Programação Dinâmica

Prof. Anderson Almeida Ferreira

Adaptado do material elaborado por

Andrea Iabrudi Tavares

Programação Dinâmica

- 1950, Bellman
- Evitar recálculos dos subproblemas em comum
  - Menor para maior (bottom-up)
  - Tabelas ou memoização
- É uma técnica de programação
- Foi desenvolvida na época em que "programação" significava "método tabular".
- Usada para problemas de otimização
  - Encontre a solução *a* com o valor ótimo.
  - Minimizar ou maximizar

Programação Dinâmica

- **Quatro passos do método**
  - Caracterize a estrutura de uma solução ótima.
  - Recursivamente defina o valor de uma solução ótima.
  - Compute o valor de uma solução ótima de maneira bottom-up.
  - Construa a solução ótima por meio da informação computada.

Fibonacci: definindo recorrência

```
RecFibo(n):
    if (n < 2)
        return n
    else
        return RecFibo(n − 1) + RecFibo(n − 2)
```

- **Grafo de recorrência**
  - Subproblemas – nós
  - Dependência – arestas

Fibonacci: definindo recursão

```
RecFibo(n):
  if (n < 2)
      return n
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

```
MemFibo(n):
  if (n < 2)
      return n
  else
      if F[n] is undefined
          F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
      return F[n]
```

- Grafo de recorrência
  - Subproblemas – nós
  - Dependência – arestas
- **Memoização**

Fibonacci: definindo recursão

```
RecFibo(n):
  if (n < 2)
      return n
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

```
MemFibo(n):
  if (n < 2)
      return n
  else
      if F[n] is undefined
          F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
      return F[n]
```

- Grafo de recorrência
  - Subproblemas – nós
  - Dependência – arestas
- **Memoização**

Fibonacci: usando tabela

```
RecFibo(n):
  if (n < 2)
      return n
  else
      return RecFibo(n − 1) + RecFibo(n − 2)

MemFibo(n):
  if (n < 2)
      return n
  else
      if F[n] is undefined
          F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
      return F[n]

IterFibo(n):
  F[0] ← 0
  F[1] ← 1
  for i ← 2 to n
      F[i] ← F[i − 1] + F[i − 2]
  return F[n]
```

- Grafo de recorrência
  - Subproblemas – nós
  - Dependência – arestas
- Memoização

Fibonacci: usando tabela

```
RecFibo(n):
    if (n < 2)
        return n
    else
        return RecFibo(n − 1) + RecFibo(n − 2)

MemFibo(n):
    if (n < 2)
        return n
    else
        if F[n] is undefined
            F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
        return F[n]

IterFibo(n):
    F[0] ← 0
    F[1] ← 1
    for i ← 2 to n
        F[i] ← F[i − 1] + F[i − 2]
    return F[n]
```

- Grafo de recorrência
  - Subproblemas – nós
  - Dependência – arestas
- Memoização
- **Tabela**
  - Ordenação parcial

Fibonacci: economizando espaço

```
RecFibo(n):
  if (n < 2)
      return n
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

```
MemFibo(n):
  if (n < 2)
      return n
  else
      if F[n] is undefined
          F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
      return F[n]
```

```
IterFibo(n):
  F[0] ← 0
  F[1] ← 1
  for i ← 2 to n
      F[i] ← F[i − 1] + F[i − 2]
  return F[n]
```

```
IterFibo2(n):
  prev ← 1
  curr ← 0
  for i ← 1 to n
      next ← curr + prev
      prev ← curr
      curr ← next
  return curr
```

- Grafo de recorrência
  - Subproblemas – nós
  - Dependência – arestas
- Memoização

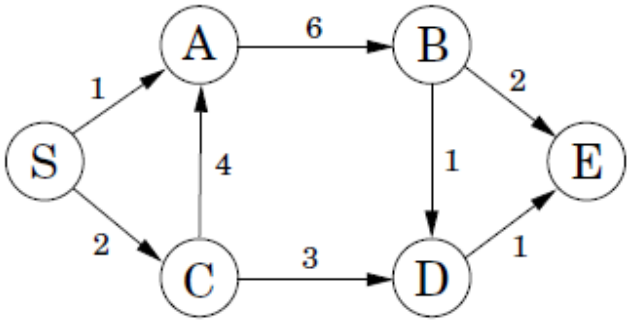
Tabela

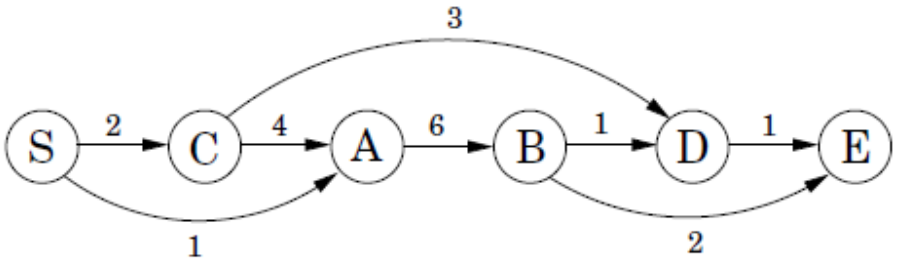
- Ordenação parcial
- Economizando memória

Problemas alvo para Programação Dinâmica (PD)

- Problema pode ser dividido em subproblemas menores.
- Sub-estrutura ótima (princípio da otimalidade)
  - Solução **ótima** do problema inclui soluções **ótimas** dos subproblemas.
- Subproblemas **são** sobrepostos.
  - Número “pequeno” de subproblemas distintos.

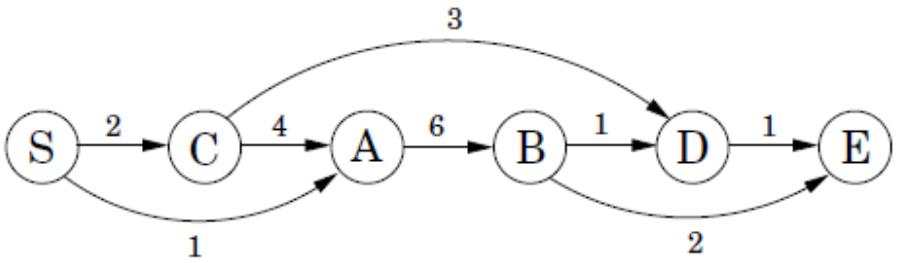
Linearização de Grafos Direcionados Acíclicos (DAGs)





Algoritmo de Linearização:
Percorrer vértices na ordem de grau de entrada.
Diminuir das arestas a cada passo.

Recorrência a partir de DAGs



Algoritmo de menor caminho em DAGs

initialize all $\text{dist}(\cdot)$ values to ∞
$\text{dist}(s) = 0$
for each $v \in V \setminus \{s\}$, in linearized order:
    $\text{dist}(v) = \min_{(u,v) \in E} \{\text{dist}(u) + l(u,v)\}$

Subproblemas Menores

Subestrutura Ótima

Subsequência Crescente Mais Longa

- Problema: Dada uma sequência de números naturais, definir qual a subsequência crescente com mais elementos.

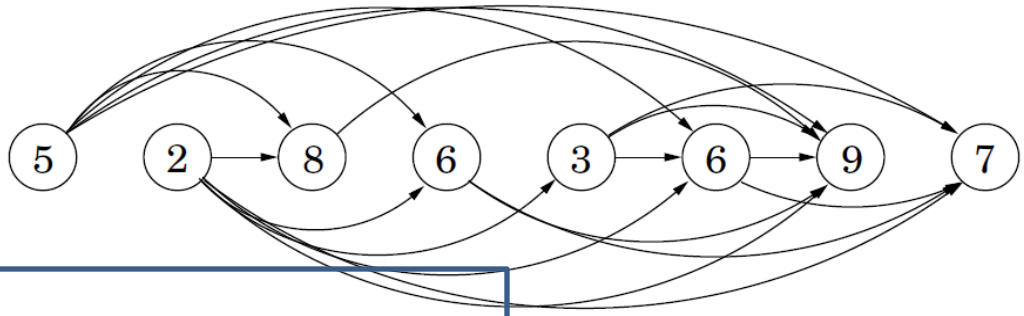
5 2 8 6 3 6 9 7

MSC: Recorrência e DAG implícito

5 2 8 6 3 6 9 7

SCML: Recorrência

Etapa1:



$$L(j) = 1 + \max\{L(i) : (i, j) \in E\}$$

Maior caminho em DAG

$$SCML = \max_{1 \le j \le n}\{L(j)\}$$

Etapa 2: Algoritmo Recursivo

$$L(j) = 1 + \max\{L(i) : (i, j) \in E\}$$

Etapa 3: Algoritmo Iterativo (Tabela)

- Vetor L, preenchido da menor posição para maior.

Etapa 3: Complexidade

```
for j = 1, 2, ..., n:
    L(j) = 1 + max{L(i) : (i, j) ∈ E}
return max_j L(j)
```

Etapa 4: Construindo solução

```
for j = 1, 2, ..., n:
    L(j) = 1 + max{L(i) : (i, j) ∈ E}
return max_j L(j)
```

5 2 8 6 3 6 9 7

Distância de Edição

- Transformar uma sequência em outra ao menor custo.
  - Casamento, substituição, inserção, remoção.

```
SITUADO
ESTUDO-
```

```
-SITUADO
ES-TU-DO
```

Edição: Subproblemas

- Problema: Alinhar duas sequências de caracteres $x[1 \cdots m]$ $y[1 \cdots n]$ $E(m, n)$

Etapa 1: Equação de Recorrência

- Problema: Alinhar duas sequências de caracteres $x[1 \cdots m]$ $y[1 \cdots n]$ $E(m, n)$

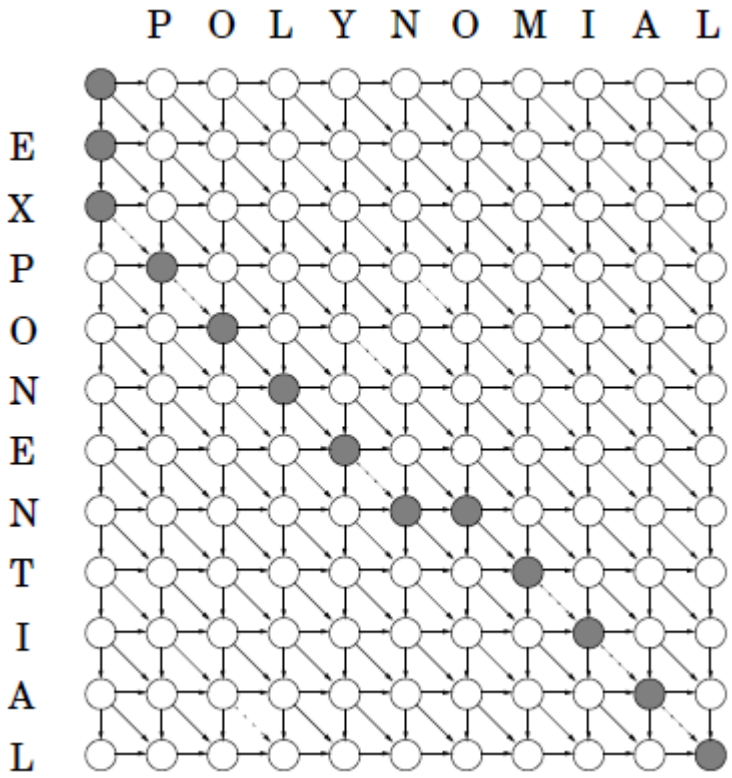
- Subproblema: alinhamento de prefixos

$E(i, j)$ $\begin{matrix} x[i] \\ - \end{matrix}$ **or** $\begin{matrix} - \\ y[j] \end{matrix}$ **or** $\begin{matrix} x[i] \\ y[j] \end{matrix}$

- Composição: remover, inserir, casar

$$E(i, j) = \min\{1 + E(i-1, j),\ 1 + E(i, j-1),\ \mathbf{diff}(i, j) + E(i-1, j-1)\}$$

Distância de Edição - DAG



Etapa 2: Algoritmo Recursivo

$$E(i,j) \quad = \quad \min\{1 + E(i-1,j),\ 1 + E(i,j-1),\ \mathbf{diff}(i,j) + E(i-1,j-1)\}$$

Etapa 3: Algoritmo Iterativo

$$E(i,j) \quad = \quad \min\{1 + E(i-1,j),\ 1 + E(i,j-1),\ \mathbf{diff}(i,j) + E(i-1,j-1)\}$$

Etapa 3: Complexidade

```
for i = 0, 1, 2, ..., m:
    E(i, 0) = i
for j = 1, 2, ..., n:
    E(0, j) = j
for i = 1, 2, ..., m:
    for j = 1, 2, ..., n:
        E(i, j) = min{E(i − 1, j) + 1, E(i, j − 1) + 1, E(i − 1, j − 1) + diff(i, j)}
return E(m, n)
```

- Θ(*mn*) de tempo e espaço

Distância de Edição - Exemplo

		P	O	L	I	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5										
E	6										
N	7										
C	8										
I	9										
A	10										
L	11										

Etapa 4: Solução I – armazenar

```
for i = 0, 1, 2, ..., m:
    E(i, 0) = i
for j = 1, 2, ..., n:
    E(0, j) = j
for i = 1, 2, ..., m:
    for j = 1, 2, ..., n:
        E(i, j) = min{E(i − 1, j) + 1, E(i, j − 1) + 1, E(i − 1, j − 1) + diff(i, j)}
return E(m, n)
```

Etapa 4: Solução I – calcular

```
for i = 0, 1, 2, ..., m:
    E(i, 0) = i
for j = 1, 2, ..., n:
    E(0, j) = j
for i = 1, 2, ..., m:
    for j = 1, 2, ..., n:
        E(i, j) = min{E(i − 1, j) + 1, E(i, j − 1) + 1, E(i − 1, j − 1) + diff(i, j)}
return E(m, n)
```

Problema da mochila

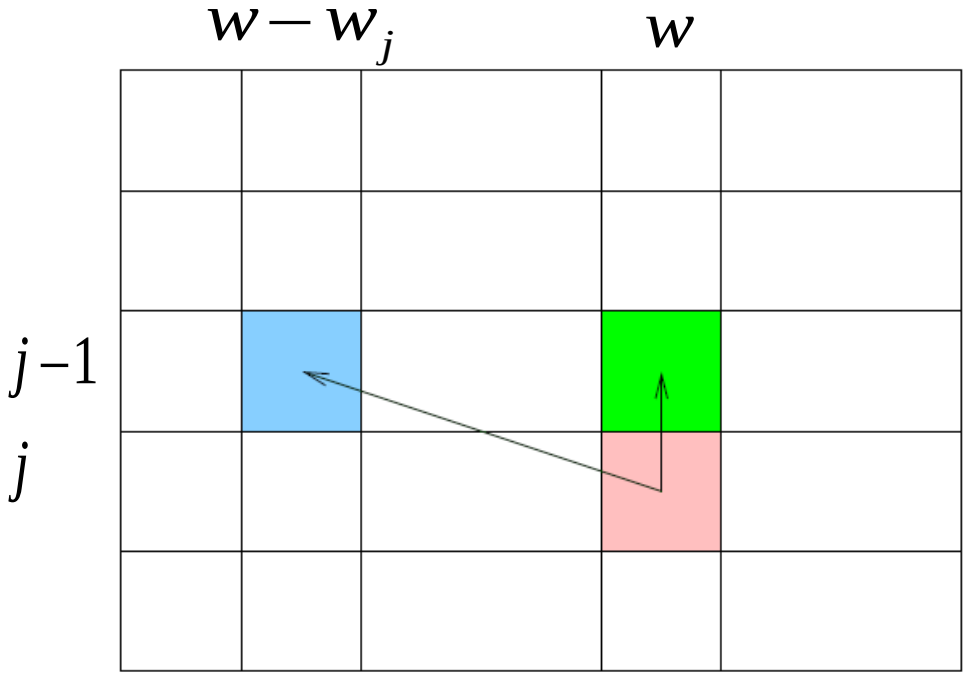
- Ladrão está com uma mochila que suporta no máximo 10 quilos e quer o maior lucro possível

Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

Etapa 1: Equação de Recorrência

$K(w, j)$

$$K(w,j) = \max\{K(w - w_j, j-1) + v_j, K(w, j-1)\}$$



$$K(w, j) = \max\{K(w, j-1), K(w-w_j, j-1) + v_j\}$$

Etapa 2: Algoritmo Recursivo

$$K(w, j) = \max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}$$

Etapa 3: Algoritmo Iterativo

$$K(w, j) = \max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}$$

Etapa 3: Complexidade

```
Initialize all K(0, j) = 0 and all K(w, 0) = 0
for j = 1 to n:
    for w = 1 to W:
        if w_j > w:    K(w, j) = K(w, j − 1)
        else:    K(w, j) = max{K(w, j − 1), K(w − w_j, j − 1) + v_j}
return K(W, n)
```

- Somente para valores inteiros
- Θ(*Wn*)

Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

k \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							

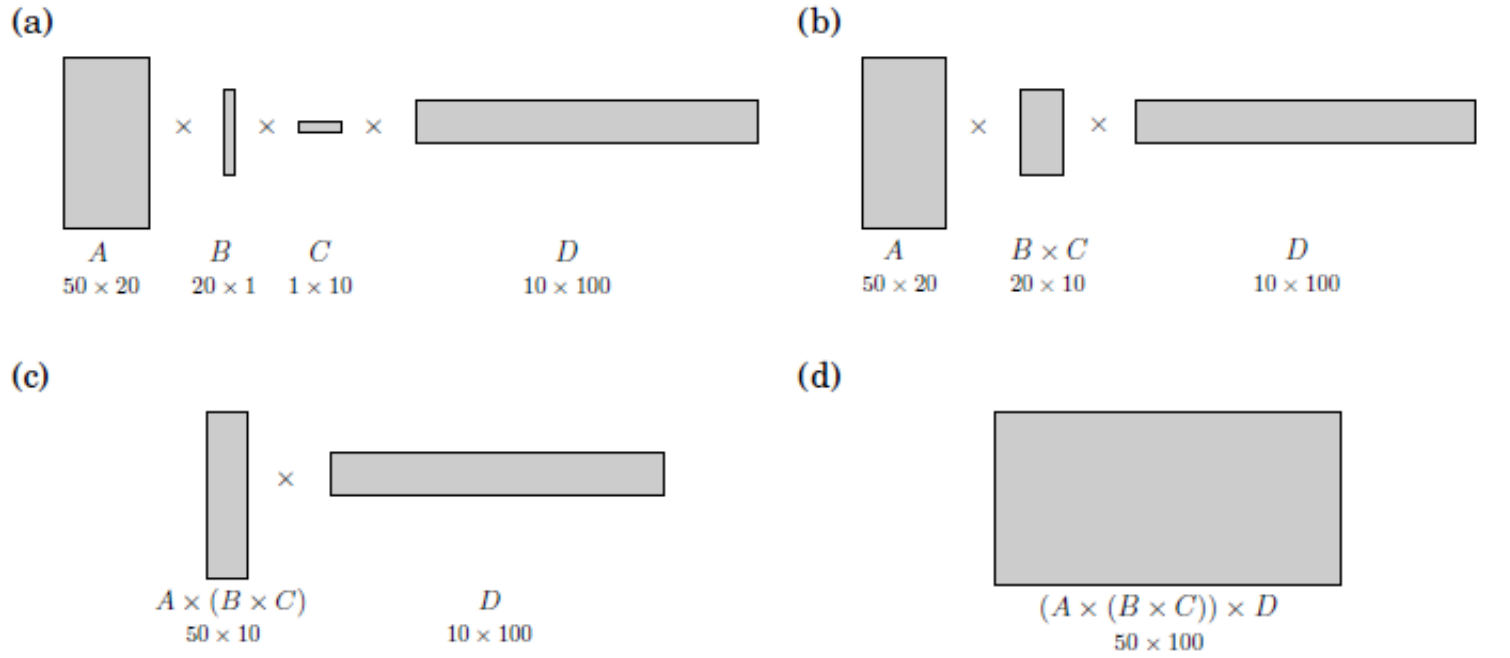
Etapa 4: Solução

Initialize all $K(0,j)=0$ and all $K(w,0)=0$
for $j=1$ to n:
    for $w=1$ to W:
        if $w_j > w$: $K(w,j)=K(w,j-1)$
        else: $K(w,j)=\max\{K(w,j-1), K(w-w_j,j-1)+v_j\}$
return $K(W,n)$

Ordem Ótima de Multiplicação de Matrizes

- AxBxCxD, de dimensões 50x20, 20x1,1x10, e 10x100, respectivamente

Figure 6.6 $A \times B \times C \times D = (A \times (B \times C)) \times D$.

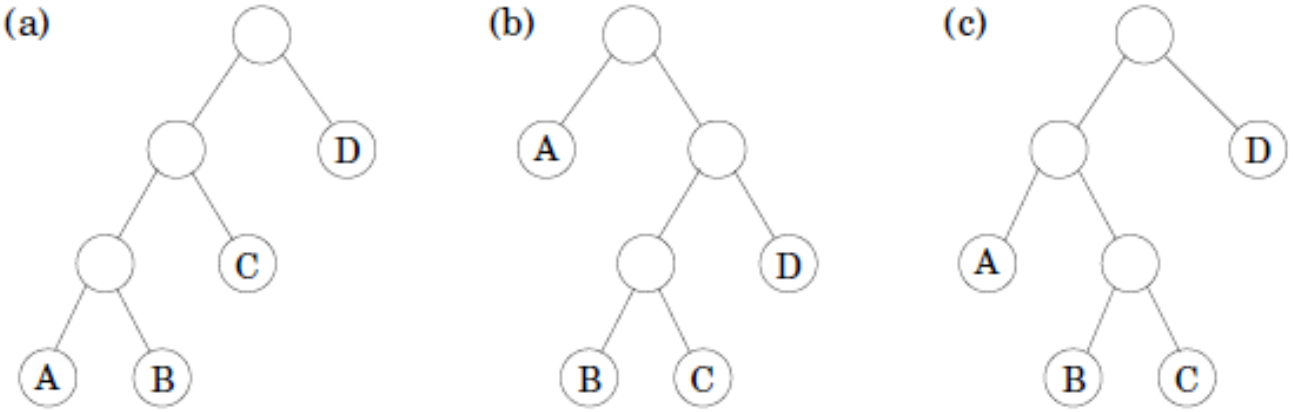


Fonte: Dasgupta et al., 2006

Ordem Ótima de Multiplicação de Matrizes

Parenthesization	Cost computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	$120,200$
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	$60,200$
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	$7,000$

Figure 6.7 (a) $((A \times B) \times C) \times D$; (b) $A \times ((B \times C) \times D)$; (c) $(A \times (B \times C)) \times D$.



Fonte: Dasgupta et al., 2006

Ordem Ótima de Multiplicação de Matrizes

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

C(1,1)	C(1,2)	C(1,3)	C(1,4)
	C(2,2)	C(2,3)	C(2,4)
		C(3,3)	C(3,4)
			C(4,4)

C(i,j) custo mínimo para multiplicar as matrizes de i a j.

Ordem Ótima de Multiplicação de Matrizes

- Multiplicar n matrizes
  - A_1 x A_2 x ... X A_n , com dimensões m_0 x m_1, m_1 x m_2, ..., m_{n-1} x m_n , respectivamente.
  - Qual a ordem ótima de realizar tal multiplicação?
  - C(i,j) = custo mínimo para multiplicar A_i x A_{i+1} x ... x A_j
  - A quantidade de subproblemas depende do número de multiplicações, $|j-i|$.
  - O menor subproblema, i=j, C(i,i) = 0

Ordem Ótima de Multiplicação de Matrizes

- Cada multiplicação divide o problema em dois subproblemas:
  - Para algum k, $A_i \times A_{i+1} \times \ldots \times A_k$ e $A_{k+1} \times A_{k+2} \times \ldots \times A_j$
  - $C(i,j) = \min_{i \leq k < j} \{C(i,k) + C(k+1,j) + m_{i-1} * m_k * m_j$

```
for i = 1 to n:   C(i,i) = 0
for s = 1 to n-1:
  for i = 1 to n-s:
    j = i + s
    C(i,j) = min{C(i,k) + C(k+1,j) + m_{i-1} · m_k · m_j : i ≤ k < j}
return C(1,n)
```

Considerações Finais

- Diferença entre PD e D&C
  - Sobreposição de problemas
- Definição da Equação de Recorrência
  - Grafo induzido
  - Automatização dos passos
- Memoização