Aula 5

- Programação Dinâmica
  - Problema da Mochila
    (com e sem repetição)
- Subsequência Comum mais longa
- Multiplicações de Matrizes

Problema da Mochila NÃO Fraccionária

- Problema da mochila não fraccionária
  - Com replicação: dispomos de quantidades ilimitadas de cada item
  - Sem replicação: dispomos de uma quantidade limitada de cada item

- Problema da mochila não fraccionária com repetição

Input:
- $\vec{w}[1, \ldots, n]$: vector de pesos
- $\vec{v}[1, \ldots, n]$: vector de valores

Output:
- k: valor que conseguimos transportar na mochila.

Problema da Mochila NÃO Fraccionária

- Problema da mochila não fraccionária
  - Com replicação: dispomos de quantidades ilimitadas de cada item
  - Sem replicação: dispomos de uma quantidade limitada de cada item

- Problema da mochila não fraccionária com repetição

Input:
- $\vec{w}[1, \ldots, n]$: vector de pesos
- $\vec{v}[1, \ldots, n]$: vector de valores

Output:
- k: valor que conseguimos transportar na mochila.

Solução Recursiva:

$$k(W) = \max \left\{ k(W - w_k) + v_k \;\middle|\; \begin{array}{l} 1 \leq k \leq n, \\ w_k \leq W \end{array} \right\}$$

$k(W)$ → valor q conseguimos transportar numa mochila com capacidade W

Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição

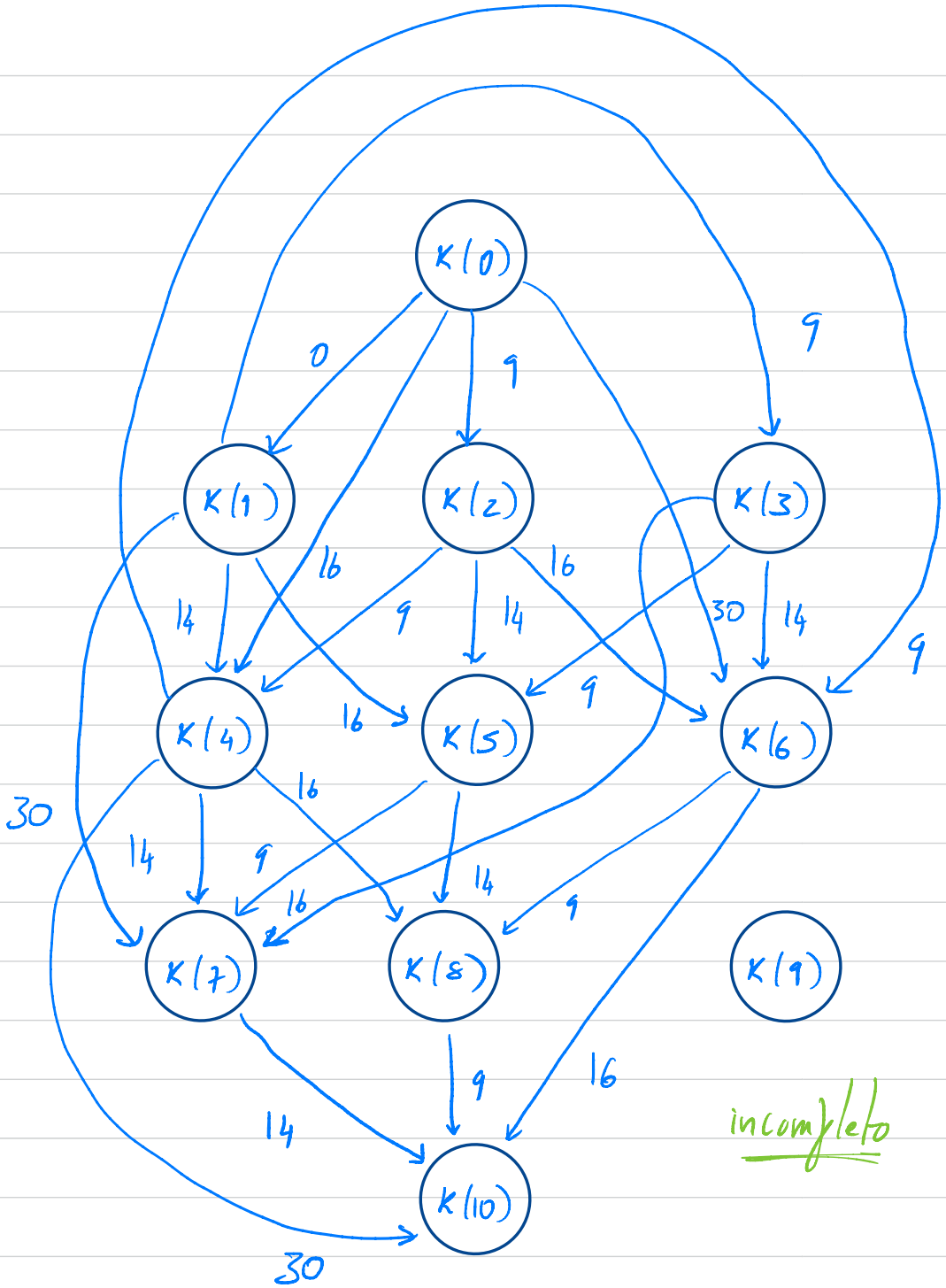
  - $K(W)$: maior valor que conseguimos transportar numa mochila com capacidade W

$$K(W) = \max \{ K(W - v_k) + v_k \mid w_k \leq W \}$$

Exemplo:

$W = 10$

Item	Peso	Valor
1	6	30 $
2	3	14 $
3	4	16 $
4	2	9 $



Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição

  - $K(W)$: maior valor que conseguimos transportar numa mochila com capacidade W

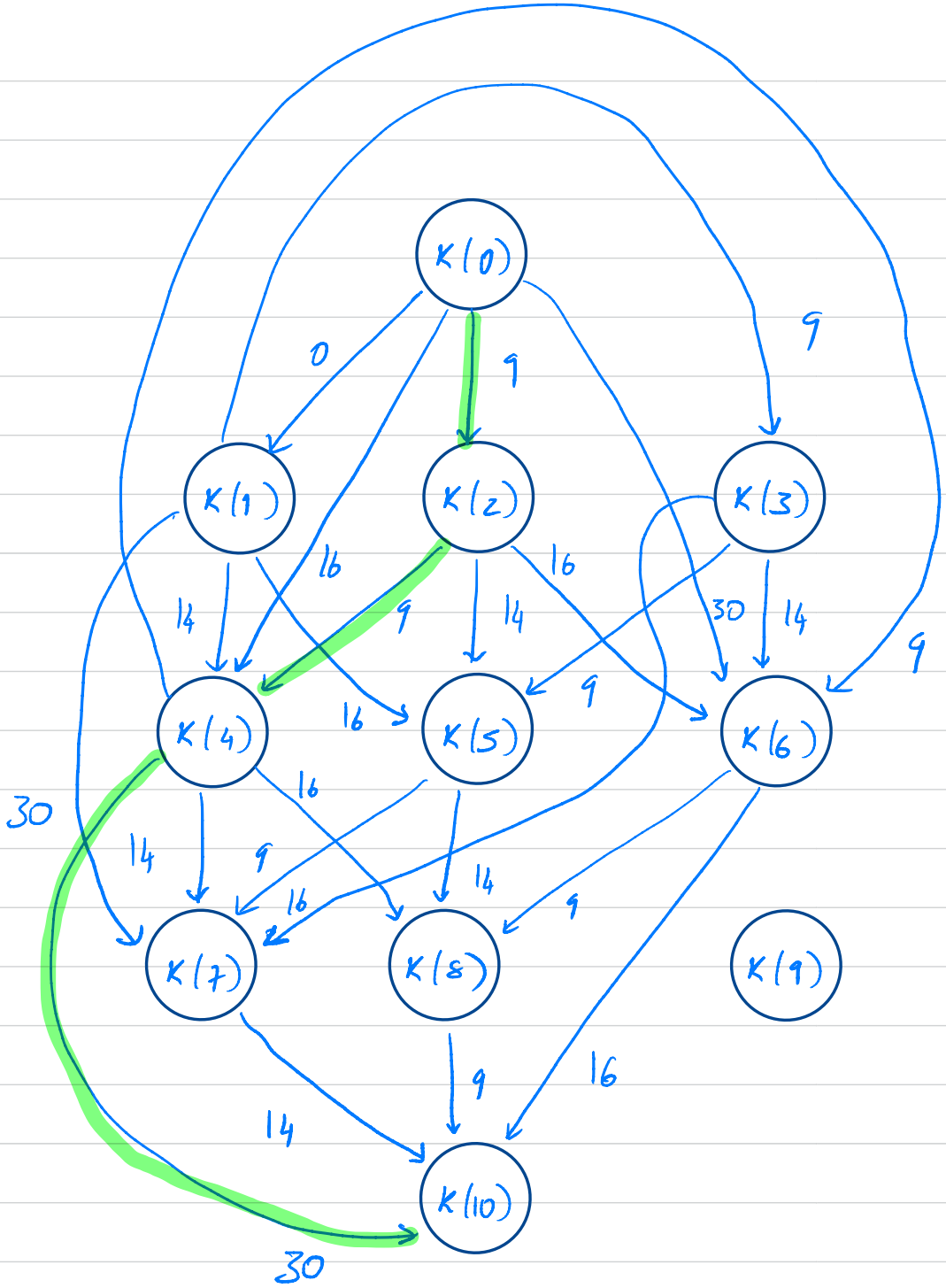
$$K(W) = \max \{ K(W - v_k) + v_k \mid w_k \leq W \}$$

Exemplo:

$W = 10$

Item	Peso	Valor
1	6	30 $
2	3	14 $
3	4	16 $
4	2	9 $

Caminho mais longo em DAG



Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição – Implementação naïf

$$K(W) = \max \{ K(W - v_k) + v_k \mid w_k \leq W \}$$

```
knapsack(W, v⃗, w⃗, n)
    let k = 0
    for i = 1 to n
        if w⃗[i] ≤ W
            k := max(k, KnapSack(W - w⃗[i], v⃗, w⃗, n))
    return k
```

Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição – Implementação naïf

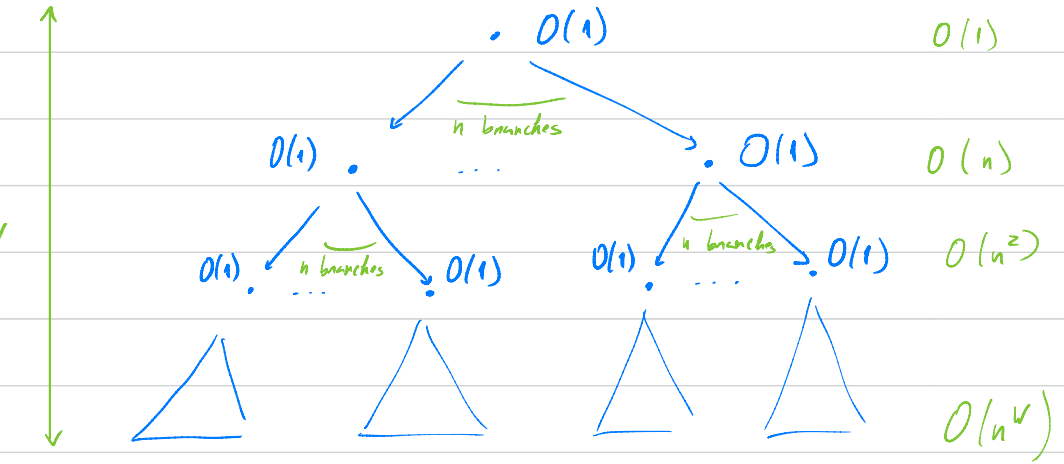
$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

```
knapsack(W, v⃗, w⃗, n)
    let k = 0
    for i = 1 to n
        if w⃗[i] ≤ W
            k := max(k, knapSack(W - w⃗[i], v⃗, w⃗, n))
    return k
```

Análise de Complexidade

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

$$T(W) = O(n^W)$$



Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição – Programação Dinâmica

$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

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

```
let k[0..W] be a new vector initialized to 0
for w=1 to W
    for i:=1 to n
        if (w[i] ≤ w)
            k[w] = max ( k[w], k[w - w[i]] + v[i] )
```

Problema da Mochila Não Fraccionária

- Problema da Mochila sem repetição – Programação Dinâmica

$$K(W) = \max \{ K(W - V_k) + V_k \mid W_k \leq W \}$$

```
Knapsack (W, V⃗, W⃗, n)

    let K⃗[0..W] be a new vector
    for w=1 to W
        for i := 1 to n
            if (W⃗[i] ≤ w)
                K⃗[w] = max ( K⃗[w], K⃗[w - W⃗[i]] + V⃗[i] )
```

Análise de Complexidade

$\Theta(n \cdot W)$

Problema da Mochila Não Fraccionária

Programação Dinâmica versus Memoization

```
Knapsack(W, v⃗, w⃗, n)
    let k⃗[0..W] be a new vector
    k⃗[0] := 0
    for w = 1 to W
        for i := 1 to n
            if (w⃗[i] ≤ w)
                k⃗[w] = max(k⃗[w], k⃗[w - w⃗[i]] + v⃗[i])
```

Programação Dinâmica
- Construção Incremental da tabela de soluções

```
KnapSack'(W, k⃗, v⃗, w⃗, n)
    if (k⃗[W] ≠ Nil) return k⃗[W]
    let r := 0
    for i = 1 to n
        if w⃗[i] ≤ W
            r := max(r, KnapSack'(W - w⃗[i], k⃗, v⃗, w⃗, n) + v⃗[i])
    k⃗[W] := r
    return r
```

Memoization
- A tabela é preenchida à medida q vamos precisando dos valores respectivos.

Problema da Mochila Não Fraccionária Sem Repetição

- Podemos usar cada elemento exactamente uma vez

$$k(W, j) = \boxed{\ldots}$$

↳ Valor máximo q podemos transportar numa mochila c/ capacidade W usando unicamente elementos do conjunto $\{1, \ldots, n\}$.

Problema da Mochila Não Fraccionária Sem Repetição

- Podemos usar cada elemento exactamente uma vez

$$k(\underline{W}, j) = \begin{cases} 0 & \text{se } T = 0 \vee \underline{W} = 0 \\ \max\left(k(\underline{W}, j-1),\ k(\underline{W} - w_j, j-1) + v_j \right) & \text{se } \underline{W} \geq w_j \\ k(\underline{W} - w_j, j-1) & \end{cases}$$

```
knapsackNoRep(W, v, w, n)
  let k[][] be a W x n array

  for w = 0 to W
      k[w, 0] := 0
  for i = 0 to n
      k[0, i] := 0
```

} inicialização

Problema da Mochila Não Fraccionária Sem Repetição

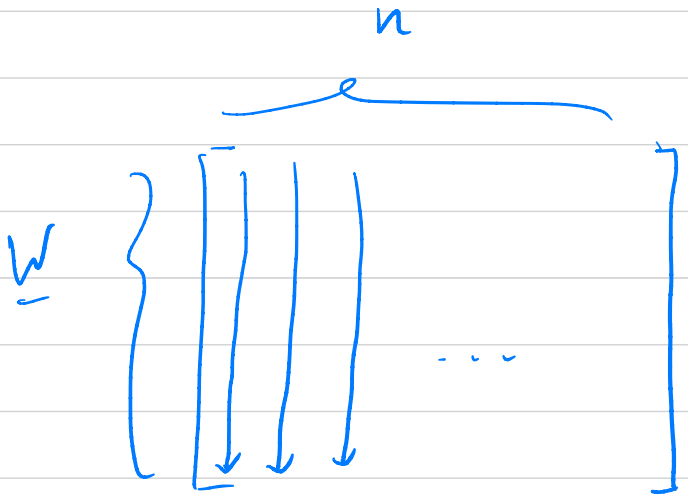
- Podemos usar cada elemento exactamente uma vez

```
knapsackNoRep( W, v⃗, w⃗, n )
  let k⃗[][] be a W x n array

  for w = 0 to W
    k⃗[w, 0] := 0
  for i = 0 to n
    k⃗[0, i] := 0

  for i = 1 to n
    for w = 1 to W
      if w⃗[i] ≤ w
        k⃗[w, i] = max( k⃗[w, i-1], k⃗[w - w⃗[i], i-1] + v⃗[i] )
      else k⃗[w, i] = k⃗[w, i-1]

  return k⃗[W, n]
```



Problema da Mochila Não Fraccionária Sem Repetição

- Podemos usar cada elemento exactamente uma vez

```
knapsackNoRep(W, v⃗, w⃗, n)
  let k⃗[][] be a W x n array

  for w = 0 to W
    k⃗[w, 0] := 0
  for i = 0 to n
    k⃗[0, i] := 0

  for i = 1 to n
    for w = 1 to W
      if w⃗[i] ≤ w
        k⃗[w, i] = max( k⃗[w, i-1], k⃗[w - w⃗[i], i-1] + v⃗[i] )
      else k⃗[w, i] = k⃗[w, i-1]

  return k⃗[W, n]
```

Análise de Complexidade

$\Theta(n \cdot W)$

Maior Subsequência Comum

Definição | Subsequência & Subsequência

- Uma sequência $\vec{Z} = \langle z_1, \ldots, z_n \rangle$ é uma **subsequência** de uma outra sequência $\vec{X} = \langle x_1, \ldots, x_m \rangle$ se existir uma sequência de índices $\langle i_1, \ldots, i_n \rangle$ tal que:

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

- Dadas duas sequências $\vec{X} = \langle x_1, \ldots, x_n \rangle$ e $Y = \langle y_1, \ldots, y_m \rangle$, dizemos que $\vec{Z} = \langle z_1, \ldots, z_k \rangle$ é uma **subsequência comum** de $\vec{X}$ e $\vec{Y}$ se $\vec{Z}$ é uma subsequência de $\vec{X}$ e de $\vec{Y}$.

Exemplo:

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

Maior Subsequência Comum

Definição | Subsequência & Subsequência

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

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

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

Exemplo:

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

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

É ainda: $\vec{z}'' = \langle B, D, A, B \rangle$

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

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

Maior Subsequência Comum

Solução Recursiva

Input: $\vec{X}[1, \ldots, n]$
$\vec{Y}[1, \ldots, m]$

Output: maior subsequência comum entre $\vec{X}$ e $\vec{Y}$: $\vec{Z}[1, \ldots, k]$

$C(i,j)$: Tamanho da maior subsequência comum entre $\vec{X}[1, \ldots, i]$ e $\vec{Y}[1, \ldots, j]$

$$C(i,j) = \left\{ \right.$$

Maior Subsequência Comum

Solução Recursiva

Input: $\vec{X}[1, \ldots, n]$
$\vec{Y}[1, \ldots, m]$

Output: maior subsequência comum entre $\vec{X}$ e $\vec{Y}$: $\vec{Z}[1, \ldots, k]$

$C(i,j)$: Tamanho da maior subsequência comum entre $\vec{X}[1, \ldots, i]$ e $\vec{Y}[1, \ldots, j]$

$$C(i,j) = \begin{cases} 0 & \text{se } i = 0 \ \vee \ j = 0 \\ C(i-1, j-1) + 1 & \text{se } \vec{X}[i] = \vec{Y}[j] \\ \max(C(i-1, j), C(i, j-1)) & \text{c.c.} \end{cases}$$

Maior Subsequência Comum

Solução Recursiva

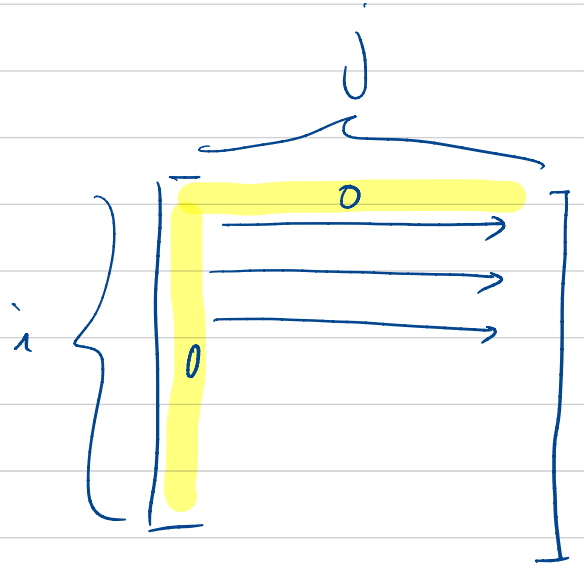
$$C(i,j) = \begin{cases} 0 & \text{se } i=0 \ \vee \ j=0 \\ C(i-1, j-1) + 1 & \text{se } \vec{x}[i] = \vec{y}[j] \\ \max(C(i-1,j), C(i,j-1)) & \text{c.c.} \end{cases}$$

```
LCS(x⃗, y⃗)
    let n = x⃗.size()
    let m = y⃗.size()
    let C[][] be an (n+1)×(m+1) matrix
    let B[][] be an n×m matrix
    fn i=0 to n
        C[i,0] := 0
    fn j=0 to n
        C[0,j] := 0
```

Maior Subsequência Comum

Solução Recursiva

$$C(i,j) = \begin{cases} 0 & \text{se } i=0 \ \vee \ j=0 \\ C(i-1,j-1)+1 & \text{se } \vec{x}[i] = \vec{y}[j] \\ \max(C(i-1,j), C(i,j-1)) & \text{c.c.} \end{cases}$$




```
LCS(x⃗, y⃗)
    let n = x⃗.size()
    let m = y⃗.size()
    let C[][] be an (n+1)×(m+1) matrix
    let B[][] be an n×m matrix
    fn i=0 to n
        C[i,0] := 0
    fn j=0 to n
        C[0,j] := 0
    fn i=1 to n
        fn j=1 to m
            if(x⃗[i] == y⃗[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,B)
```

Maior Subsequência Comum

Solução Recursiva

$$C(i,j) = \begin{cases} 0 & \text{se } i=0 \ \vee \ j=0 \\ C(i-1,j-1)+1 & \text{se } \vec{x}[i] = \vec{y}[j] \\ \max(C(i-1,j), C(i,j-1)) & \text{c.c.} \end{cases}$$

Análise de Complexidade

```
LCS(x⃗, y⃗)
    let n = x⃗.size()
    let m = y⃗.size()
    let C[][] be an (n+1)x(m+1) matrix
    let B[][] be an nxm matrix
    fn i=0 to n
        C[i,0] := 0
    fn j=0 to n
        C[0,j] := 0
    fn i=1 to n
        fn j=1 to m
            if(x⃗[i] == y⃗[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,B)
```

Maior Subsequência Comum

Solução Recursiva

$$C(i,j) = \begin{cases} 0 & \text{se } i=0 \vee j=0 \\ C(i-1,j-1)+1 & \text{se } \vec{x}[i] = \vec{y}[j] \\ \max(C(i-1,j), C(i,j-1)) & \text{c.c.} \end{cases}$$

Análise de Complexidade

$\Theta(n.m)$

```
LCS(x⃗, y⃗)
    let n = x⃗.size()
    let m = y⃗.size()
    let C[][] be an (n+1)x(m+1) matrix
    let B[][] be an nxm matrix
    fn i=0 to n
        C[i,0] := 0
    fn j=0 to n
        C[0,j] := 0
    fn i=1 to n
        fn j=1 to m
            if(x⃗[i]== y⃗[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,B)
```

Maior Subsequência Comum

		0	B	D	C	A	B	A
	0	0	0	0	0	0	0	0
A	0							
B	0							
C	0							
B	0							
D	0							
A	0							
B	0							

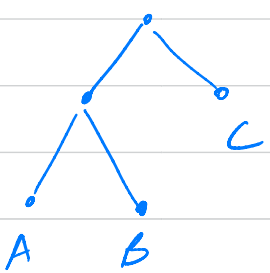
Maior Subsequência Comum

		0	B	D	C	A	B	A
	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 ↑

Multiplicação de Matrizes

- $A \times B \times C$

$a \times b_1 \quad b_1 \times b_2 \quad b_2 \times c$



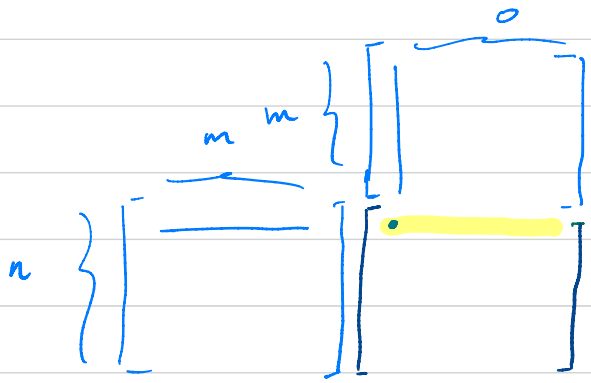
$(A \times B) \times C$

$A \times (B \times C)$

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

Multiplicação de Matrizes

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



Custo da matriz:

- $A \times B \times C$

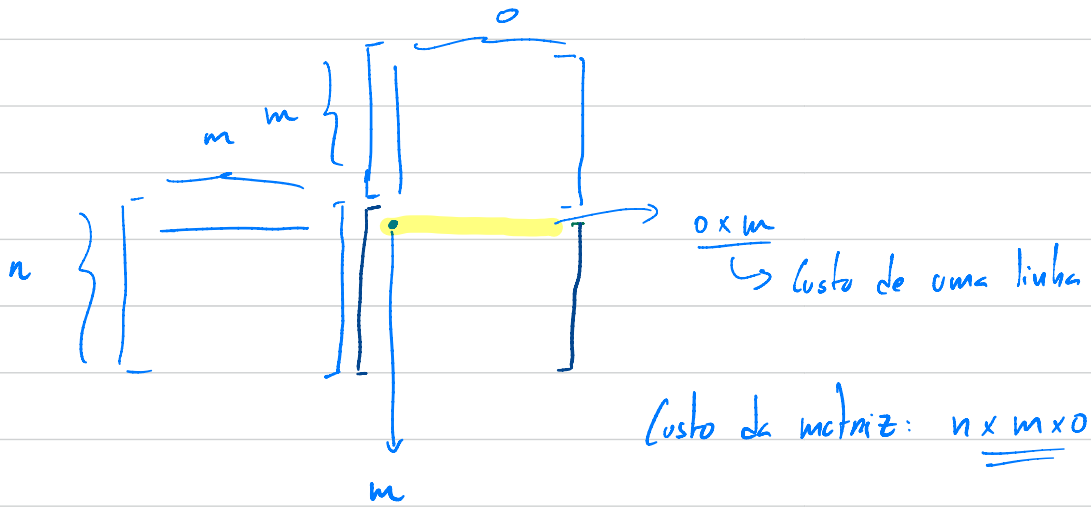
$a \times b_1 \quad b_1 \times b_2 \quad b_2 \times c$

① $(A \times B) \times C$] Custo:

② $A \times (B \times C)$] Custo:

Multiplicação de Matrizes

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



Custo da matriz: $n \times m \times o$

- $A \times B \times C$

$a \times b_1$ $b_1 \times b_2$ $b_2 \times c$

① $(A \times B) \times C$ — $a \times b_2$ — $b_2 \times c$ — Custo: $a \times b_1 \times b_2 + a \times b_2 \times c$

$\neq$

② $A \times (B \times C)$ — $a \times b_1$ — $b_1 \times c$ — Custo: $b_1 \times b_2 \times c + a \times b_1 \times c$

Multiplicação de Matrizes

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

$C[i,j]$: menor custo de multiplicar $A_i \times \ldots \times A_j$

$$C[i,j] = \begin{cases} \end{cases}$$

- $A_i \times \ldots \times A_k \times A_{k+1} \times \ldots \times A_j$

  $m_{i-1} \times m_k$ $\quad$ $m_k \times m_j$

Multiplicação de Matrizes

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

- $A_i \times \ldots \times A_k \times A_{k+1} \times \ldots \times A_j$
  - $A_i \times \ldots \times A_k$: $m_{i-1} \times m_k$
  - $A_{k+1} \times \ldots \times A_j$: $m_k \times m_j$

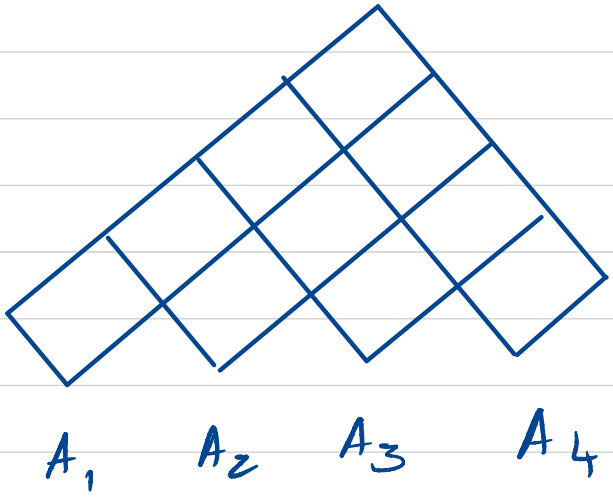
$C[i,j]$: menor custo de multiplicar $A_i \times \ldots \times A_j$

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

Multiplicação de Matrizes

- $A_1 \times A_2 \times A_3 \times A_4$

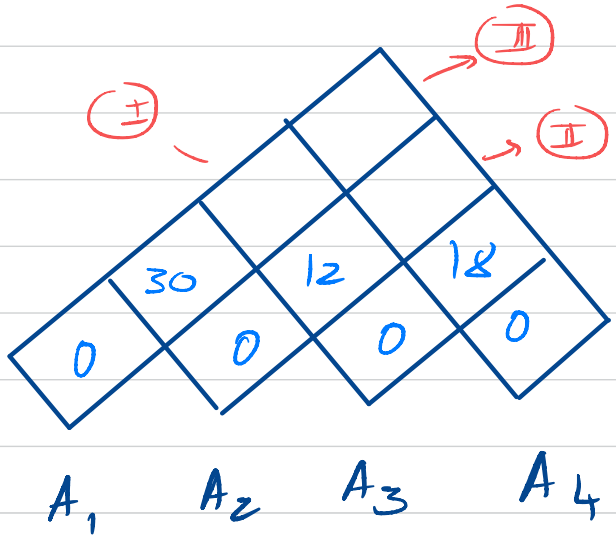
5×2 $\quad$ 2×3 $\quad$ 3×2 $\quad$ 2×3



Multiplicação de Matrizes

- $A_1 \times A_2 \times A_3 \times A_4$

A_1: 5×2 — A_2: 2×3 — A_3: 3×2 — A_4: 2×3



- $C[1,2] = 5 \times 2 \times 3 = 30$
- $C[2,3] = 2 \times 3 \times 2 = 12$
- $C[3,4] = 3 \times 2 \times 3 = 18$

(II) $C[2,4] \Rightarrow (A_2 \times A_3) \times A_4$

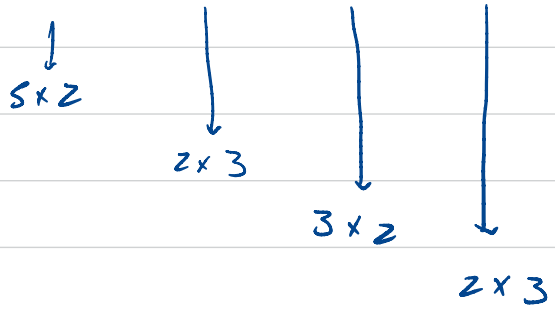
- $C[2,3] + C[4,4] + 2 \times 2 \times 3 = 12 + 0 + 12 = 24$
- $C[2,2] + C[3,4] + 2 \times 3 \times 3 = 0 + 18 + 18 = 36$

(I) $C[1,3] \Rightarrow A_1 \times (A_2 \times A_3)$

- $C[1,2] + C[3,3] + 5 \times 3 \times 2 = 30 + 0 + 30 = 60$
- $C[1,1] + C[2,3] + 5 \times 2 \times 3 = 0 + 12 + 30 = 42$

Multiplicação de Matrizes

- $A_1 \times A_2 \times A_3 \times A_4$



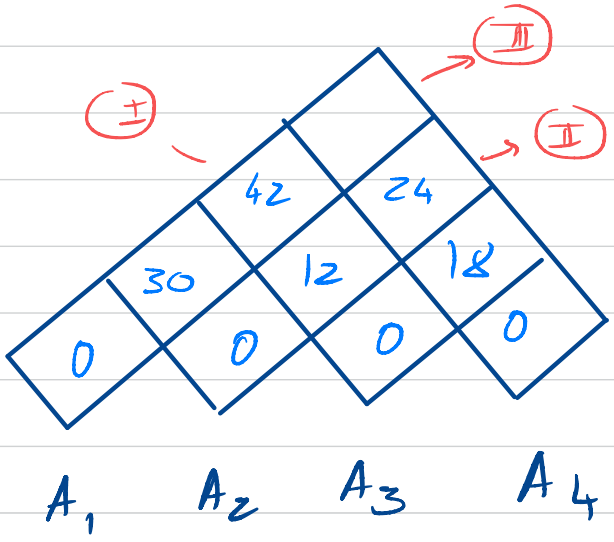
- $C[1,2] = 5 \times 2 \times 3 = 30$
- $C[2,3] = 2 \times 3 \times 2 = 12$
- $C[3,4] = 3 \times 2 \times 3 = 18$

(I) $C[1,3] \Rightarrow A_1 \times (A_2 \times A_3)$

42

(II) $C[2,4] \Rightarrow (A_2 \times A_3) \times A_4$

24



(III)

$(A_1 \times (A_2 \times A_3)) \times A_4$

- $42 + 5 \times 2 \times 3 = 72$

$A_1 \times ((A_2 \times A_3) \times A_4)$

- $24 + 5 \times 2 \times 3 = 54$

$(A_1 \times A_2) \times (A_3 \times A_4)$

- $30 + 18 + 5 \times 3 \times 3 = \underline{93}$

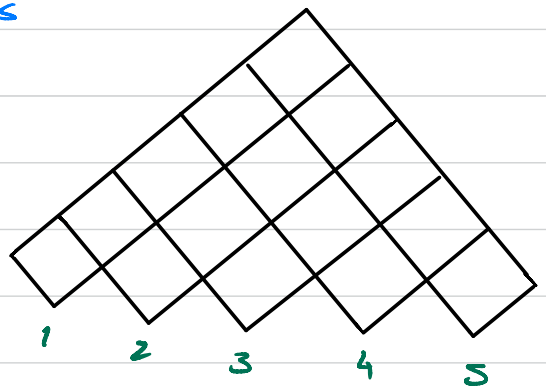
Multiplicação de Matrizes

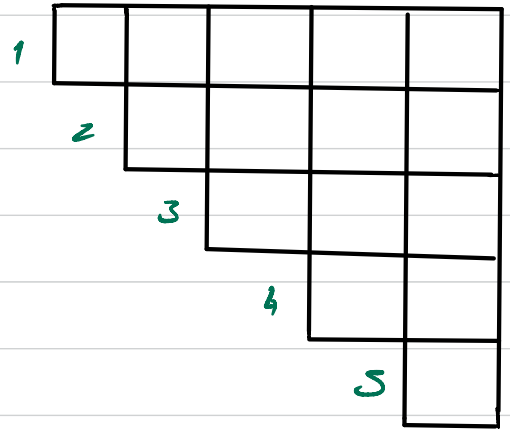
$$C[i,j] = \begin{cases} \min\limits_{i \le k < j} \{ C[i,k] + C[k+1,j] + m_{i-1} \times m_k \times m_j \} & \text{se } j > i \\ 0 & \text{se } i = j \end{cases}$$

Multiplicação de Matrizes

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

MMultiplication Cost $(\vec{m}, n)$ // $\vec{m}[0, \ldots, n]$ vector das dimensões

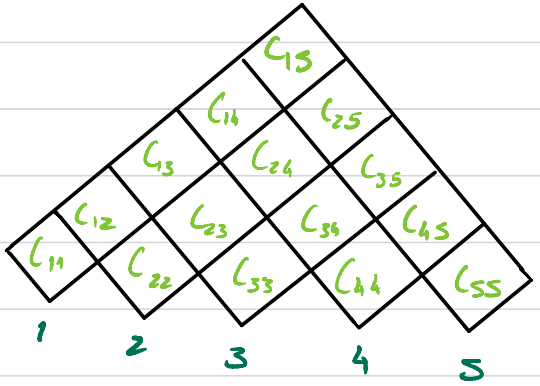


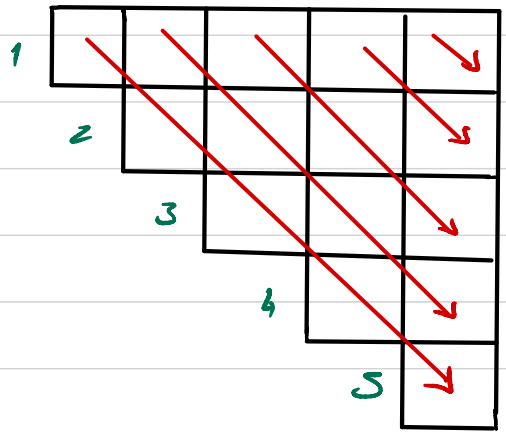


Multiplicação de Matrizes

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

```
MMultiplicationCost(m⃗, n)   // m⃗[0,...,n] vector das dimensões
let C[1,...,n; 1,...,n] be a new n x 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{ C[i,k] + C[k, i+s]
                          + m⃗[i-1] x m⃗[k] x m⃗[i+s] ) i ≤ k ≤ i+s }
return C
```

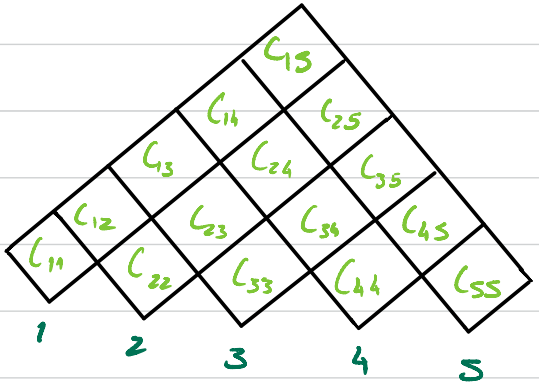


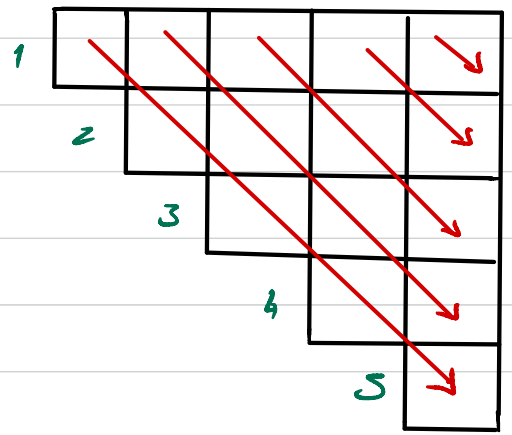


Multiplicação de Matrizes

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

```
M Multiplication Cost(m⃗, n)   // m⃗[0,...,n] vector das dimensões
    let C be a new n x 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{ C[i,k] + C[k, i+s]
                              + m⃗[i-1] x m⃗[k] x m⃗[i+s] } i≤k<i+s
    return C
```





Análise de Complexidade: $\Theta(n^3)$