

# Programação Dinâmica

Análise de Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

- 1 Introdução
- 2 Programação Dinâmica
- 3 Problemas



# Sumário

---

## 1 Introdução



# Introdução

---

## Divisão e Conquista

- Combina soluções de subproblemas para construir uma solução maior.
- Se subproblemas se sobrepõem, fazemos mais trabalho que o necessário.
  - ▶ Um problema se sobrepõe a outro quando compartilham subproblemas.



# Introdução

---

## Programação Dinâmica

- Na programação dinâmica, não há necessidade em resolver subproblemas que já foram resolvidos anteriormente.
- Resolvemos cada subproblema uma única vez e guardamos sua solução.
- Geralmente aplicamos esse paradigma em problemas de otimização.
  - ▶ Maior/menor resposta.
- Contudo, nada impede de utilizar as respostas de cada subproblema para resolver o problema de busca associado.



# Introdução

---

## Framework de Programação Dinâmica

- 1 Caracterize a estrutura de uma solução ótima.
- 2 Recursivamente, defina o valor de uma solução ótima.
- 3 Compute o valor da solução ótima.
- 4 Construa a solução ótima de informação já computadas.



# Sumário

---

## 2 Programação Dinâmica



# Programação Dinâmica

---

## Corte de Barras de Aço

- Entrada: comprimento de uma barra de aço (em cm) e uma tabela  $P$  contendo os preços do pedaço por seu tamanho.
- Saída: o maior lucro.





## Corte de Barras de Aço

---

### Exemplo

comprimento $i$	0	1	2	3	4	5	6	7	8	9	10
preço $P_i$	0	1	5	8	9	10	17	17	20	24	30

- Qual a melhor solução para uma barra de tamanho 4?



# Corte de Barras de Aço

---

- Quantos jeitos de cortar a barra nós temos?



# Corte de Barras de Aço

---

- Quantos jeitos de cortar a barra nós temos?
- $2^{n-1}$ .



# Corte de Barras de Aço

---

- Quantos jeitos de cortar a barra nós temos?
- $2^{n-1}$ .
- Necessariamente temos um algoritmo  $\Omega(2^n)$ ?



## Corte de Barras de Aço

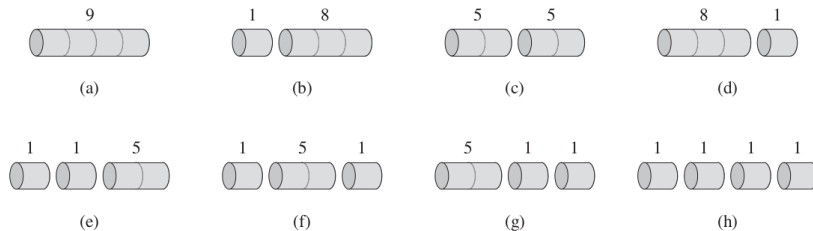


Figura: Maneiras de se cortar uma barra de tamanho 4.



## Corte de Barras de Aço

---

- Suponha um corte ótimo:

$$n = i_1 + i_2 + \dots + i_k$$

- A renda máxima obtida é:

$$r_n = p_{i_1} + p_{i_2} + \dots + p_{i_k}$$

- De maneira geral, temos:

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$



## Cortes de Barro de Aço

---

- $r_1 = 1$  (sem corte).
- $r_2 = 5$  (sem corte).
- $r_3 = 8$  (sem corte).
- $r_4 = 10$  (solução  $r_2 + r_2$ ).
- $r_5 = 13$  (solução  $r_2 + r_3$ ).
- $r_6 = 17$  (sem corte).
- $r_7 = 18$  (solução  $r_1 + r_6$  ou  $r_2 + r_2 + r_3$ ).
- $r_8 = 22$  (solução  $r_2 + r_6$ ).
- $r_9 = 25$  (solução  $r_3 + r_6$ ).
- $r_{10} = 30$  (sem corte).



## Corte de Barras de Aço

---

- Esse problema de subestrutura ótima!.
- Soluções ótimas de subproblemas incorporam soluções ótimas de problemas maiores.
- Outra forma de enxergar a solução: um corte ótimo vai se basear em um pedaço de barra de tamanho  $i$  e na solução do corte de um pedaço de barra de tamanho  $n - i$ .
- Podemos resolvê-lo recursivamente com base na seguinte equação:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

- Estamos assumindo  $r_0 = 0$ .





# Algoritmo Recursivo

---

---

**Algorithm 1:** CUT-ROD( $p, k$ )

---

**Input:**  $P[0, n], k$

**Output:**  $r_k$

```
1 if(  $k == 0$  )
2   return 0
3 else
4    $q \leftarrow -\infty$ 
5   for(  $i \leftarrow 1; i \leq n; i++$  )
6      $q \leftarrow \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
7 return  $q$ 
```

---



# Análise

---

- O custo de pior caso do algoritmo é dado pela seguinte relação de recorrência:

$$T(n) = \begin{cases} \Theta(1), & n \leq 1 \\ \sum_{j=1}^{n-1} T(j) + \Theta(1), & n > 1 \end{cases}$$

- Mostre que  $T(n) = \Omega(2^n)$ .



## Análise

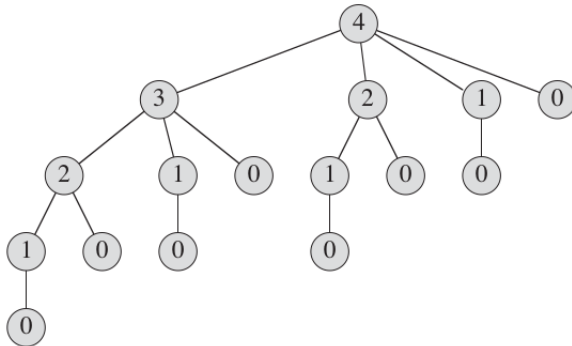


Figura: Árvore de recursão.



# Programação Dinâmica

---

- Qual o problema da solução?



# Programação Dinâmica

---

- Qual o problema da solução?
- Resolve os mesmos problemas várias vezes. . .



# Programação Dinâmica

---

- Qual o problema da solução?
- Resolve os mesmos problemas várias vezes. . .
- A programação dinâmica se encarrega disso ao salvar as soluções dos subproblemas, dessa forma não é necessário recomputá-los.



# Programação Dinâmica

---

- Qual o problema da solução?
- Resolve os mesmos problemas várias vezes. . .
- A programação dinâmica se encarrega disso ao salvar as soluções dos subproblemas, dessa forma não é necessário recomputá-los.
- Trocamos tempo por espaço!



# Programação Dinâmica

---

## Top-Down vs Bottom-Up

- Temos duas abordagens, Top-Down ou Bottom-Up.
- Top-Down: usa a técnica **memoization**. Se baseia na recursão salvando o progresso para cada subproblema resolvido. Soluções já computadas são aproveitadas sem esforço adicional.
- Bottom-Up: resolvemos os problemas menores primeiro para depois resolver os problemas maiores com base nas soluções dos problemas menores.





# Algoritmo Recursivo com Memoization

---

---

## Algorithm 2: INITIALIZATION( $P, R, n$ )

---

**Input:**  $P[0, n], n$

**Output:**  $R[0, n] = (r_0, r_1, \dots, r_n)$ , inicializado

- 1  $R[0] \leftarrow 0$
  - 2 **for**(  $i \leftarrow 1; i \leq n; i++$  )
  - 3      $R[i] \leftarrow -\infty$
-



## Algoritmo Recursivo com Memoization

---

### Algorithm 3: MEMOIZED-CUT-ROD( $P, R, k$ )

---

**Input:**  $P[0, n], R[0, n], k$

**Output:**  $r_k$

```
1 if(  $R[k] \geq 0$  )
2   | return  $R[k]$ 
3 else
4   | for(  $i \leftarrow 1; i \leq k; i++$  )
5     |   |  $R[k] \leftarrow \max(R[k], P[i] + \text{MEMOIZED-CUT-ROD}(P, R, k - i))$ 
6 return  $R[k]$ 
```

---

- Chamada de interesse: MEMOIZED-CUT-ROD( $P, R, k$ ), em que  $k$  é o tamanho da barra inteira.



# Algoritmo Bottom-Up

---

- Podemos implementar a solução de uma outra maneira.
- Computando as soluções de problemas menores para então computar as soluções dos problemas maiores.
- Abordagem *Bottom-Up*.
- Implementação iterativa.



## Algoritmo Bottom-Up

---

---

### Algorithm 4: BOTTOM-UP-CUT-ROD( $P, R$ )

---

**Input:**  $P[0, n], R[0, n], n$

**Output:**  $r_n$

```
1  $R[0] \leftarrow 0$ 
2 for(  $j \leftarrow 1; j \leq n; j++$  )
3    $q \leftarrow -\infty$ 
4   for(  $i \leftarrow 1; i \leq j; i++$  )
5      $q \leftarrow \max(q, P[i] + R[j - i])$ 
6    $R[j] \leftarrow q$ 
7 return  $R[n]$ 
```

---



# Análise

---

- Qual a complexidade da solução usando Programação Dinâmica?



# Análise

---

- Qual a complexidade da solução usando Programação Dinâmica?
- $\Theta(n^2)$ .



# Sumário

---

## 3 Problemas



# Problemas

---

- Apresentaremos alguns problemas que possuem solução eficiente utilizando Programação Dinâmica.
- Perceba que os problemas possuem subproblemas compartilhados.
- Procuraremos elaborar uma solução recursiva e depois elaborar a solução utilizando Programação Dinâmica.





# Sumário

---

## 3 Problemas

- Parentetização Ótima de Matrizes
- Subsequência Comum mais Longa



# Multiplicação de Matrizes

---

**Algorithm 5:** MATRIX-MULTIPLICATION( $A, B$ )

---

**Input:**  $A[0, n_1 - 1][0, m_1 - 1], B[0, n_2 - 1][0, m_2 - 1]$

**Output:**  $C[0, n_1 - 1][0, m_2 - 1]$

```
1 if(  $m_1 \neq n_2$  )
2   | REPORT-ERROR()
3 else
4   | for(  $i \leftarrow 0; i < n_1; i++$  )
5     | for(  $j \leftarrow 0; j < m_2; j++$  )
6       |  $C[i][j] \leftarrow 0$ 
7       | for(  $k \leftarrow 0; k < m_1; k++$  )
8         |  $C[i][j] \leftarrow C[i][j] + A[i][k] \cdot B[k][j]$ 
9 return  $C$ 
```

---



# Multiplicação de Matrizes

---

- Claramente, o algoritmo mostrado é  $\Theta(n^3)$ .
- Só é possível multiplicar duas matrizes que são compatíveis.
- E se quiséssemos multiplicar várias matrizes? Qual a melhor maneira?
- Lembre-se que a multiplicação em matrizes possui a propriedade associativa:

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$



# Multiplicação de Matrizes

---

- Imagine que queremos fazer uma multiplicação de matrizes, e suponha que temos as matrizes  $A_{10 \times 100}^0$ ,  $A_{100 \times 5}^1$  e  $A_{5 \times 50}^2$ .
- Quantas operações são feitas em  $(A^0 \cdot A^1) \cdot A^2$ .
- Quantas operações são feitas em  $A^0 \cdot (A^1 \cdot A^2)$ .
- Qual o melhor jeito de multiplicar as matrizes?



# Parentetização Ótima de Matrizes

---

## Parentetização Ótima de Matrizes

- Entrada: matrizes  $(A_0, \dots, A_{n-1})$ . Cada matriz  $A_i$  tem dimensão  $p_i \times p_{i+1}$ .
- Saída: parentetização ótima de matrizes que minimize o número de operações feitas.



# Parentetização Ótima de Matrizes

---

## Exemplo

Tabela: Exemplo Anterior

$i$	0	1	2	3
$P[i]$	10	100	5	50



# Parentetização Ótima de Matrizes

---

## Contando o Número de Parentetizações

- O número de parentetizações de matrizes é expressa pela seguinte relação de recorrência:

$$T(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} T(k) \cdot T(n-k), & n > 1 \end{cases}$$



# Parentetização Ótima de Matrizes

---

## Contando o Número de Parentetizações

- O número de parentetizações de matrizes é expressa pela seguinte relação de recorrência:

$$T(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} T(k) \cdot T(n-k), & n > 1 \end{cases}$$

- $T(n) \in \Omega\left(\frac{4^n}{n^{3/2}}\right)$





# Parentetização Ótima de Matrizes

---

- Esse problema pode ser resolvido por Programação Dinâmica.
- Vários subproblemas são compartilhados entre os problemas?
- Vamos seguir o framework para desenvolver um algoritmo de PD.
  - 1 Caracterizar a estrutura de uma solução ótima.
  - 2 Recursivamente definir o valor de uma solução ótima.
  - 3 Computar o valor de uma solução ótima.
  - 4 Construir uma solução ótima a partir de valores computados anteriormente.



# Parentetização de Matrizes

---

## Caracterização de uma Solução Ótima

- Buscamos uma parentetização ótima do subproblema envolvendo as submatrizes  $A_i \dots A_j$ .
- Temos que achar o índice  $k$  que divide  $A_i \dots A_k$  de  $A_{k+1} \dots A_j$  de modo que o número de operações entre  $A_{i,k}$  e  $A_{k+1,j}$  seja o menor possível.
- Note que para o resultado final ser ótimo,  $A_{i,k}$  tem que ser uma parentetização ótima, caso contrário, poderíamos substituir para uma melhor e ter um resultado final melhor. O mesmo argumento vale para  $A_{k+1,j}$ .
- Podemos construir uma solução ótima a partir de soluções ótimas dos subproblemas.



# Parentetização de Matrizes

---

## Definição de uma Solução Recursiva

- Seja  $M(i, j)$  o número mínimo de operações para computar o produto de  $A_{i,j}$ .
- $M(i, j)$  é trivial quando  $i = j$ , pois são necessárias 0 operações para computar  $A_{i,i}$  (não há produto).
- Suponha que a parentetização ótima divide  $A_{i,j}$  em produto de  $A_{i,k}$  e  $A_{k+1,j}$



# Parentetização de Matrizes

---

## Definição de uma Solução Recursiva

- Neste caso,  $M(i, j) = M(i, k) + M(k + 1, j) + p_i \cdot p_{k+1} \cdot p_{j+1}$ .
- De maneira genérica, a parentetização ótima é aquela que minimiza o resultado dentre todas as formas de particionamento:

$$M(i, j) = \begin{cases} 0, & i = j \\ \min_{i \leq k < j} \{M(i, k) + M(k + 1, j) + p_i \cdot p_{k+1} \cdot p_{j+1}\}, & j > i \end{cases}$$



# Parentetização de Matrizes

---

## Computação da Solução Ótima

- A partir da solução recursiva, podemos implementar a solução PD.
- Como temos vários problemas que compartilham subproblemas, a PD vai ser bem eficiente.



# Parentetização de Matrizes

---

**Algorithm 6:** MATRIZ-CHAIN-ORDER( $P$ )

---

**Input:**  $P[0, n]$

**Output:**  $M[0, n - 1][0, n - 1], S[0, n - 1][0, n - 1]$

```
1 for(  $i \leftarrow 0; i < n; i++$  )
2    $M[i][i] \leftarrow 0$ 
3 for(  $l \leftarrow 1; l < n; l++$  )
4   for(  $i \leftarrow 0; i < n - l; i++$  )
5      $j \leftarrow i + l$ 
6      $M[i][j] \leftarrow \infty$ 
7     for(  $k \leftarrow i; k < j; k++$  )
8        $q \leftarrow M[i][k] + M[k + 1][j] + P[i] \cdot P[k + 1] \cdot P[j + 1]$ 
9       if(  $q < M[i][j]$  )
10          $M[i][j] \leftarrow q$ 
11          $S[i][j] \leftarrow k$ 
```



## Parentetização de Matrizes

- Vamos rodar o algoritmo para a seguinte entrada:

Matriz	$A_0$	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	
dimensão	$30 \times 35$	$35 \times 15$	$15 \times 5$	$5 \times 10$	$10 \times 20$	$20 \times 25$	
$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0					
1		0				
2			0			
3				0		
4					0	
5						0





# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	●				
1		0				
2			0			
3				0		
4					0	
5						0

$$M[0][1] = \min \{ M[0][0] + M[1][1] + P[0] \cdot P[1] \cdot P[2] = 15750 \}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750				
1		0				
2			0			
3				0		
4					0	
5						0

$$M[1][2] = \min \{ M[1][1] + M[2][2] + P[1] \cdot P[2] \cdot P[3] = 2625 \}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0			
3				0		
4					0	
5						0

$$M[2][3] = \min \{ M[2][2] + M[3][3] + P[2] \cdot P[3] \cdot P[4] = 750 \}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0	750		
3				0		
4					0	
5						0

$$M[3][4] = \min \{ M[3][3] + M[4][4] + P[3] \cdot P[4] \cdot P[5] = 1000 \}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25


	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0	750		
3				0	1000	
4					0	●
5						0

$$M[4][5] = \min \{ M[4][4] + M[5][5] + P[4] \cdot P[5] \cdot P[6] = 5000 \}$$



## Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0	750		
3				0	1000	
4					0	5000
5						0

$$M[0][2] = \min \left\{ \begin{array}{l} M[0][0] + M[1][2] + P[0] \cdot P[1] \cdot P[3] = 7875 \\ M[0][1] + M[2][2] + P[0] \cdot P[2] \cdot P[3] = 18000 \end{array} \right\}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25


	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625			
2			0	750		
3				0	1000	
4					0	5000
5						0

$$M[1][3] = \min \left\{ \begin{array}{l} M[1][1] + M[2][3] + P[1] \cdot P[2] \cdot P[4] = 6000 \\ M[1][2] + M[3][3] + P[0] \cdot P[4] \cdot P[4] = 4375 \end{array} \right\}$$



## Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625	4375		
2			0	750		
3				0	1000	
4					0	5000
5						0

$$M[2][4] = \min \left\{ \begin{array}{l} M[2][2] + M[3][4] + P[2] \cdot P[3] \cdot P[5] = 2500 \\ M[2][3] + M[4][4] + P[2] \cdot P[4] \cdot P[5] = 3750 \end{array} \right\}$$





## Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625	4375		
2			0	750	2500	
3				0	1000	●
4					0	5000
5						0

$$M[3][5] = \min \left\{ \begin{array}{l} M[3][3] + M[4][5] + P[3] \cdot P[4] \cdot P[6] = 6250 \\ M[3][4] + M[5][5] + P[3] \cdot P[5] \cdot P[6] = 3500 \end{array} \right\}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	●		
1		0	2625	4375		
2			0	750	2500	
3				0	1000	3500
4					0	5000
5						0

$$M[0][3] = \min \left\{ \begin{array}{l} M[0][0] + M[1][3] + P[0] \cdot P[1] \cdot P[4] = 14875 \\ M[0][1] + M[2][3] + P[0] \cdot P[2] \cdot P[4] = 21000 \\ M[0][2] + M[3][3] + P[0] \cdot P[3] \cdot P[4] = 9375 \end{array} \right\}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	9375		
1		0	2625	4375		
2			0	750	2500	
3				0	1000	3500
4					0	5000
5						0

$$M[1][4] = \min \left\{ \begin{array}{l} M[1][1] + M[2][4] + P[1] \cdot P[2] \cdot P[5] = 13000 \\ M[1][2] + M[3][4] + P[0] \cdot P[3] \cdot P[5] = 7125 \\ M[1][3] + M[4][4] + P[0] \cdot P[4] \cdot P[5] = 11375 \end{array} \right\}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	9375		
1		0	2625	4375	7125	
2			0	750	2500	●
3				0	1000	3500
4					0	5000
5						0

$$M[2][5] = \min \left\{ \begin{array}{l} M[2][2] + M[3][5] + P[2] \cdot P[3] \cdot P[6] = 5375 \\ M[2][3] + M[4][5] + P[2] \cdot P[4] \cdot P[6] = 9500 \\ M[2][4] + M[5][5] + P[2] \cdot P[5] \cdot P[6] = 10000 \end{array} \right\}$$



## Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	9375	●	
1		0	2625	4375	7125	
2			0	750	2500	5375
3				0	1000	3500
4					0	5000
5						0

$$M[0][4] = \min \left\{ \begin{array}{l} M[0][0] + M[1][4] + P[0] \cdot P[1] \cdot P[5] = 28125 \\ M[0][1] + M[2][4] + P[0] \cdot P[2] \cdot P[5] = 27250 \\ M[0][2] + M[3][4] + P[0] \cdot P[3] \cdot P[5] = 11875 \\ M[0][3] + M[4][4] + P[0] \cdot P[4] \cdot P[5] = 15375 \end{array} \right\}$$



## Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	9375	11875	
1		0	2625	4375	7125	●
2			0	750	2500	5375
3				0	1000	3500
4					0	5000
5						0

$$M[1][5] = \min \left\{ \begin{array}{l} M[1][1] + M[2][5] + P[1] \cdot P[2] \cdot P[6] = 18500 \\ M[1][2] + M[3][5] + P[1] \cdot P[3] \cdot P[6] = 10500 \\ M[1][3] + M[4][5] + P[1] \cdot P[4] \cdot P[6] = 18125 \\ M[1][4] + M[5][5] + P[1] \cdot P[5] \cdot P[6] = 24625 \end{array} \right\}$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	9375	11875	●
1		0	2625	4375	7125	10500
2			0	750	2500	5375
3				0	1000	3500
4					0	5000
5						0

$$M[0][5] = \min \left\{ \begin{array}{l} M[0][0] + M[1][5] + P[0] \cdot P[1] \cdot P[6] = 36750 \\ M[0][1] + M[2][5] + P[0] \cdot P[2] \cdot P[6] = 32375 \\ M[0][2] + M[3][5] + P[0] \cdot P[3] \cdot P[6] = 15125 \\ M[0][3] + M[4][5] + P[0] \cdot P[4] \cdot P[6] = 21875 \\ M[0][4] + M[5][5] + P[0] \cdot P[5] \cdot P[6] = 26875 \end{array} \right.$$



# Parentetização de Matrizes

$i$	0	1	2	3	4	5	6
$P$	30	35	15	5	10	20	25

	0	1	2	3	4	5
0	0	15750	7875	9375	11875	15125
1		0	2625	4375	7125	10500
2			0	750	2500	5375
3				0	1000	3500
4					0	5000
5						0

$$M[0][5] = \min \left\{ \begin{array}{l} M[0][0] + M[1][5] + P[0] \cdot P[1] \cdot P[6] = 36750 \\ M[0][1] + M[2][5] + P[0] \cdot P[2] \cdot P[6] = 32375 \\ M[0][2] + M[3][5] + P[0] \cdot P[3] \cdot P[6] = 15125 \\ M[0][3] + M[4][5] + P[0] \cdot P[4] \cdot P[6] = 21875 \\ M[0][4] + M[5][5] + P[0] \cdot P[5] \cdot P[6] = 26875 \end{array} \right.$$





## Parentetização de Matrizes

---

- Qual a complexidade do algoritmo?
- Está claro que é  $O(n^3)$ , pois temos que preencher a metade superior de uma matriz quadrada levando tempo  $O(n)$  para cada célula.
- Será que também é  $\Omega(n^3)$  e portanto  $\Theta(n^3)$ ?



# Parentetização de Matrizes

---

Queremos mostrar que

$$\sum_{l=1}^{n-1} \sum_{i=0}^{n-l-1} \sum_{k=0}^{l-1} \Theta(1) \in \Omega(n^3)$$



## Parentetização de Matrizes

---

$$\begin{aligned} & \sum_{l=1}^{n-1} \sum_{i=0}^{n-l-1} \sum_{k=0}^{l-1} \Theta(1) \\ & \geq d \sum_{l=1}^{n-1} \sum_{i=0}^{n-l-1} \sum_{k=0}^{l-1} 1 \\ & = d \sum_{l=1}^{n-1} \sum_{i=0}^{n-l-1} l \\ & = d \sum_{l=1}^{n-1} nl - l^2 \end{aligned}$$



## Parentetização de Matrizes

---

$$\begin{aligned} &= d \left( \sum_{l=1}^{n-1} nl - \sum_{l=1}^{n-1} l^2 \right) \\ &= d \left( n \sum_{l=1}^{n-1} l - \sum_{l=1}^{n-1} l^2 \right) \\ &= d \left( \frac{n(n^2 - n)}{2} - \sum_{l=1}^{n-1} l^2 \right) \\ &= d \left( \frac{n^3 - n^2}{2} - \sum_{l=1}^{n-1} l^2 \right) \end{aligned}$$



## Parentetização de Matrizes

---

$$\begin{aligned} & d \left( \frac{n^3 - n^2}{2} - \sum_{l=1}^{n-1} l^2 \right) \\ &= d \left( \frac{n^3 - n^2}{2} - \frac{(n-1)(n)(2n)}{6} \right) \\ &= d \left( \frac{n^3 - n^2}{2} - \frac{2n^3 - 2n^2}{6} \right) \\ &= d \left( \frac{3n^3 - 3n^2 - 2n^3 + 2n^2}{6} \right) \\ &= d \left( \frac{n^3 - n^2}{6} \right) = \frac{dn^3 - dn^2}{6} \geq cn^3, \quad \diamond \quad 0 < c < \frac{d}{6} \text{ e } n \text{ s.f.g} \end{aligned}$$



# Parentetização de Matrizes

---

## Construindo a Solução Ótima

- A matriz  $S$  guarda o índice  $k$  que torna a parentetização ótima para cada subproblema.
- Podemos utilizar  $S$  para construir a melhor parentetização.
- Um problema de otimização vira agora um problema de busca.



## Parentetização de Matrizes

---

---

**Algorithm 7:** PRINT-OPTIMAL-PARENS( $S, i, j$ )

---

**Input:**  $S[0, n - 1][0, n - 1], i, j$

**Output:** Parentetização ótima de  $A_{i,j}$

```
1 if(  $i = j$  )
2   | PRINT( " $A_i$ " )
3 else
4   | PRINT( "(" )
5   | PRINT-OPTIMAL-PARENS( $S, i, S[i, j]$ )
6   | PRINT-OPTIMAL-PARENS( $S, S[i, j] + 1, j$ )
7   | PRINT( ")" )
```

---



# Sumário

---

## 3 Problemas

- Parentetização Ótima de Matrizes
- Subsequência Comum mais Longa





# LCS

---

## Definição (Subsequência)

Dada uma sequência  $X = x_0x_1 \dots x_{n-1}$  e outra sequência  $Z = z_0 \dots z_{m-1}$ , dizemos que  $Z$  é subsequência de  $X$ , se existe uma sequência crescente de índices  $(i_0, i_1, \dots, i_{m-1})$  tal que, para todo  $k$ ,  $0 \leq k < m$ ,  $X_{i_k} = Z_k$ .

## Exemplo

$Z = BCDB$  é subsequência de  $X = ABCBDAB$ .



# LCS

---

## Definição (Subsequência Comum)

Dadas duas sequências  $X$  e  $Y$ ,  $Z$  é uma subsequência comum de  $X$  e  $Y$  se:

- 1  $Z$  é subsequência de  $X$ .
- 2  $Z$  é subsequência de  $Y$ .



# LCS

---

## Definição (Longest Common Subsequence)

- Entrada: duas sequências  $X$  e  $Y$ .
- Saída: maior subsequência comum de  $X$  e  $Y$  (LCS).

## Exemplo

- Entrada:  $X = ABCBDAB$  e  $Y = BDCABA$ .
- Saída:  $Z = BCBA$  ou  $Z = BDAB$ .
- Não existe subsequência comum de tamanho  $\geq 5$ .



# LCS

---

## Solução Força-Bruta

- Gere todas as subsequências de  $X$ .
- Para cada subsequência gerada a partir de  $X$ , verifique se ela também é uma subsequência de  $Y$ .
- Armazene a maior subsequência comum encontrada.
- Inviável:  $2^n$  subsequências de  $X$ .



# LCS

---

## Caracterizando uma LCS

- Seja  $X = x_0 \dots x_{n-1}$  e  $Y = y_0 \dots y_{m-1}$  sequências. Tome  $Z = z_0 \dots z_{k-1}$  a maior subsequência comum entre  $X$  e  $Y$ .
  - ① Se  $x_{n-1} = y_{m-1}$ , então  $z_{k-1} = x_{n-1} = y_{m-1}$ .  $Z[0, k-2]$  tem que ser uma LCS de  $X[0, n-2]$  e  $Y[0, m-2]$ .
  - ② Se  $x_{n-1} \neq y_{m-1}$ , e  $z_{k-1} \neq x_{n-1}$ , então  $Z$  é uma LCS de  $X[0, n-2]$  e  $Y$ .
  - ③ Se  $x_{n-1} \neq y_{m-1}$ , e  $z_{k-1} \neq y_{m-1}$ , então  $Z$  é uma LCS de  $X$  e  $Y[0, m-2]$ .



# LCS

---

## Caracterizando uma LCS

- É fácil ver que a caracterização possui subestrutura ótima.



# LCS

---

## Solução Recursiva

- Seja  $C(i, j)$  o tamanho da maior subsequência entre  $X[0, i - 1]$  e  $Y[0, j - 1]$ .
- Logo:

$$C(i, j) = \begin{cases} 0, & i = 0 \vee j = 0 \\ C(i - 1, j - 1) + 1, & i, j > 0 \wedge x_{i-1} = y_{j-1} \\ \max(C(i - 1, j), C(i, j - 1)), & i, j > 0 \wedge x_{i-1} \neq y_{j-1} \end{cases}$$

- Podemos ver que existem várias sobreposições de subproblemas:
  - ▶ LCS de  $X[0, n - 1]$  e  $Y[0, m - 2]$  e LCS de  $X[0, n - 2]$  e  $Y[0, m - 2]$ .



# LCS

---

## Solução Bottom-Up

- A partir da solução recursiva, podemos elaborar uma solução PD bottom-up.
- Essa solução resolve subproblemas menores para assim, resolver os subproblemas maiores.





# LCS

---

## Algorithm 8: LCS( $X, Y$ )

---

**Input:**  $X[0, n - 1], Y[0, m - 1]$

**Output:**  $C[0, n - 1][0, m - 1], D[0, n - 1][0, m - 1]$

```
1  $C[i][0] \leftarrow 0, \quad 0 \leq i \leq n$ 
2  $C[0][j] \leftarrow 0, \quad 0 \leq j \leq m$ 
3 for(  $i \leftarrow 1; i \leq n; i++$  )
4   for(  $j \leftarrow 1; j \leq m; j++$  )
5     if(  $X[i - 1] = Y[j - 1]$  )
6        $C[i][j] \leftarrow C[i - 1][j - 1] + 1$ 
7        $D[i][j] \leftarrow '\swarrow'$ 
8     else if(  $C[i - 1][j] \geq C[i][j - 1]$  )
9        $C[i][j] \leftarrow C[i - 1][j]$ 
10       $D[i][j] \leftarrow '\uparrow'$ 
11    else
12       $C[i][j] \leftarrow C[i][j - 1]$ 
13       $D[i][j] \leftarrow '\leftarrow'$ 
```

---



# LCS

---

## Recuperando a LCS

- Para recuperar a LCS, basta seguir a matriz de direções  $D$ .



# LCS

$y_j$	<b>B</b>	D	<b>C</b>	A	<b>B</b>	<b>A</b>
$x_i$	0	0	0	0	0	0
A	0	↑ 0	↑ 0	↖ 0	← 1	↖ 1
<b>B</b>	0	↖ <b>1</b>	← 1	← 1	↑ 1	↖ 2
<b>C</b>	0	↑ 1	↑ 1	↖ <b>2</b>	← 2	↑ 2
<b>B</b>	0	↖ 1	↑ 1	↑ 2	↖ <b>3</b>	← 3
D	0	↑ 1	↖ 2	↑ 2	↑ 3	↑ 3
<b>A</b>	0	↑ 1	↑ 2	↖ 3	↑ 3	↖ <b>4</b>
B	0	↖ 1	↑ 2	↑ 3	↖ 4	↑ 4

Figura: LCS entre  $X$  e  $Y$ .



# LCS

---

## Análise

- Qual a complexidade do Algoritmo?



# LCS

---

## Análise

- Qual a complexidade do Algoritmo?
- $\Theta(n \cdot m)$ .