

Programação dinâmica.

CLRS Cap. 15

Instituto Superior Técnico

2022/2023

Resumo

Programação dinâmica

Problema da Mochila (Knapsack)

Maior Sub-Sequência Comum (LCS)

Realizar Trocos

Maior Palíndromo

Multiplicação de Cadeias de Matrizes

Contexto

- Revisão [CLRS, Cap.1-13]
 - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
 - Programação dinâmica [CLRS, Cap.15]
 - Algoritmos greedy [CLRS, Cap.16]
- Algoritmos em Grafos [CLRS, Cap.21-26]
 - Algoritmos elementares
 - Caminhos mais curtos [CLRS, Cap.22,24-25]
 - Fluxos máximos [CLRS, Cap.26]
 - Árvores abrangentes [CLRS, Cap.23]
- Programação Linear [CLRS, Cap.29]
 - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
 - Complexidade Computacional

Técnicas para Síntese de Algoritmos

Técnicas para Síntese de Algoritmos

- Dividir para conquistar
 - Exemplo: MergeSort
- Programação dinâmica
 - Exemplo: Floyd-Warshall
- Algoritmos greedy
 - Exemplo: Prim, Dijkstra

Programação Dinâmica

Passos para a realização de um algoritmo baseado em programação dinâmica:

- Caracterizar **estrutura** de uma solução óptima
- Definir **recursivamente** o valor de uma solução óptima
- Calcular valor da solução óptima utilizando abordagem **bottom-up**
- Construir **solução** a partir da informação obtida

Exemplo

Calcular o n -ésimo elemento da sequência de Fibonacci, sabendo que:

$$fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{caso contrário} \end{cases}$$

Solução Recursiva

fib(n)

```

if  $n \leq 1$  then
  return  $n$ 
else
  return  $fib(n-1) + fib(n-2)$ 
end if

```

Complexidade: $O(2^n)$

Solução Programação Dinâmica

fib(n)

```

if  $n \leq 1$  then
  return  $n$ 
else
  Alocar vetor  $f$  com  $n + 1$  posições
   $f[0] = 0$ 
   $f[1] = 1$ 
   $i = 2$ 
  while  $i \leq n$  do
     $f[i] = f[i-1] + f[i-2]$ 
     $i = i + 1$ 
  end while
  return  $f[n]$ 
end if

```

Complexidade: $\Theta(n)$

Exemplo

Calcular as combinações de n , k a k

$$C_k^n = \begin{cases} 1 & \text{se } k = 0 \text{ ou } k = n \\ C_{k-1}^{n-1} + C_k^{n-1} & \text{se } 0 < k < n \\ 0 & \text{caso contrário} \end{cases}$$

Solução Recursiva

$C(n, k)$

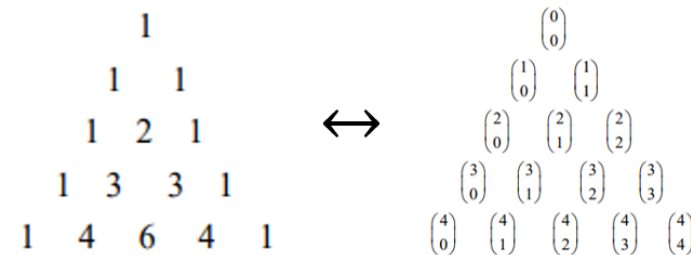
```

if  $k = 0$  or  $k = n$  then
    return 1
else
    return  $C(n - 1, k - 1) + C(n - 1, k)$ 
end if
    
```

- Cada chamada a $C(n, k)$ retorna 1 ou invoca o cálculo de dois sub-problemas
- Solução é calculada somando 1's !
- Tempo de execução é: $\Omega(C_k^n)$

Exemplo

- Número de $C(n, k)$ distintos é apenas $n \times k$
- Complexidade da solução recursiva deriva do **cálculo repetido** de sub-problemas
 - $C(5, 3) = C(4, 2) + C(4, 3)$
 - $C(4, 2) = C(3, 1) + C(3, 2)$
 - $C(4, 3) = C(3, 2) + C(3, 3)$
- Solução: solução construtiva (*bottom-up*)
 - Preencher tabela ($n \times k$) (triângulo de Pascal)



Características da Programação Dinâmica

- Solução ótima do problema composta por soluções ótimas para **sub-problemas**
- Solução recursiva resolve repetidamente os mesmos sub-problemas
 - Sobreposição de problemas
 - Possível usar solução construtiva para evitar resolver repetidamente o mesmo problema
- Memorização (**Memoization**)

Abordagem

- Caracterizar a estrutura da solução ótima
- Definir recursivamente o valor da solução ótima
- Calcular o valor da solução ótima (*bottom-up*)
- Construir a solução ótima

Definição

- Dados n objetos $(1, \dots, n)$ e uma mochila
- Cada objeto tem um valor v_i e um peso w_i
- Peso transportado pela mochila não pode exceder W
- Objetivo: maximizar o valor transportado pela mochila e respeitar a restrição de peso

Formalização

- $x_i = 1$ se objeto i incluído na mochila; 0 caso contrário

$$\begin{aligned} &\text{maximizar} && \sum_{i=1}^n v_i x_i \\ &\text{sujeito a} && \sum_{i=1}^n w_i x_i \leq W \\ &&& x_i \in \{0, 1\} \end{aligned}$$

Solução

- $v[i, j]$: máximo valor que é possível transportar se:
 - se o peso limite é j , ($j \leq W$)
 - e se apenas podem ser seleccionados os objetos numerados de 1 a i

- Solução óptima encontra-se em $v[n, W]$

- Definição:

$$v[i, j] = \max(v[i-1, j], v[i-1, j-w_i] + v_i)$$

$$v[0, j] = 0, \quad j \geq 0$$

$$v[i, j] = -\infty, \quad j < 0$$

$$v[i, 0] = 0$$

Tentativa

- Algoritmo que a cada passo selecciona objeto com maior valor v_i/w_i
- Problema:
 - $v_1 = 8, w_1 = 6$
 - $v_2 = 5, w_2 = 5$
 - $v_3 = 5, w_3 = 5$
 - $W = 10$
- Primeiro objeto seleccionado (objeto 1) impede encontrar solução óptima (com objetos 2 e 3)

Análise

- Solução óptima é composta por soluções óptimas para os sub-problemas:

$$v[i, j] = \max(v[i-1, j], v[i-1, j-w_i] + v_i)$$

- Se $v[i, j]$ é solução óptima:
 - Se objeto i não é incluído, $v[i-1, j]$ é sub-solução óptima
 - Se objeto i é incluído, $v[i-1, j-w_i]$ é sub-solução óptima

- Caso contrário: seria possível obter solução com valor superior ao da solução óptima; uma contradição !

Análise

- Solução recursiva tem tempo de execução exponencial em n e W
 - Mas: número de sub-problemas distintos é $n \times W$
 - Conclusão: resolução repetida dos mesmos sub-problemas !
- Utilizar solução construtiva (*bottom-up*)
 - preencher tabela ($n \times W$)

Complexidade: $\Theta(nW)$

- pseudo-polinomial (depende do valor, não do tamanho do input)

Definição

- Dados n objetos $(1, \dots, n)$ e uma mochila
- Cada objeto tem um valor v_i e um peso w_i
- Peso transportado pela mochila não pode exceder W
- Objetivo: determinar os objetos que devem ser colocados na mochila por forma a maximizar o valor transportado pela mochila e respeitar a restrição de peso

Formalização

- $x_i = k$ se objeto incluído k vezes na mochila; 0 caso contrário

$$\begin{array}{ll} \text{maximizar} & \sum_{i=1}^n v_i x_i \\ \text{sujeito a} & \sum_{i=1}^n w_i x_i \leq W \\ & x_i \in \mathbb{N} \end{array}$$

Solução

- $v[j]$: máximo valor que é possível transportar se:
 - se o peso limite é j , ($j \leq W$)
- Solução ótima encontra-se em $v[W]$
- Definição:

$$v[j] = \max(0, \max_{1 \leq i \leq n} \{v[j - w_i] + v_i\}), \quad j \leq W$$

$$v[0] = 0$$

$$v[j] = -\infty, \quad j < 0$$

Definição

- Dada uma sequência $X = \langle x_1, \dots, x_n \rangle$, uma sequência $Z = \langle z_1, \dots, z_k \rangle$ é uma sub-sequência de X se existe uma sequência estritamente crescente $\langle i_1, \dots, i_k \rangle$ tal que para todo o $j = 1, \dots, k$, $x_{i_j} = z_j$
- Dadas as sequências $X = \langle x_1, \dots, x_n \rangle$ e $Y = \langle y_1, \dots, y_m \rangle$, Z é uma sub-sequência comum se Z é sub-sequência de X e de Y
Obs: $X_i = \langle x_1, \dots, x_i \rangle$
- Objetivo: encontrar sub-sequência comum de maior comprimento (LCS) entre duas sequências X e Y

Exemplo

- Um caso concreto:
 - $X = \langle abefcghd \rangle$
 - $Y = \langle eagbdfh \rangle$
 - $Z = \langle abcd \rangle$ é sub-sequência comum de X e Y
- Uma solução exaustiva é impraticável:
 - Considerar inclusão (ou não) de cada carater de X e de Y
 - Total de sub-sequências em X : 2^n
 - Total de sub-sequências em Y : 2^m
 - Total de casos a analisar: 2^{n+m}
 - Impraticável para valores elevados de n e m

Alguns Resultados

- Sejam $X = \langle x_1, \dots, x_n \rangle$ e $Y = \langle y_1, \dots, y_m \rangle$ duas sequências, e seja $Z = \langle z_1, \dots, z_k \rangle$ uma LCS de X e Y
- Se $x_n = y_m$, então $z_k = x_n = y_m$ e Z_{k-1} é LCS de X_{n-1} e Y_{m-1}
- Se $x_n \neq y_m$, então:
 - se $z_k \neq x_n$ implica que Z é LCS de X_{n-1} e Y
 - se $z_k \neq y_m$ implica que Z é LCS de X e Y_{m-1}

Abordagem

- Se $x_n = y_m$, encontrar LCS W de X_{n-1} e Y_{m-1}
 - Adicionar $x_n = y_m$ a W permite obter Z
- Se $x_n \neq y_m$, encontrar LCS' s para X_{n-1} e Y e para X e Y_{m-1}
 - Escolher a maior LCS

Abordagem

- Se $x_n = y_m$, encontrar LCS W de X_{n-1} e Y_{m-1}
 - Adicionar $x_n = y_m$ a W permite obter Z
- Se $x_n \neq y_m$, encontrar LCS' s para X_{n-1} e Y e para X e Y_{m-1}
 - Escolher a maior LCS

Formulação

- $c[i, j]$: comprimento da LCS para as sequências X_i e Y_j

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

Tempo de execução: $O(nm)$

Exemplo

	-	a	b	e	f	c	g	h	d
-									
e									
a									
g									
b									
c									
f									
d									
h									

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0								
a	0								
g	0								
b	0								
c	0								
f	0								
d	0								
h	0								

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0							
a	0								
g	0								
b	0								
c	0								
f	0								
d	0								
h	0								

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0							
a	0	1							
g	0								
b	0								
c	0								
f	0								
d	0								
h	0								

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0							
a	0	1							
g	0	1							
b	0								
c	0								
f	0								
d	0								
h	0								

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0							
a	0	1							
g	0	1							
b	0	1							
c	0								
f	0								
d	0								
h	0								

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0							
a	0	1							
g	0	1							
b	0	1							
c	0	1							
f	0	1							
d	0	1							
h	0	1							

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0						
a	0	1							
g	0	1							
b	0	1							
c	0	1							
f	0	1							
d	0	1							
h	0	1							

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0						
a	0	1	1						
g	0	1							
b	0	1							
c	0	1							
f	0	1							
d	0	1							
h	0	1							

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0						
a	0	1	1						
g	0	1	1						
b	0	1	2						
c	0	1							
f	0	1							
d	0	1							
h	0	1							

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0						
a	0	1	1						
g	0	1	1						
b	0	1	2						
c	0	1	2						
f	0	1	2						
d	0	1	2						
h	0	1	2						

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0	1					
a	0	1	1	1					
g	0	1	1	1					
b	0	1	2	2					
c	0	1	2	2					
f	0	1	2	2					
d	0	1	2	2					
h	0	1	2	2					

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0	1	1				
a	0	1	1	1	1				
g	0	1	1	1	1				
b	0	1	2	2	2				
c	0	1	2	2	2				
f	0	1	2	2					
d	0	1	2	2					
h	0	1	2	2					

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0	1	1				
a	0	1	1	1	1				
g	0	1	1	1	1				
b	0	1	2	2	2				
c	0	1	2	2	2				
f	0	1	2	2	3				
d	0	1	2	2					
h	0	1	2	2					

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0	1	1	1	1	1	1
a	0	1	1	1	1	1	1	1	1
g	0	1	1	1	1	1	2	2	2
b	0	1	2	2	2	2	2	2	2
c	0	1	2	2	2	3	3	3	3
f	0	1	2	2	3	3	3	3	3
d	0	1	2	2	3	3	3	3	4
h	0	1	2	2	3	3	3	4	4

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0	1	1	1	1	1	1
a	0	1	1	1	1	1	1	1	1
g	0	1	1	1	1	1	2	2	2
b	0	1	2	2	2	2	2	2	2
c	0	1	2	2	2	3	3	3	3
f	0	1	2	2	3	3	3	3	3
d	0	1	2	2	3	3	3	3	4
h	0	1	2	2	3	3	3	4	4

LCS: **abch** (de tamanho 4)

Exemplo

	-	a	b	e	f	c	g	h	d
-	0	0	0	0	0	0	0	0	0
e	0	0	0	1	1	1	1	1	1
a	0	1	1	1	1	1	1	1	1
g	0	1	1	1	1	1	2	2	2
b	0	1	2	2	2	2	2	2	2
c	0	1	2	2	2	3	3	3	3
f	0	1	2	2	3	3	3	3	3
d	0	1	2	2	3	3	3	3	4
h	0	1	2	2	3	3	3	4	4

LCS: **abcd** (de tamanho 4)

Definição

- Dado um conjunto de moedas, denominadas $1, \dots, n$, com valores d_1, \dots, d_n , calcular o **menor número de moedas** cuja soma de valores é T
 - Número ilimitado de moedas de cada denominação
- Solução greedy pode não funcionar:
 - $d_1 = 1; d_2 = 5; d_3 = 20; d_4 = 25$
 - Troco de 40?!
- Solução baseada em programação dinâmica

Definição

- Dada uma sequência $X = \langle x_1, \dots, x_n \rangle$, calcular a **maior sub-sequência** $Z = \langle z_1, \dots, z_k \rangle$ de X tal que Z seja um palíndromo

Exemplo

- $X = \langle abcfdbca \rangle$
- $Z = \langle abdba \rangle$

Formulação

- $c[i, j]$: menor número de moedas necessárias para pagar j unidades, $0 \leq j \leq T$, utilizando apenas moedas com denominação entre 1 e i , $1 \leq i \leq n$
- Objetivo é calcular $c[n, T]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > 0 \text{ e } j = 0 \\ +\infty & \text{se } i = 0 \text{ ou } j < 0 \\ \min(c[i-1, j], c[i, j-d_i] + 1) & \text{caso contrário} \end{cases}$$

Tempo de execução: $O(n T)$

Formulação

- $l[i, j]$: tamanho da maior sub-sequência palíndromo de X (entre os índices i e j , tal que $j \geq i$)

$$l[i, j] = \begin{cases} 1 & i = j \\ l[i+1, j-1] + 2 & \text{se } x_i = x_j \\ \max(l[i, j-1], l[i+1, j]) & \text{caso contrário} \end{cases}$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1							
2		1						
3			1					
4				1				
5					1			
6						1		
7							1	
8								1

$$l[i, j] = 1, \quad i = j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1						
2		1						
3			1					
4				1				
5					1			
6						1		
7							1	
8								1

$$\max(l[i, j - 1], l[i + 1, j]), \quad x_i \neq x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1					
2		1	1					
3			1	1				
4				1	1			
5					1	1		
6						1	1	
7							1	1
8								1

$$\max(l[i, j - 1], l[i + 1, j]), \quad x_i \neq x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1	1				
2		1	1	1	1			
3			1	1	1	1		
4				1	1	1	1	
5					1	1	1	1
6						1	1	1
7							1	1
8								1

$$\max(l[i, j - 1], l[i + 1, j]), \quad x_i \neq x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1	1	1			
2		1	1	1	1			
3			1	1	1	1		
4				1	1	1	1	
5					1	1	1	1
6						1	1	1
7							1	1
8								1

$$\max(l[i, j-1], l[i+1, j]), \quad x_i \neq x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1	1	1			
2		1	1	1	1	3		
3			1	1	1	1	3	
4				1	1	1	1	1
5					1	1	1	1
6						1	1	1
7							1	1
8								1

$$l[i+1, j-1] + 2, \quad x_i = x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	3	
2		1	1	1	1	3	3	
3			1	1	1	1	3	3
4				1	1	1	1	1
5					1	1	1	1
6						1	1	1
7							1	1
8								1

$$\max(l[i, j-1], l[i+1, j]), \quad x_i \neq x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1	1	1	3	3	
2		1	1	1	1	3	3	3
3			1	1	1	1	3	3
4				1	1	1	1	1
5					1	1	1	1
6						1	1	1
7							1	1
8								1

$$\max(l[i, j-1], l[i+1, j]), \quad x_i \neq x_j$$

Exemplo

i	1	2	3	4	5	6	7	8
X_i	a	b	c	f	d	b	c	a

$i \setminus j$	1	2	3	4	5	6	7	8
1	1	1	1	1	1	3	3	5
2		1	1	1	1	3	3	3
3			1	1	1	1	3	3
4				1	1	1	1	1
5					1	1	1	1
6						1	1	1
7							1	1
8								1

$$l[i+1, j-1] + 2, \quad x_i = x_j$$

Exemplo

- Calcular $A(13 \times 5) \times B(5 \times 89) \times C(89 \times 3) \times D(3 \times 34)$
- $((A \times B) \times C) \times D$:
 - $13 \times 5 \times 89 + 13 \times 89 \times 3 + 13 \times 3 \times 34 = 10.582$ produtos
- $((A \times B) \times (C \times D))$:
 - $13 \times 5 \times 89 + 89 \times 3 \times 34 + 13 \times 89 \times 34 = 54.201$ produtos
- $((A \times (B \times C)) \times D)$:
 - $5 \times 89 \times 3 + 13 \times 5 \times 3 + 13 \times 3 \times 34 = 2.856$ produtos !
- $(A \times ((B \times C) \times D))$:
 - $5 \times 89 \times 3 + 5 \times 3 \times 34 + 13 \times 5 \times 34 = 4.055$ produtos
- $(A \times (B \times (C \times D)))$:
 - $89 \times 3 \times 34 + 5 \times 89 \times 34 + 13 \times 5 \times 34 = 26.418$ produtos

Definição

- Dadas matrizes A_1, A_2, \dots, A_n , tal que A_i tem dimensões $(l_i \times c_i)$
- Objetivo: colocar parêntesis na cadeia de produtos de matrizes $A_1 \times A_2 \times \dots \times A_n$, tal que o número de multiplicações escalares é minimizado

Observações

- Tempo para multiplicar as n matrizes é dominado pelo tempo para realizar as multiplicações escalares necessárias
 - Para multiplicar duas matrizes $(r \times s)$ e $(s \times t)$, o número de multiplicações escalares é: $r \times s \times t$
- Número de produtos depende do modo como os produtos de matrizes são organizados
 - Colocação de parêntesis define organização da multiplicação de matrizes

- Número de colocações possíveis de parêntesis cresce exponencialmente com número de matrizes:

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{k=1}^{n-1} P(k) P(n-k) & \text{se } n \geq 2 \end{cases}$$

$$P(n) = C(n-1)$$

$$C(n) = \frac{C_n^{2n}}{n+1} = \Omega(4^n/n^{3/2})$$

Características da Solução Ótima

Seja $A_{1\dots n}$ solução com colocação ótima de parêntesis

- Admitir solução ótima com parêntesis em k , $A_{1\dots k}A_{k+1\dots n}$
- Facto:
 - Colocação de parêntesis para $A_{1\dots k}$ é também ótima
- Porquê?
 - Caso contrário seria possível encontrar uma melhor colocação de parêntesis para $A_{1\dots k}$ e portanto para $A_{1\dots n}$
- Conclusão:
 - Solução ótima para o problema da colocação de parêntesis é composta por soluções ótimas para os seus sub-problemas

Solução Recursiva

$$m[i, j] = \begin{cases} 0 & \text{se } i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & \text{se } i < j \end{cases}$$

$$s[i, j] = k \text{ sse } m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$$

Solução Recursiva

- $m[i, j]$: menor número de multiplicações escalares necessário para calcular multiplicação cadeias matrizes $A_{i\dots j}$
- Solução ótima para $A_{1\dots n}$ é $m[1, n]$
- $i = j$: $m[i, j] = 0$
- $i < j$:
 - Admitir que solução ótima coloca parêntesis em k :
 - ▶ $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$
 - Mas qual é o valor de k ?
 - ▶ certamente k tem valor entre i e $j - 1$
 - ▶ considerar todos os valores de k possíveis
- $s[i, j]$: define colocação ótima de parêntesis entre i e j

Cálculo dos valores de $m[i, j]$

- Número de sub-problemas distintos:
 - 1 para cada $1 \leq i \leq j \leq n$
 - número de problemas: $\Theta(n^2)$
- Problema:
 - solução recursiva requer tempo exponencial
 - resolução repetida dos mesmos subproblemas
- Solução:
 - solução construtiva (*bottom-up*)
 - tempo de execução: $O(n^3)$

- Permite obter tempo de execução das soluções dos sub-problemas, mas utilizando abordagem recursiva (*top-down*)
 - É necessário **memorizar** resultados de sub-problemas já resolvidos
- Exemplo: caminhos mais curtos num DAG, com DFS
- Exemplo: cálculo das combinações
 - Não calcular todo o triângulo de Pascal
 - Calcular apenas as entradas necessárias
 - Calcular cada entrada apenas 1 vez