←	3 ←	4 ←
V	2 ↑	1 ↑	1 ↑	2 ↑	3 ↓	4 ↓
N	3 ↑	2 ↑	1 ↑	2 ↓	3 ↓	4 ↓
W	4 ↑	3 ↑	2 ↗	2 ↑	3 ↓	4 ↑
g	5 ↑	4 ↑	3 ↑	3 ↑	3 ↓	3 ↓

(I)

s u o w g  
s u n n g  
✓ T T T ✓

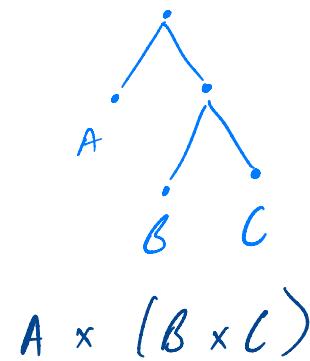
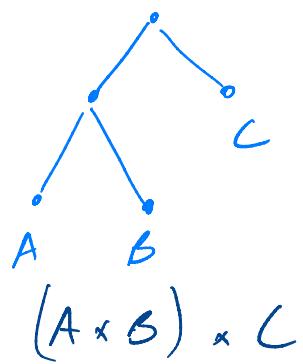
	S	N	O	W	g	
I	0 ←	2 ←	3 ←	4 ←	5 ←	
S	1 ↑	0 ↑	1 ←	2 ←	3 ←	4 ←
V	2 ↑	1 ↑	1 ↑	2 ↑	3 ↓	4 ↓
N	3 ↑	2 ↑	1 ↑	2 ↓	3 ↓	4 ↓
W	4 ↑	3 ↑	2 ↗	2 ↑	3 ↓	4 ↑
g	5 ↑	4 ↑	3 ↑	3 ↑	3 ↓	3 ↓

(II)

s - n o w g  
s o n n - g  
✓ D V T I ✓

# Multiplicação de Matrizes

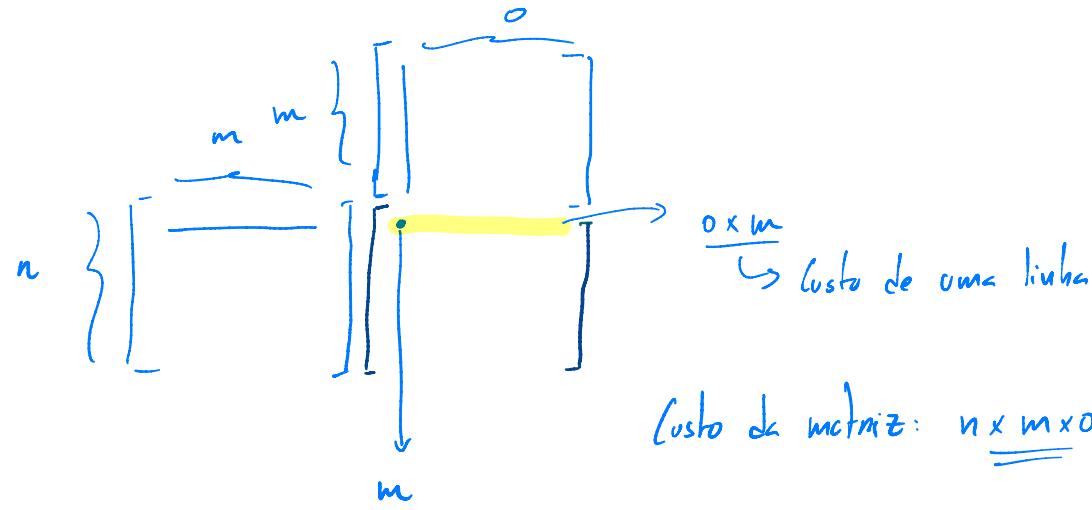
- $A \times B \times C$   
 $\downarrow \quad \downarrow \quad \downarrow$   
 $a \times b_1 \quad b_1 \times b_2 \quad b_2 \times c$



- Queremos escolher a configuração que minimize o nº de multiplicações escalares.

## Multiplicação de Matrizes

- $(n \times m) \times (m \times o) : n \times m \times o$



$$\begin{matrix} & A \times B \times C \\ \downarrow & \downarrow & \downarrow \\ a \times b_1 & b_1 \times b_2 & b_2 \times c \end{matrix}$$

①  $\left[ \begin{matrix} (A \times B) \times C \\ a \times b_2 \quad b_2 \times c \end{matrix} \right]$  Custo:  $a \times b_1 \times b_2 + a \times b_2 \times c$

$\neq$

②  $\left[ \begin{matrix} A \times (B \times C) \\ a \times b_1 \quad b_1 \times c \end{matrix} \right]$  Custo:  $b_1 \times b_2 \times c + a \times b_1 \times c$

## Multiplicação de Matrizes

$$\bullet A \times B \times C$$

$$\begin{matrix} \downarrow & \downarrow & \downarrow \\ a \times b_1 & b_1 \times b_2 & b_2 \times c \end{matrix}$$

$$\textcircled{1} \quad \begin{matrix} (A \times B) \times C \\ \overbrace{a \times b_2}^{\downarrow} \quad \overbrace{b_2 \times c}^{\downarrow} \end{matrix} \quad ] \text{ Custo: } a \times b_1 \times b_2 + a \times b_2 \times c$$

≠

$$\textcircled{2} \quad \begin{matrix} A \times (B \times C) \\ \downarrow \quad \overbrace{b_1 \times c}^{\downarrow} \\ a \times b_1 \end{matrix} \quad ] \text{ Custo: } b_1 \times b_2 \times c + a \times b_1 \times c$$

$$\bullet A \times B \times C$$

$$\begin{matrix} \downarrow & \downarrow & \downarrow \\ 50 \times 20 & 20 \times 1 & 1 \times 10 \end{matrix}$$

$$\textcircled{1} \quad \text{Custo: } 50 \times 20 \times 1 + 50 \times 1 \times 10 = 1000 + 500 \\ = \underline{\underline{1500}}$$

$$50 \times 20 \times 1 \times 10$$

$$\textcircled{2} \quad \text{Custo: } 20 \times 1 \times 10 + 50 \times 20 \times 10 = 200 + 10.000 \\ = \underline{\underline{10200}} \quad \quad \quad \neq$$

## Multiplicação de Matrizes

$\vec{A} = \langle A_1, \dots, A_n \rangle \Rightarrow$  Queremos calcular  $A_1 \times \dots \times A_n$

$$\bullet \underbrace{A_i \times \dots \times A_k}_{m_{i-1} \times m_k} \times \underbrace{A_{k+1} \times \dots \times A_j}_{m_k \times m_j}$$

$C[i, j]$ : menor custo da multiplicação  $A_i \times \dots \times A_j$

$$C[i, j] = \begin{cases} \min_{i \leq k < j} \{ C[i, k] + C[k+1, j] + m_{i-1} \times m_k \times m_j \} & \text{se } j > i \\ 0 & \text{se } i = j \\ \perp & \text{c.c.} \end{cases}$$

# Multiplicação de Matrizes

$$C[i, j] = \begin{cases} \min_{i \leq k < j} \{ C[i, k] + C[k+1, j] + m_{i-1} \times m_k \times m_j \} & \text{se } j > i \\ 0 & \text{se } i = j \\ \perp & \text{c.c.} \end{cases}$$

M Multiplication Cost ( $\vec{m}, n$ ) //  $\vec{m} [0, \dots, n]$  vetor das dimensões

let  $C[1..n, 1..n]$  be a  $n \times n$  matrix

for  $i=1$  to  $n$

|:  $C[i, i] = 0$

for  $s=1$  to  $(n-1)$

|: for  $i=1$  to  $n-s$

|: |:  $C[i, i+s] = \min_{i \leq k < i+s} \{ C[i, k] + C[k+1, i+s] + \vec{m}[i-1] \times \vec{m}[k] \times \vec{m}[j] \}$

return  $C[1..n]$

# Multiplicação de Matrizes

$$C[i, j] = \begin{cases} \min_{i \leq k < j} \{ C[i, k] + C[k+1, j] + m_{i-1} \times m_k \times m_j \} & \text{se } j > i \\ 0 & \text{se } i = j \\ \perp & \text{c.c.} \end{cases}$$

M Multiplication Cost ( $\vec{m}, n$ ) //  $\vec{m} [0, \dots, n]$  vetor das dimensões

let  $C[1..n, 1..n]$  be a  $n \times n$  matrix

for  $i=1$  to  $n$

|:  $C[i, i] = 0$

for  $s=1$  to  $(n-1)$

|: for  $i=1$  to  $n-s$

|: |:  $C[i, i+s] = \min_{i \leq k < i+s} \{ C[i, k] + C[k+1, i+s] + \vec{m}[i-1] \times \vec{m}[k] \times \vec{m}[i+s] \}$

$O(n^3)$

return  $C[1..n]$

## Travelling Salesman Problem

Input: Grafo pesado  $G = (V, E, w)$

Output: Caminho circular de menor peso a começar no vértice  $v_0$  que passe exactamente uma vez em cada um dos outros vértices do grafo.

Solução naïve: enumere todos os caminhos possíveis.

Quantos caminhos existem?  $\frac{n!}{=}$  (onde  $n = |V|$ )

$$10! = \underline{\underline{3\,628\,800}}$$

## Travelling Salesman Problem

Input: Grafo pesado  $G = (V, E, w)$

Output: Caminho circular de menor peso a começar no vértice  $v_i$  que passe exactamente uma vez em cada um dos outros vértices do grafo.

$C(S, v_j)$ : peso do caminho mais curto a começar em  $v_i$  e terminar em  $v_j$  passando por todos os vértices em  $S$  exactamente uma vez.

$$C(S, v_j) = \min \left\{ C(S \setminus \{v_j\}, v_i) + w_{ij} \mid v_i \in S \setminus \{v_j\} \right\}$$

$$C(\{\}, v_1) = 0$$

## Travelling Salesman Problem

$$C(s, v_j) = \min \left\{ C(s \setminus \{v_j\}, v_i) + w_{ij} \mid v_i \in S \setminus \{v_j\} \right\}$$

$$C(\{v_i\}, v_i) = 0$$

for  $s=2$  to  $n$

for all subsets  $S \subseteq \{v_1, \dots, v_n\}$

$$C(S, v_i) = \infty$$

for all  $v_j \in S \setminus \{v_i\}$

$$C(S, v_j) = \min \left\{ C(S \setminus \{v_j\}, v_i) + w_{ij} \mid v_i \in S \setminus \{v_j\} \right\}$$

return  $\min \left\{ C(V, v_i) + w_{i1} \mid v_i \in V \right\}$

## Travelling Salesman Problem

$$C(s, v_j) = \min \left\{ C(s \setminus \{v_j\}, v_i) + w_{ij} \mid v_i \in S \setminus \{v_j\} \right\}$$

$$C(\{v_i\}, v_i) = 0$$

for  $s=2$  to  $n$

for all subsets  $S \subseteq \{v_1, \dots, v_n\}$

$$C(S, v_i) = \infty$$

for all  $v_j \in S \setminus \{v_i\}$

$$C(S, v_j) = \min \left\{ C(S \setminus \{v_j\}, v_i) + w_{ij} \mid v_i \in S \setminus \{v_j\} \right\}$$

How many subsets of  $V$ ?  
 $\hookrightarrow 2^n$

return  $\min \left\{ C(V, v_i) + w_{i1} \mid v_i \in V \right\}$   $O(n^2 2^n)$

$$10! = 3628800$$

$$10^2 \times 2^{10} = 102400$$

## Subsequência Contígua de Soma Máxima

$$A = \langle 5, 15, -30, 10, -5, 40, 10 \rangle$$

- $S[i]$  ⇒ o valor da soma da subsequência contígua de soma máxima a terminar na posição  $i$ .

$$S[i] = \begin{cases} S[i-1] + A[i] & \text{se } S[i-1] \geq 0 \\ A[i] & \text{se } S[i-1] < 0 \\ 0 & \text{se } i=0 \end{cases}$$

## Subsequência Contígua de Soma Máxima

$$A = \langle 5, 15, -30, 10, -5, 40, 10 \rangle$$

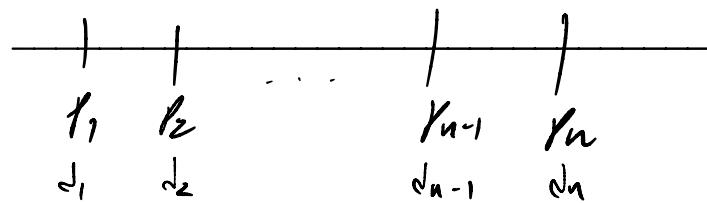
- $S[i]$  ⇒ o valor da soma da subsequência contígua de soma máxima a terminar na posição  $i$ .

$$S[i] = \begin{cases} S[i-1] + A[i] & \text{se } S[i-1] \geq 0 \\ A[i] & \text{se } S[i-1] < 0 \\ 0 & \text{se } i=0 \end{cases}$$

$$\langle 5, 15, -30, 10, -5, 40, 10 \rangle$$

$$\begin{array}{ccccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ S & 20 & -10 & 10 & 5 & 45 & 55 \end{array}$$

## Restaurantes na Estrada

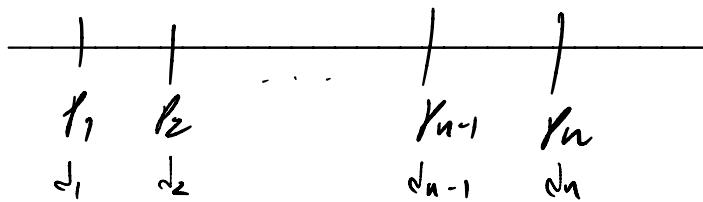


- $p_i \Rightarrow$  lucro esperado do restaurante  $i$
- $d_i \Rightarrow$  distância do restaurante  $i$

Restrição: Restaurantes consecutivos têm de estar a no  
máximo  $k$  km de distância

$L[i] \Rightarrow$  lucro máximo possível à distância  $i$

## Restaurantes na Estrada



- $p_i \Rightarrow$  lucro esperado do restaurante  $i$
- $d_i \Rightarrow$  distância do restaurante  $i$

Restrição: Restaurantes consecutivos têm de estar a no máximo  $k$  km de distância

$L[i] \Rightarrow$  lucro máximo possível à distância  $i$

$$L[i] = \max \left\{ L[j] + \alpha(d_j, d_i) \cdot p_i \mid j < i \right\}$$

$$\alpha(d_j, d_i) = \begin{cases} 1 & \text{se } d_i - d_j \geq k \\ 0 & \text{c.c.} \end{cases}$$