① 10ª Aula

Programação dinâmica

→ "Recursão com apoio de uma tabela"

→ Programação dinâmica ≠ Programação de computadores

→ Aplicável a problemas de otimização

→ Estrutura recursiva: combina soluções de subproblemas que não são independentes

②

→ Relação problema-subproblema geralmente expressa em termos de uma fórmula recursiva

→ Não existe prova de correção

→ Estratégia bottom-up e possui muitas sequências de decisões

③ Cálculo do fatorial

Solução recursiva natural:

```
fatorial (n){
  if (n == 0)
    return 1;
  else
    return n * fatorial(n-1);
}
```

Complexidade: $O(n)$

Como é uma solução com programação dinâmica?

④ Solução por programação dinâmica:

```
fatorial (n){
  if (n == 0)
    return 1;
  else
    if (fat[n] != 0) // caso já calculado
      return fat[n];
    else
      return fat[n] = n * fat(n-1);
}
```

Persistência de memória

fat: 

n: suficientemente grande

⑤ Multiplicação de matrizes

Problema:

Multiplicar n matrizes realizando o número mínimo de operações.

$$M = M_1 \times M_2 \times \ldots \times M_n$$

Exemplo:

$(A_1)_{10 \times 100}$, $(A_2)_{100 \times 5}$, $(A_3)_{5 \times 50}$

$(A_1 \times A_2) \times A_3$ — 7500 operações

$A_1 \times (A_2 \times A_3)$ — 75.000 operações

⑥ Solução por força bruta:

- Impraticável
- $\Omega(2^n)$ possibilidades (números de Catalão)

Aplicando programação dinâmica:

- É possível determinar uma solução eficiente (polinomial)

(7)

A forma de se obter o produto ótimo é determinado por uma posição.

$$\overset{p_{i-1}}{A_i} \overset{p_i}{\cdots} A_k \Big| \left(A_{k+1} \overset{p_{k+1}}{\cdots} \overset{p_j}{A_j} \right)$$

Particiona-se o problema em uma posição k, resolve-se otimamente os dois subproblemas $A_1 \cdots A_k$ e $A_{k+1} \cdots A_n$, multiplicando-se no final as duas matrizes obtidas.

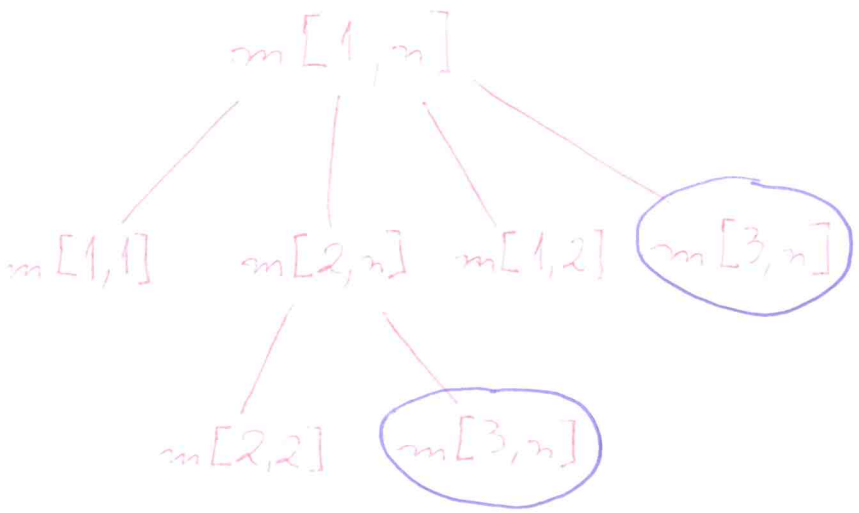
(8)

Denotemos por $m[i,j]$ o número mínimo de multiplicações necessárias para multiplicar $A_i \cdots A_j$, $i \leq j$. Temos que

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

onde $p_0, p_1, \ldots, p_n$ determinam as dimensões das das matrizes: cada A_i tem dimensão $p_{i-1} \times p_i$.

(9)



A solução (algoritmo) recursiva para calcular $m[i,j]$ por esta fórmula seria excessivamente lenta. (exponencial - Escreva a recorrência e verifique!)

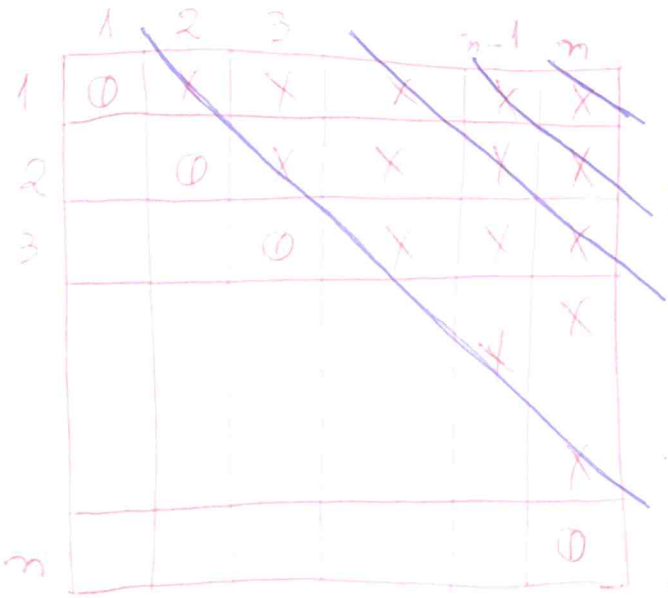
---

(10)

Com programação dinâmica é possível construir uma tabela para armazenar os valores de $m[i,j]$ e calculá-los bottom-up, ou recursivamente usando memorização.

1. $m[i,i] = 0$, para todo i $(1 \leq i \leq n)$
2. $m[i,i+1]$ são calculados, para $1 \leq i \leq n-1$
3. $m[i,i+2]$ são calculados, para $1 \leq i \leq n-2$
4. ... e assim por diante.

(11)



m[i, j]

$$\sum_{i=1}^{n-1} i(n-i) = O(n^3)$$

(12)

```
ordem_cadeia_matrizes( p, n){
1.  para i de 1 até n faça m[i,i] = 0
2.  para l de 2 até n faça{
3.      para i de 1 até n-l+1 faça{
4.          j = i + l - 1;
5.          m[i,j] = ∞;
6.          para K de i até j-1 faça{
7.              q = m[i,K] + m[K+1,j] + p[i-1] p[K] p[j];
8.              se q < m[i,j]{
9.                  m[i,j] = q;
10.                 s[i,j] = k; // usado para recuperar a
11.                                parentização ótima!
12.             }
13.         }
14.     }
```

(13)

```
imprime_parenteses_ótimo(A, s, i, j){
1.  se (i == j)
2.      imprime "Ai";
3.  senão{
4.      imprime "(";
5.      imprime_parenteses_ótimo(A, s, i, s[i,j]);
6.      imprime_parenteses_ótimo(A, s, s[i,j]+1, j);
7.      imprime ")";
8.  }
```

(14)

```
calcula_produto(A, s, i, j){
1.  se (i == j)
2.      devolve Ai;
3.  senão{
4.      X = calcula_produto(A, s, i, s[i,j]);
5.      Y = calcula_produto(A, s, s[i,j]+1, j);
6.      devolve multiplicação(X, Y);
7.  }
```

(15) Atividades

(1) Simule a solução com programação dinâmica para as matrizes $(A_1)_{10 \times 100}$, $(A_2)_{100 \times 5}$ e $(A_3)_{5 \times 50}$.

(2) Simule também para as matrizes do quadro a seguir

matriz	A_1	A_2	A_3	A_4	A_5	A_6
dimensão	30x35	35x15	15x5	5x10	10x20	20x25