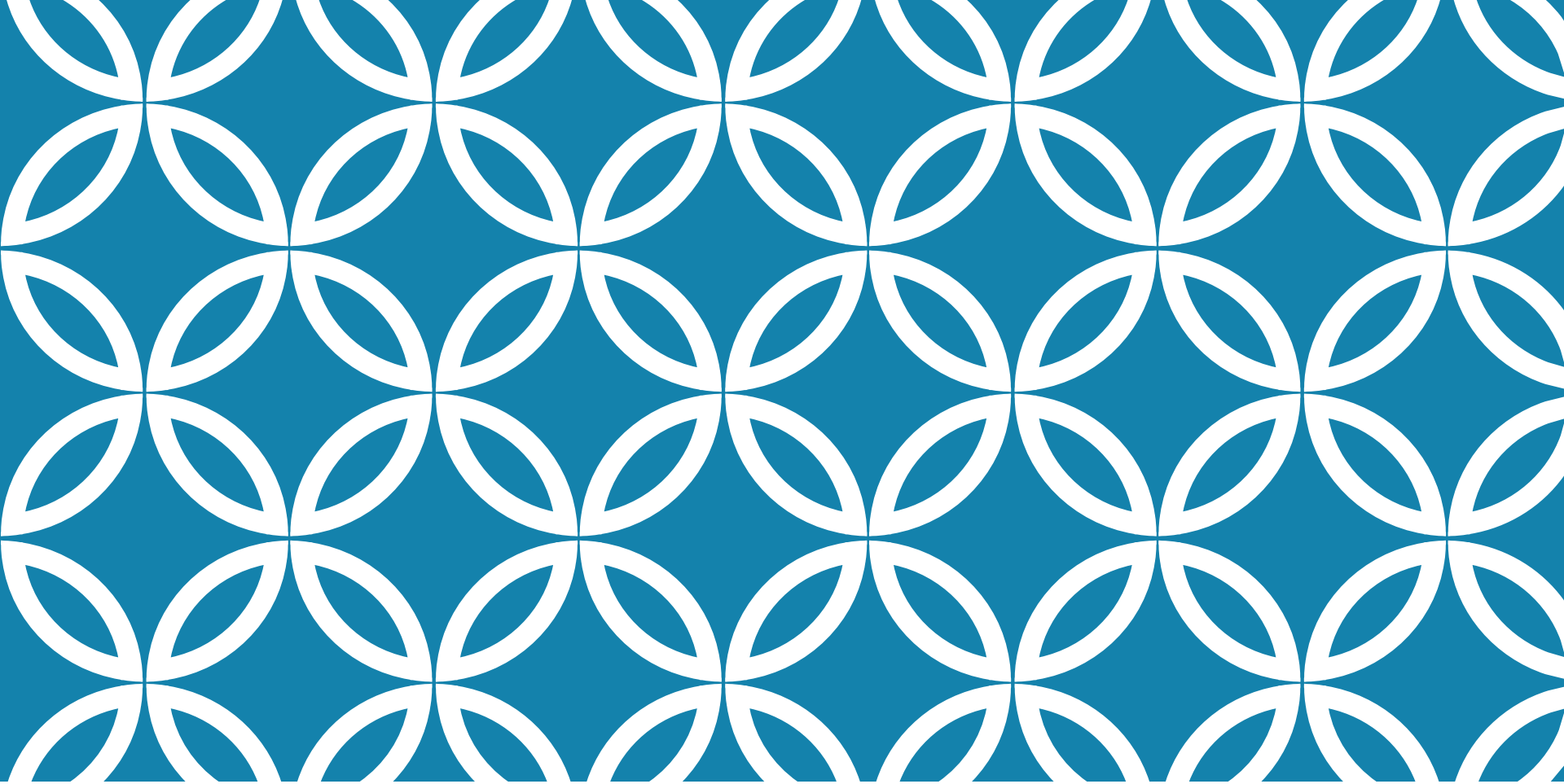

PROGRAMAÇÃO DINÂMICA

Prof. André Chaves Lima

INTRODUÇÃO À PROGRAMAÇÃO DINÂMICA

O que é Programação Dinâmica?

- Técnica para resolver problemas grandes dividindo-os em partes menores.
- Reutiliza soluções de subproblemas para evitar cálculos repetidos.

Quando usar?

- Problemas com subproblemas sobrepostos.
- A solução do problema maior depende das soluções ótimas dos subproblemas.

CARACTERÍSTICAS DE PROBLEMAS COM PROGRAMAÇÃO DINÂMICA

Subproblemas Sobrepostos:

- Partes menores do problema aparecem várias vezes.

  Exemplo: Fibonacci ($F(n)=F(n-1)+F(n-2)F(n)=F(n-1)+F(n-2)$).

Subestrutura Ótima:

- A solução do problema maior pode ser construída a partir de soluções ótimas de subproblemas.

EXEMPLO INTUITIVO: SUBIR UMA ESCADA

Problema:

Você pode subir 1 ou 2 degraus de cada vez. Quantas maneiras existem para subir nnn degraus?

Resposta:

$F(n)=F(n-1)+F(n-2)F(n) = F(n-1) + F(n-2)F(n)=F(n-1)+F(n-2)$.

O número de maneiras de chegar ao degrau n depende de como você chegou ao degrau anterior e dois degraus atrás.

PROBLEMA: MULTIPLICAÇÃO DE CADEIAS DE MATRIZES

Objetivo: Encontrar a ordem ótima para multiplicar uma sequência de matrizes.

Por que isso é importante?

A ordem de multiplicação afeta o número de operações.

Multiplicação de matrizes **não é comutativa,** mas **é associativa**.

Exemplo:

Matriz $A_1(10x20)$, $A_2(20x30)$, $A_3(30x40)$.

Ordem $(A_1 \times A_2) \times A_3$: 18.000 operações.

Ordem $A_1 \times (A_2 \times A_3)$: 32.000 operações.

COMO RESOLVER COM PROGRAMAÇÃO DINÂMICA

Dividimos o problema em subproblemas menores.

- **Subproblema:** Multiplicar matrizes de A[*i*] até A[*j*].

Definimos:

- m[i,j]: Custo mínimo para multiplicar A[*i*] até A[*j*].

Fórmula Recorrente:

$$m[i,j] = \min_{i \leq k < j} \left(m[i,k] + m[k+1,j] + p[i-1] \cdot p[k] \cdot p[j] \right)$$

Condição Base:

m[i,i]=0 (uma única matriz não tem custo de multiplicação).

ALGORITMO DE MULTIPLICAÇÃO DE CADEIAS DE MATRIZES

MATRIX-CHAIN-ORDER (p)

```
1   n = p.comprimento − 1
2   sejam m[1 .. n, 1 .. n] e s[1 .. n − 1,2 .. n] tabelas novas
3   for i = 1 to n
4       m[i, i] = 0
5   for l = 2 to n                    // l é o comprimento da cadeia
6       for i = 1 to n − l + 1
7           j = i + l − 1
8           m[i, j] = ∞
9           for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m e s
```

ANÁLISE DO ALGORITMO

Tamanho da tabela m[*i*,*j*]: $n \times n$ ($O(n^2)$).

Laços aninhados: 3 laços para subsequências, índices e divisões.

Complexidade de tempo: $O(n^3)$.

Espaço total usado: $O(n^2)$.

EXEMPLO PRÁTICO COM MATRIZES

Entrada:

$A_1(10 \times 20)$, $A_2(20 \times 30)$, $A_3(30 \times 40)$.

Dimensões: p = [10, 20, 30, 40].

Etapas:

Calcular m[1,2]=10·20·30 = 6.000.

Calcular m[2,3]=20·30·40= 24.000.

Calcular (m[1, 3]):

- Divisão 1: (A1×A2)×A3: 6.000+24.000+10·20·40=32.000.
- Divisão 2: A1×(A2×A3): 18.000 (ótima).

ALGORITMO PARA OBTER A SOLUÇÃO ÓTIMA

- Após calcular os custos mínimos, determinamos a **ordem ótima** usando a tabela *s*[*i*,*j*].

```
ImprimirSolucaoOtima(s, i, j):
    se i == j:
        imprime "A[i]"
    senão:
        imprime "("
        ImprimirSolucaoOtima(s, i, s[i][j])
        ImprimirSolucaoOtima(s, s[i][j] + 1, j)
        imprime ")"
```

- **Entrada:** Tabelas s, índices i e j.
- **Saída:** Parênteses que indicam a ordem ótima.