Projeto e Análise de Algoritmos
Aula 07 – Programação Dinâmica

Prof. Napoleão Nepomuceno

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

Objetivos

- Apresentar o princípio da Programação Dinâmica;

- Discutir os elementos fundamentais da Programação Dinâmica;

- Abordar alguns exemplos.

Programação Dinâmica

- Estratégia para resolução de problemas de natureza recursiva;
- Combina a solução de subproblemas.

- Diferente de Divisão e Conquista:
  - Subproblemas não são independentes;
  - Subproblemas compartilham subsubproblemas.

Programação Dinâmica

- Estrutura do problema é recursiva ...
  - Por que um algoritmo recursivo não seria uma boa solução?

- Algoritmos recursivos executariam mais passos do que o necessário ...
  - Subproblemas comuns seriam resolvidos mais de uma vez.

Exemplo *Fibonacci*

- Números de *Fibonacci* são definidos pela seguinte função recursiva:

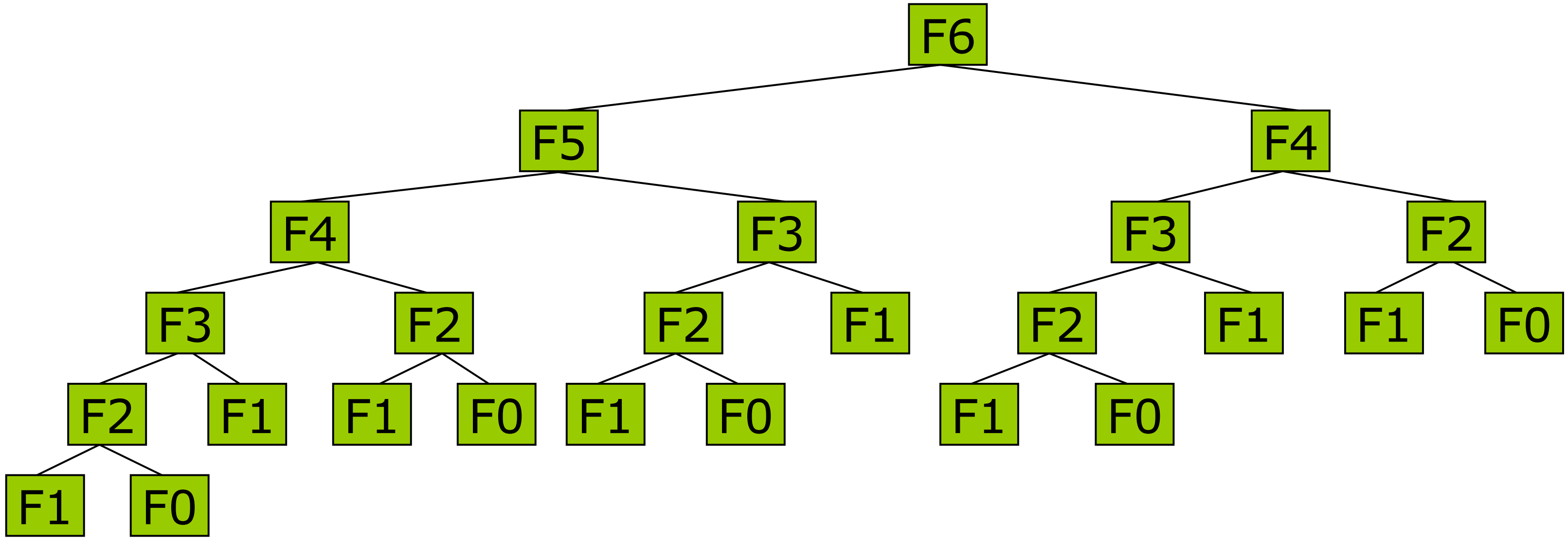
$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

- Código recursivo para gerar o *n*-ézimo termo:

```
int fib(int n){
    if(n <= 1)
      return n;
    else
      return fib(n – 1) + fib(n – 2);
}
```

Exemplo *Fibonacci*

- O algoritmo recursivo é ineficiente, pois há muito trabalho redundante.



Princípio da Programação Dinâmica

- Guardar os resultados dos subproblemas que já foram resolvidos em uma tabela;

- Consultar a tabela sempre que estes resultados forem necessários.

Exemplo *Fibonacci*

- Para o caso dos Números de Fibonacci:
  - Resolver o problema de forma sequencial;
  - Guardando apenas os dois últimos resultados.

```
int fibonacci(int n) {
   if (n <= 1)
      return n;
   else {
      int resultado = 1;
      int ultimo    = 1;
      int penultimo = 1;
      for (int i = 3; i <= n; i++) {
         resultado = ultimo + penultimo;
         penultimo = ultimo;
         ultimo = resultado;
      }
      return resultado;
   }
}
```

Programação Dinâmica

- Aplica-se tipicamente a problemas de otimização, onde uma série de escolhas deve ser feita para se alcançar uma solução ótima;

- O problema apresenta muitas soluções viáveis, cada uma delas associada a um valor que se quer otimizar.

Desenvolvimento da Programação Dinâmica

- Caracterizar a estrutura de uma solução ótima;
- Definir recursivamente o valor de uma solução;
- Computar o valor de uma solução ótima num processo *bottom-up;*
- Construir uma solução ótima a partir de informações computadas.

Multiplicação de Cadeia de Matrizes

- Entrada:
  - Multiplicação de uma cadeia de n matrizes
  - $A_1 A_2 \ldots A_n$.
- Saída:
  - Parentização da cadeia que minimiza o total de multiplicações escalares.
- Parentizações da cadeia $A_1A_2A_3A_4$:
  - $(A_1(A_2(A_3A_4)))$, $(A_1((A_2A_3)A_4))$, $((A_1A_2)(A_3A_4))$, $((A_1(A_2A_3))A_4)$, $(((A_1A_2)A_3)A_4)$

Multiplicação de Cadeia de Matrizes

- Custo de multiplicar duas matrizes A_{pxq} e B_{qxr}:

MATRIX-MULTIPLY(A, B)

if $A.columns \neq B.rows$
    error "incompatible dimensions"
else let C be a new $A.rows \times B.columns$ matrix
    for $i = 1$ to $A.rows$
        for $j = 1$ to $B.columns$
            $c_{ij} = 0$
            for $k = 1$ to $A.columns$
                $c_{ij} = c_{ij} + \boxed{a_{ik} \cdot b_{kj}}$
    return C

- Número de multiplicações escalares: $p \times q \times r$.

Multiplicação de Cadeia de Matrizes

- O custo de se multiplicar uma cadeia $A_1A_2 \ldots A_n$ depende da parentização!
- Considere uma cadeia de 3 matrizes $A_1A_2A_3$ com dimensões 10×100, 100×5 e 5×50.
  - $((A_1A_2)A_3)$
    - $A_1A_2 = B_1 \rightarrow 10 \times 100 \times 5 = 5000$
    - $B_1A_3 \rightarrow 10 \times 5 \times 50 = 2500$
    - 7500 multiplicações escalares
  - $(A_1(A_2A_3))$
    - $A_2A_3 = B_2 \rightarrow 100 \times 5 \times 50 = 25000$
    - $A_1B_2 \rightarrow 10 \times 100 \times 50 = 50000$
    - 75000 multiplicações escalares

Multiplicação de Cadeia de Matrizes

- Por que não testar o custo de todas as diferentes parentizações possíveis?

- Número de parentizações possíveis:

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

- A quantidade é exponencialmente grande: $\Omega(2^n)$.

Multiplicação de Cadeia de Matrizes

- **Etapa 1**: Estrutura de uma solução ótima:
  - Seja $A_{i..j} = A_i A_{i+1} \dots A_j$, onde $i \leq j$;
  - Uma parentização ótima de $A_{i..j}$ deve dividir o produto em $A_{i..k}$ e $A_{k+1..j}$ para algum $i \leq k < j$;
  - Seu custo é dado pela soma dos custos:
    - Cálculo de $A_{i..k}$ e $A_{k+1..j}$;
    - Multiplicação das matrizes resultantes de $A_{i..k}$ e $A_{k+1..j}$.
  - Supondo que a parentização ótima de $A_{i..j}$ divida o produto entre A_k e A_{k+1}.
    - Então a parentização ótima de $A_{i..j}$ pressupõe as parentizações ótimas de $A_{i..k}$ e $A_{k+1..j}$.

Multiplicação de Cadeia de Matrizes

- **Etapa 2**: Solução recursiva:
  - Seja $m[i,j]$ o número mínimo de multiplicações escalares necessárias para se calcular a matriz $A_{i..j}$.
  - Se $i = j$, o problema é trivial e $m[i,j] = 0$.
  - Se a divisão ótima ocorre entre A_k e A_{k+1},

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \ .$$

  - Mas não conhecemos o valor de k!
  - Devemos verificar todos os valores de $k = i, i+1, \ldots, j-1$.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \ , \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \ . \end{cases}$$

  - Onde cada matriz A_i é p_{i-1} x p_i.

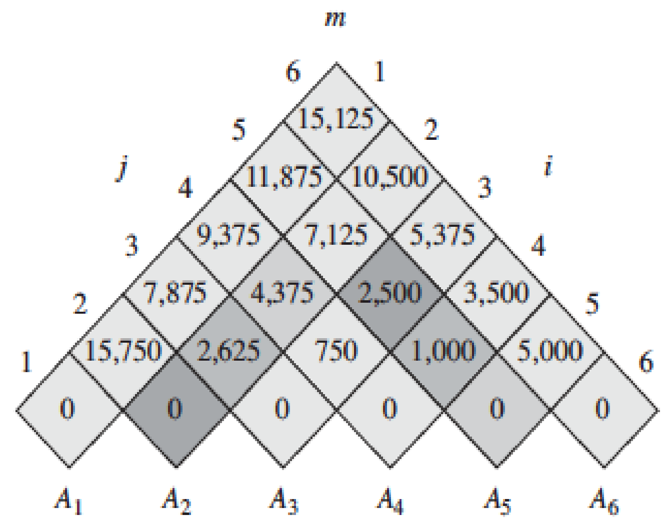
Multiplicação de Cadeia de Matrizes

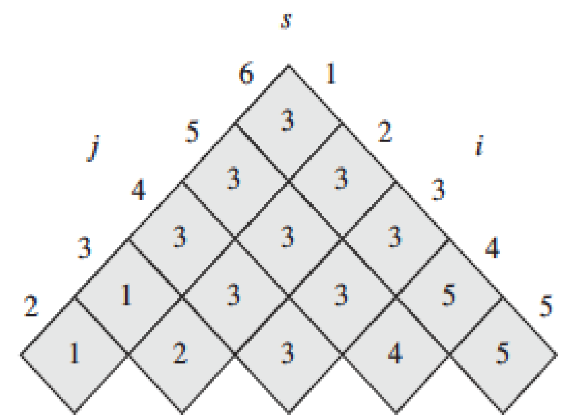
- **Etapa 3**: Computar custo ótimo:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

MATRIX-CHAIN-ORDER(p)

```
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n              // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```



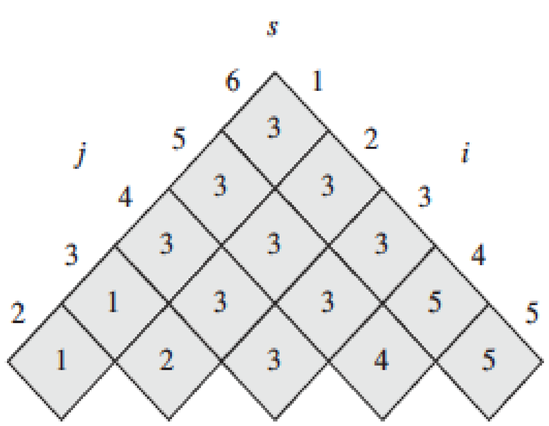


$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000 , \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 , \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$

Multiplicação de Cadeia de Matrizes

- **Etapa 4**: Construção da solução:

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```



$$((A_1(A_2A_3))((A_4A_5)A_6))$$