Algoritmos para Automação e Sistemas

Programação Dinâmica

Universidade Federal do Amazonas

Departamento de Eletrônica e Computação

Roteiro

- Programação Dinâmica
- Problemas de Otimização
  - Linha de Montagem
  - Multiplicação de Matrizes
- Princípios de Programação Dinâmica
- Maior Subseqüência Comum

Programação Dinâmica

- Resolver problemas combinando as soluções do subproblema (parecido com a divisão e conquista)
  - Subproblemas compartilham subsubproblemas
  - Resolve cada subsubproblema **uma vez só**
- Caracterizar a estrutura de uma solução ótima
- Definir recursiva. o valor de uma solução ótima
- Calcular o valor da solução ótima em processo *bottom-up* (de problemas menores para maiores)
- Construir uma solução ótima a partir de informações calculadas

Problemas de Otimização

- Problemas que apresentam várias soluções, cada uma com um valor (custo) associado
- Procura-se a solução com valor ótimo (mínimo ou máximo)
- Um solução geralmente apresenta uma estrutura
  - É composta de uma seqüência de escolhas
  - Tais escolhas devem ser feitas para se chegar à solução ótima

Exemplo:
Linha de Montagem

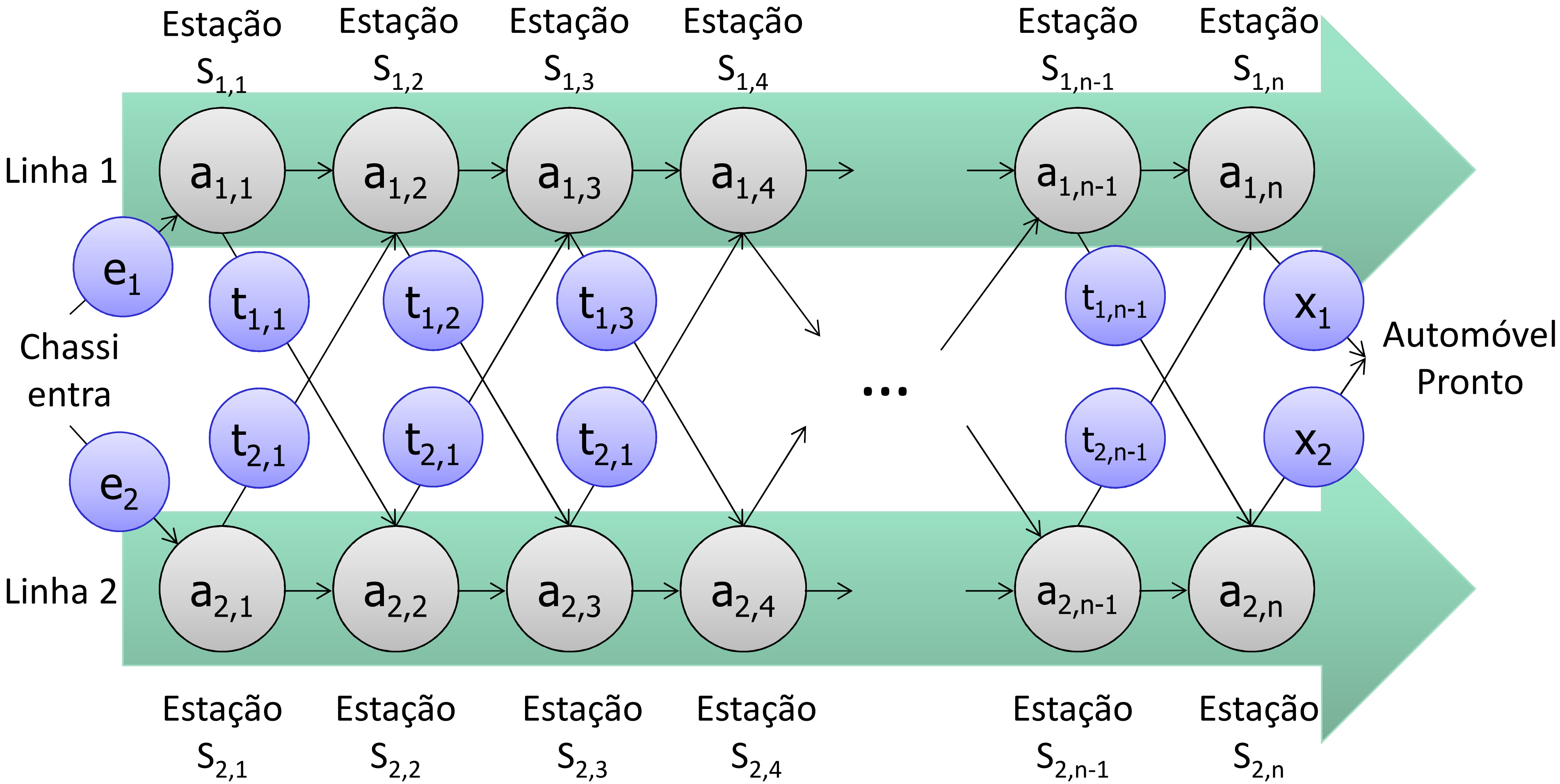
Linha de Montagem (1)

- A *Colonel Motors Corporation* produz automóveis em uma fábrica que tem duas linhas de montagem
- Um chassi de automóvel entra em cada linha de montagem
  - Tem as peças adicionadas a ele em uma série de estações
  - Um automóvel pronto sai no final da linha
- As estações foram construídas em épocas diferentes e com tecnologias diferentes
  - O tempo exigido em cada estação varia, até mesmo entre as estações na mesma posição nas duas linhas

Linha de Montagem (2)

- Cada Estação $S_{i,j}$ demanda $a_{i,j}$ tempo de montagem
- O tempo $t_{i,j}$ é o tempo de transferência do chassis após passagem por $S_{i,j}$
- Existe um tempo de entrada e_i e um tempo de saída x_i
- Problema:
  - Escolher as estações a fim de MINIMIZAR o tempo de passagem de um automóvel
  - Se, resolvido na "força bruta", demandaria tempo $O(2^n)$

Linha de Montagem (3)



Linha de Montagem (4)

- Primeira etapa: caracterizar a estrutura de uma solução ótima
  - Existem dois caminhos para estação $\mathbf{S_{1,j}}$
    - Vir de $\mathbf{S_{1,j-1}}$
    - Vir de $\mathbf{S_{2,j-1}}$. Nesse caso, considera-se $\mathbf{t_{2,j-1}}$
- Segunda etapa: Solução recursiva para o sub-problema:

$$f_1[j] = \begin{cases} e_1 + a_{1,1}, & j = 1 \\ min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}), & j \geq 2 \end{cases}$$

Tempo mais rápido

$$f_2[j] = \begin{cases} e_2 + a_{2,1}, & j = 1 \\ min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}), & j \geq 2 \end{cases}$$

Linha de Montagem (5)

Complexidade O(n)

```
FASTEST-WAY(a, t, e, x, n)
    f_1[1] ← e_1 + a_{1,1}
    f_2[1] ← e_2 + a_{2,1}
    for j ← 2 to n
            do if f_1[j-1]+a_{1,j} ≤ f_2[j-1]+t_{2,j-1}+a_{1,j}
               then  f_1[j] ← f_1[j-1]+a_{1,j}
                     l_1[j] ← 1
               else  f_1[j] ← f_2[j-1]+t_{2,j-1}+a_{1,j}
                     l_1[j] ← 2
               if f_2[j-1]+a_{2,j} ≤ f_1[j-1]+t_{1,j-1}+a_{2,j}
               then  f_2[j] ← f_2[j-1]+a_{2,j}
                     l_2[j] ← 2
               else  f_2[j] ← f_1[j-1]+t_{1,j-1}+a_{2,j}
                     l_2[j] ← 1
    if f_1[n]+x_1 ≤ f_2[n]+x_2
          then  f* ← f_1[n]+x_1
                l* ← 1
          else  f* ← f_2[n]+x_2
                l* ← 2
```

Entrada do chassi na linha

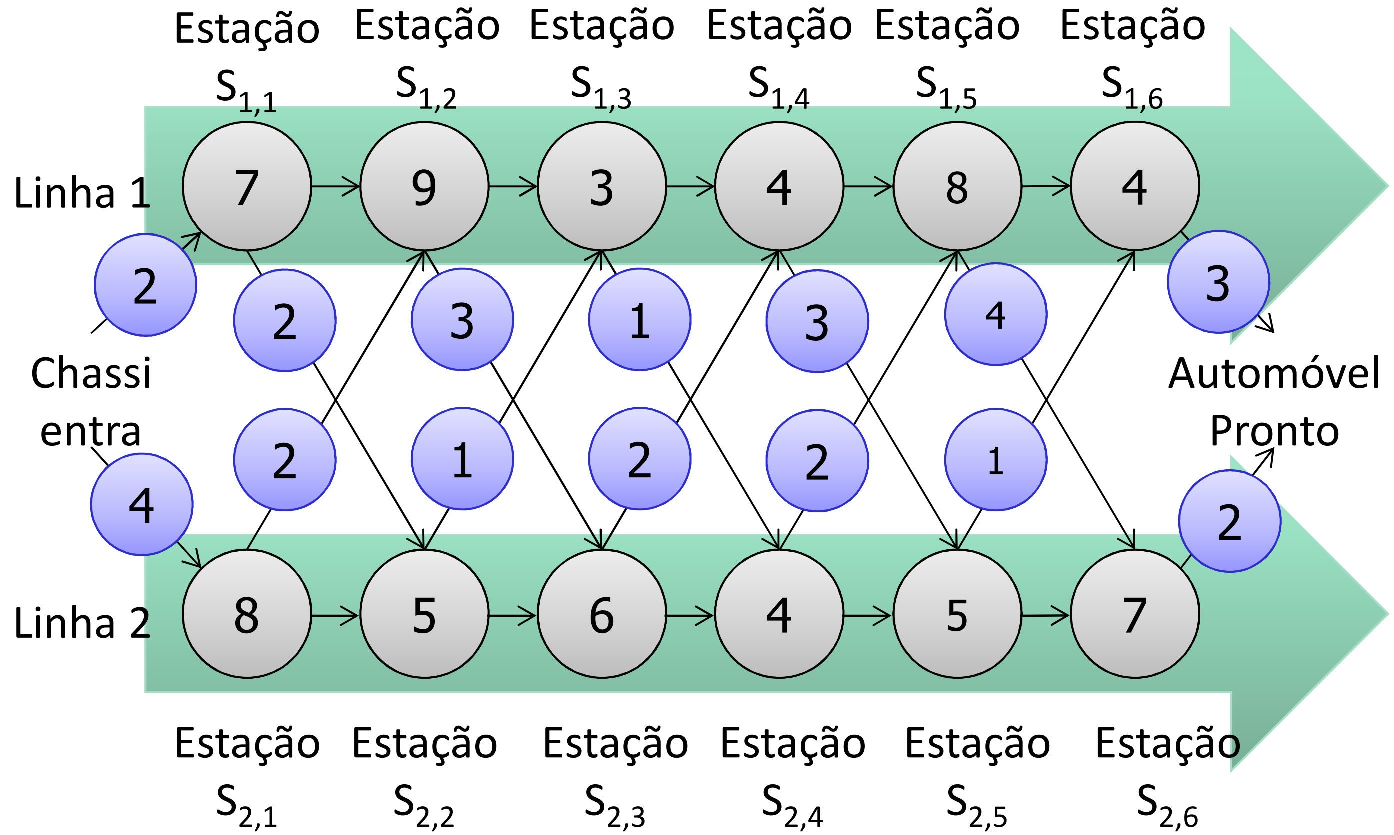
Calcula f_1 até n usando equação definida em slide anterior

Calcula f_2 até n usando equação definida em slide anterior

Cálcula melhor tempo de saída f^*

$f^* = min(f_1[n]+x_1, f_2[n]+x_2)$

Linha de Montagem (6)



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Linha de Montagem (7)

- O procedimento imprime as estações usadas, em ordem decrescente de número de estações

```
PRINT-STATIONS(l, n)
i ← l*
imprimir "linha " i ", estação " n
for j ← n downto 2
      do i ← l_i[j]
imprimir "linha " i ", estação " j – 1
```

PRINT-STATIONS produziria a saída:

linha 1, estação 6
linha 2, estação 5
linha 2, estação 4
linha 1, estação 3
linha 2, estação 2
linha 1, estação 1

Exemplo:
Multiplicação de cadeias de matrizes

Multiplicação de Matrizes (1)

- Duas matrizes *A:n×m e B:m×k* podem ser multiplicadas usando *nmk* multiplicações escalares

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} = \begin{pmatrix} \ldots & \ldots & \ldots \\ \ldots & c_{22} & \ldots \\ \ldots & \ldots & \ldots \end{pmatrix} \qquad c_{i,j} = \sum_{l=1}^{m} a_{i,l} \cdot b_{l,j}$$

- Problema: Computar o produto de muitas matrizes de forma eficiente
- A multiplicação de matrizes é associativa
  - *(AB)C = A(BC)*

Multiplicação de Matrizes (2)

- O algoritmo padrão é dado pelo pseudocódigo a seguir

```
MATRIX-MULTIPLY(A, B)
if colunas[A] ≠ linhas[B]
  then error "dimensões incompatíveis"
  else for i ← 1 to linhas[A]
    do for j ← 1 to colunas[B]
      do C[i,j] ← 0
        for k ← 1 to colunas[A]
          do C[i,j] ← C[i,j] + A[i,k] × B[k, j]
    return C
```

- Se A é uma matriz $n \times m$ e B é uma matriz $m \times k$, a matriz resultante C é uma matriz $n \times k$

Multiplicação de Matrizes (3)

- A ordem ditada pelos parênteses afeta o custo da multiplicação
- Considere $A \times B \times C \times D$, *onde*
  - A:30×1, B :1×40, C:40×10, D :10×25
- Custos (nmk):

30x1x40 30x40x10 30x10x25

  - $(AB)C)D =$ 1200 + 12000 + 7500 = 20700
  - $(AB)(CD) =$ 1200 + 10000 + 30000 = 41200
  - $A((BC)D) =$ 400 + 250 + 750 = 1400
- Procuramos uma seqüência ótima para multiplicar
  - A_1 X A_2 X … X A_n onde A_i é uma matriz d_{i-1} X d_i

Multiplicação de Matrizes (4)

- **Etapa 1:** Seja $M(i,j)$ o número mínimo de multiplicações necessárias, onde $i \leq j$

$$\prod_{k=i}^{j} A_k$$

Problema trivial: i=j
Problema não trivial: $i<j$

- Observações importantes:
  - Os parênteses mais externos da cadeia de matrizes, dividem a sequência em algum k, ($i \leq k < j$): $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$
  - A solução ótima da sequência (i,j) é formada por soluções ótimas das duas subsequências (i,k) e $(k+1,j)$

Multiplicação de Matrizes (5)

- **Etapa 2:** Uma solução recursiva:

$$M(i,i) = 0 \qquad \textit{se } i{=}j$$

$$M(i,j) = \min_{i \le k < j} \{ M(i,k) + M(k+1,j) + d_{i-1} d_k d_j \} \qquad \textit{se } i{<}j$$

onde *M(i,j)* representa o número mínimo de multiplicações

- O produto de matrizes $A_{i..k} A_{k+1..j}$ exige $d_{i-1} d_k d_j$ multiplicações escalares
- A implementação recursiva direta é exponencial
  - Muita computação redundante ocorre
  - No entanto, o número de subproblemas distintos é consideravelmente menor

Exemplo

Considere i=1 e j=n

$$M(i,i) = 0$$

$$M(i,j) = \min_{i \le k < j} \left\{ M(i,k) + M(k+1,j) + d_{i-1} d_k d_j \right\}$$

Quantos subproblemas nós temos?

Problemas na forma $M(i,i)$

Problemas da forma:

$M(1,1)=M(2,2)=...=M(n,n)=0$

São computados quase de graça, pois temos uma única matriz (cadeias de comprimento 1)!

Subproblemas na forma *M(i,i+1)*

- Podem ser computados em função dos problemas na forma *M(i,i)*

$$M(i,i) = 0$$

$$M(i,j) = \min_{i \le k < j} \{M(i,k) + M(k+1,j) + d_{i-1} d_k d_j\}$$

- M(1,2)=M(1,1)+M(2,2)+d...
- Único valor possível para *k* é 1

Subproblemas na forma M(i,i+2)

k=1

$$M(1,3) = \min_{1 \le k < 3} \{ M(1,k) + M(k+1,3) + d_0.d_k.d_3 \}$$

$$M(1,3) = \min \begin{cases} M(1,1) + M(2,3) + d_0.d_1.d_3 = 1 : (A_1).(A_2.A_3) \\ M(1,2) + M(3,3) + d_0.d_2.d_3 = 2 : (A_1.A_2).(A_3) \end{cases}$$

k=2

- Agora temos 2 valores possíveis para *k* (todos em função de valores já computados)
  - Isto é, *M(i,i), M(i,i+1)*

Subproblemas na forma M(i,i+3)

k=1

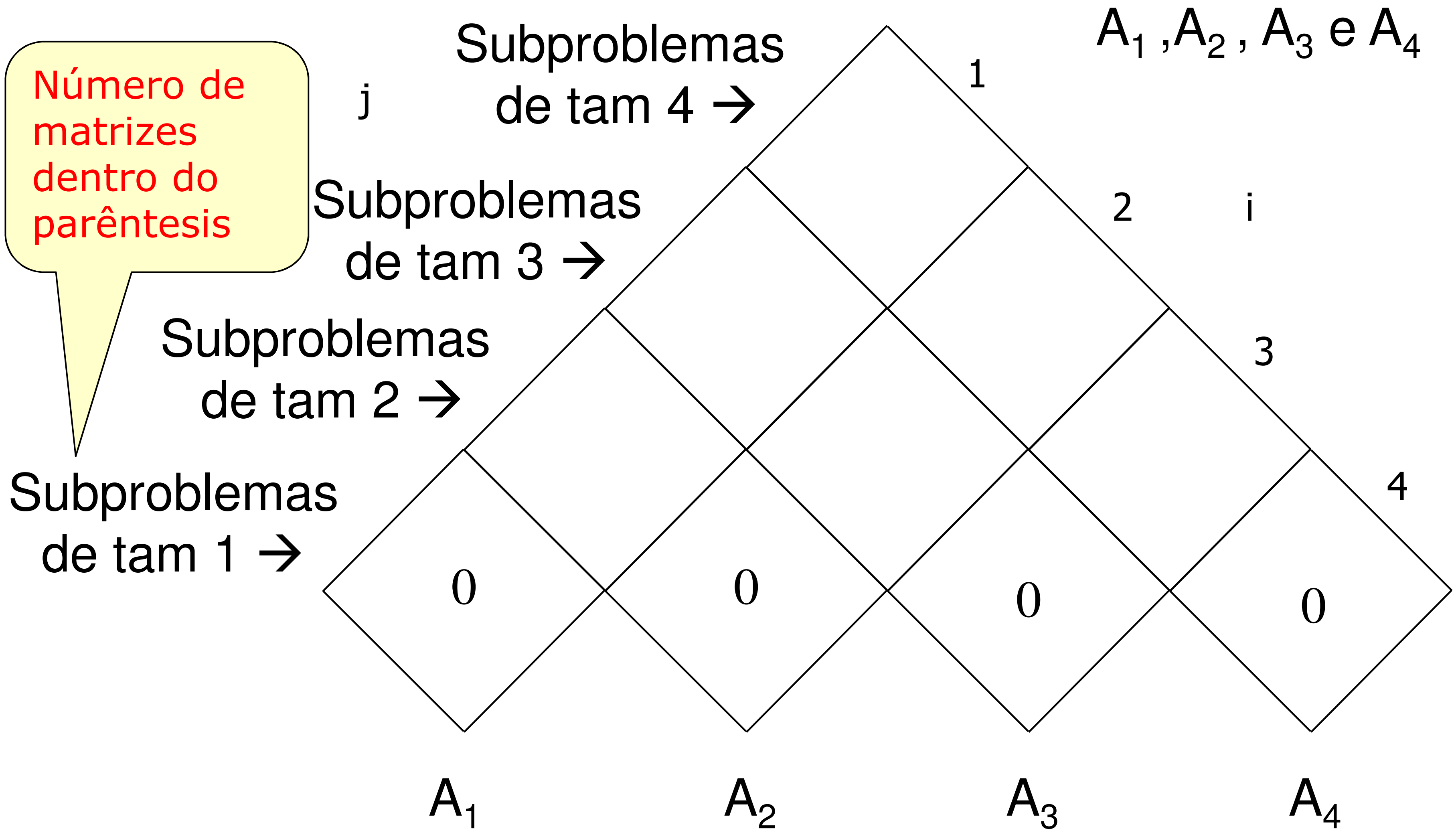
$$M(1,4) = \min_{1 \le k < 4} \{M(1,k) + M(k+1,4) + d_0.d_k.d_4\}$$

$$M(1,4) = \min \begin{cases} M(1,1) + M(2,4) + d_0.d_1.d_4 = 1 : (A_1).(A_2.A_3.A_4) \\ M(1,2) + M(3,4) + d_0.d_2.d_4 = 2 : (A_1.A_2).(A_3.A_4) \\ M(1,3) + M(4,4) + d_0.d_3.d_4 = 3 : (A_1.A_2.A_3).(A_4) \end{cases}$$

k=2

k=3

Número de Matrizes



Algoritmo de Multiplicação de Matrizes

```
MATRIX-CHAIN-ORDER(d0…dn+1)
para i←1 até n faça
    M[i,i] ← 0
para l←2 até n faça
    para i←1 até n-l+1 faça
        j ← i+l-1
        M[i,j] ← ∞
        para k←i até j-1 faça
            q ← M[i,k]+M[k+1,j]+di-1dkdj
            se q < M[i,j] então
            M[i,j] ← q
            c[i,j] ← k
retorne M, c
```

Custo mínimo para cadeias de comprimento 1

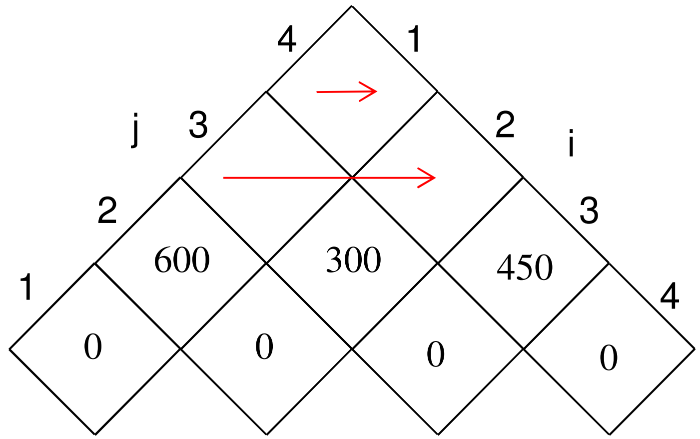
Custo mínimo para cadeias de comprimento l=2 (*m[i,i+1]*), l=3 (*m[i,i+2]*), l=4 (*m[i,i+3]*),..., l=n (*m[i,i+n]*)

Exemplo (1)

Matrizes		A_{10x20}	B_{20x3}	C_{3x5}	D_{5x30}
	d0	d1	d2	d3	d4
Dimensão	10	20	3	5	30

$A_i = d_{i-1} \times d_i$
$A_1 = d_0 \times d_1$
$A_2 = d_1 \times d_2$
...

```
Ordem(d0...dn)
para i←1 até n faça        } Subproblema de tamanho 1
    M[i,i] ←0
para l←2 até n faça
    para i←1 até n−l+1 faça
        j ←i+l−1
        M[i,j] ←∞
        para k←i até j−1 faça
            q ←M[i,k]+M[k+1,j]+d_{i−1}d_k d_j
            se q < M[i,j] então
                M[i,j] ←q
                c[i,j] ←k
retorne M, c
```



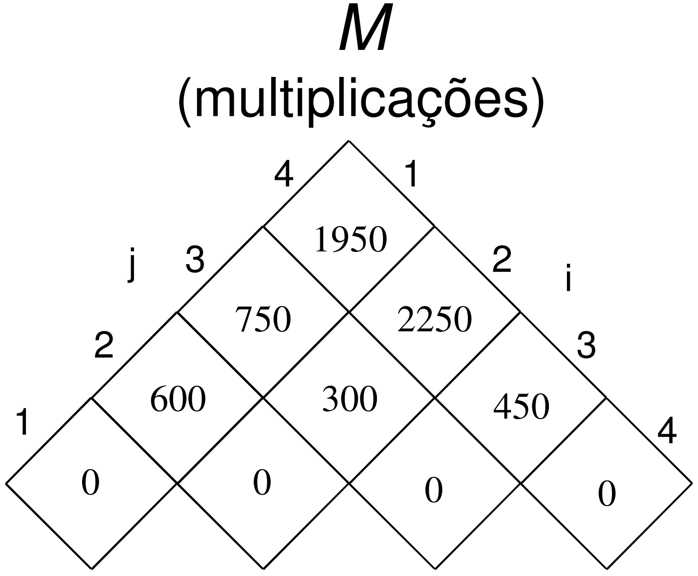
q=M[1,1]+M[2,2]+10x20x3=600

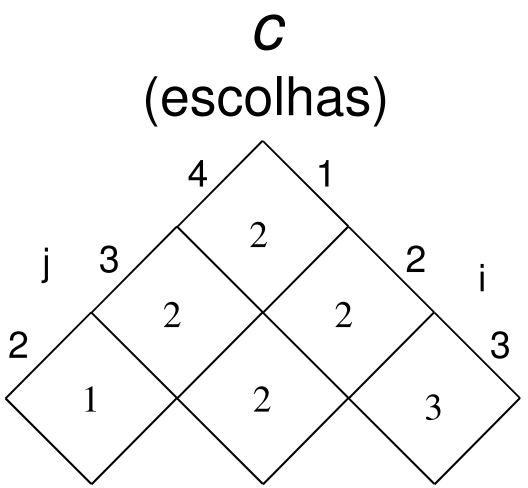
q=M[2,2]+M[3,3]+20x3x5=300

q=M[3,3]+M[4,4]+3x5x30=450

Exemplo (2)

- Ao fim da execução
  - *M[1,n]* contêm o valor da solução ótima
  - *c* as escolhas ótimas de *k* para cada subproblema
  - *c* guarda os resultados obtidos a cada solução!
  - Resultado completo do exemplo do slide anterior





Análise de Complexidade

- Assim, podemos armazenar as soluções ótimas de cada $\Theta(n^2)$ subproblema e reutilizá-las na solução dos problemas maiores
  - Estas soluções podem ser armazenadas em um arranjo *M[1..n,1..n]*
- O custo computacional de cada M[i,j] é O(n)
- Tempo de Execução: $O(n^3)$
- Melhoramos de expoencial para polinomial!!!

Imprimindo o Resultado

- A chamada inicial PRINT-OPTIMAL-PARENS(c,1,n) imprime uma colocação ótima dos paratênses de $<A_1,A_2,...,A_n>$

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

- A chamada PRINT-OPTIMAL-PARENS(c,1,4) imprime $((A_1A_2)(A_3A_4))$

Memoização (1)

- Computar solução usando solução recursiva inicial, mas guardando os valores da matriz já computados para evitar processamento desnecessário
- Adaptar a solução recursiva para tabular as soluções intermediárias
- As chamadas recursivas continuam, mas não precisam necessariamente executar as operações custosas
- No exemplo das matrizes
  - Iniciar os elementos de M com ∞ e executar **Busca-Sequência**(d, i, j)

Memoização (2)

```
Busca-Sequência(d,i,j)
se M[i,j] < ∞ então
    retorna m[i,j]
se i=j então
    m[i,j] ←0
senão para k ←i até j-1 faça
        q ←Busca-Sequência(d,i,k)+
    Busca-Sequência(d,k+1,j)+d_{i-1}d_kd_j
        se q < M[i,j] então
            M[i,j] ←q
retorne M[i,j]
```

Programação Dinâmica (1)

- Em geral, para aplicar PD, alguns aspectos devem ser considerados:
  - 1. Sub-estrutura ótima: Uma solução ótima para o problema deve ser composta de soluções ótimas para os seus subproblemas
  - 2. Escrever uma recorrência para determinar o valor da solução ótima
    - $M_{ótima} = \min_{\text{sobre todas as escolhas de } k}$ {(Soma de $M_{ótima}$ de todos os subproblemas que resultam da escolha de k)+ (custo associado com a escolha de k)}
    - Mostrar que o número de instâncias distintas dos subproblemas é limitado por um polinômio

Programação Dinâmica (2)

- 3. Computar o valor da solução ótima de maneira *bottom-up*, de forma que todos os sub-resultados já estejam pré-computados
  - Verificar se é possível reduzir espaço eliminando sub-resultados que não são mais necessários
  - Construir a solução ótima a partir da informação pré-computada

Maior Subseqüência Comum

- São dadas duas cadeias de caracteres
- Deseja-se estabelecer o quão semelhante elas são
  - Comparação de seqüências de DNA
  - Correção ortográfica
- Uma medida de semelhança é o comprimento da Maior Subseqüência Comum (MSC) entre os dois

Exemplo da Cadeia de DNA

- Uma cadeia de DNA consiste em uma cadeia de moléculas chamadas **bases** que são
  - adenina, guanina, citosina e timina
    - pode ser expressada por {A, C, G, T}
  - DNA de dois organismos S1 = {ACCGGTCGAGTGCGCGGAAGCCGGCCGAA} e S2 = {GTCGTTCGGAATGCCGTTGCTCTGTAAA}
- Comparação de duas cadeias de DNA consiste em determinar o quanto as duas cadeias são semelhantes
  - Medida de quanto os dois **organismos** estão **intimamente relacionados**

MSC: Definição

- *Z* é uma subseqüência de X, se é possível gerar Z removendo alguns caracteres de X
  - *X* ="ACGGTTA", *Y*="CGTAT",
  - MSC(*X*,*Y*) = "CGTA" ou "CGTT"
- Dada uma sequência $X=<x_1,..., x_n>$, outra sequência $Z=<z_1,..., z_n>$, *Z* é uma subsequência de *X* se existe uma sequência crescente $<i_1,...,i_n>$ de índices de *X*, para todo $j=1,..., k$, temos $x_{ij} = z_j$
  - *Z =<B,C,D,B>* é uma subsequência de *X=<A,B,C,B, D,A,B> com sequência de índices <2,3,5,7>*

Exemplo da Cadeia de DNA

- Em nosso exemplo, a cadeia mais longa é
  - S1 = "ACCG**GTCG**AG**T**G**CG**CG**GAAGCCGGCCGAA**"
  - S2 = "GTCGTTCGGAATGCCGTTGCTCTGTAAA"
  - MSC(*X*, *Y*) = "GTCGTCGGAAGCCGGCCGAA"
  - Índice de S1 = <5, 6, 7, 8, 11, 13, 14, 17, 18,..., 29>

Usando a Força Bruta

- Enumerar todas as subsequências de X e conferir cada subsequência para ver se ela também é uma subsequência de Y
  - Neste caso, deve-se armazenar a subsequência mais longa encontrada
- Cada subsequência de *X* corresponde a um subconjunto dos índices $\{1, 2, \dots, m\}$
- Existem 2^m subsequências de X; assim essa abordagem exige tempo exponencial
  - o que torna impraticável para longas sequências

MSC: Subestrutura Ótima

- Etapa 1: Caracterização de uma subsequência comum mais longa
- Seja X_m="$x_1 x_2 \ldots x_m$" e Y_n="$y_1 y_2 \ldots y_n$"
  - Se $x_m = y_n$, inclua este caractere no início de Z e encontre MSC (X_{m-1}, Y_{n-1})
    - Ou seja, o algoritmo "pula" para o próximo caractere das duas sequências
  - Se $x_m \neq y_n$,
    - Pular um caractere de X ou de Y
    - Decidir o que fazer comparando MSC(X_m, Y_{n-1}) e MSC(X_{m-1}, Y_n)

MSC: Recorrência

- Etapa 2: Uma solução recursiva para uma subsequência comum mais longa
- Seja *c[i,j] = MSC(X_i, Y_j)*

uma das sequências tem comprimento 0

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

c[i,j] define o comprimento de uma MSC das sequências X_i e Y_j

MSC: Algoritmo

```
MSC(X, Y, m, n)
para i←1 até m faça
    c[i,0] ← 0
para j←0 até n faça
    c[0,j] ← 0
para i←1 até m faça
    para j←1 até n faça
        se x_i = y_j então
            c[i,j] ← c[i-1,j-1]+1
            b[i,j] ← "↖"
        senão se c[i-1,j] ≥ c[i,j-1] então
                c[i,j] ← c[i-1,j]
                b[i,j] ← "↑"
            senão
                c[i,j] ← c[i,j-1]
                b[i,j] ← "←"
retorne c, b
```

Exemplo (1)

- Mostre como a tabela seria construída para as seqüências ABCBDAB e BDCABA
- Qual o tamanho da MSC encontrada?
- Qual seria a MSC?

Exemplo (2)

X = "ABCBDAB"

Y = "BDCABA"

MSC = BCBA

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Imprimindo o Resultado

- O procedimento abaixo imprime uma MSC de X e Y na ordem direta apropriada
  - A chamada inicial é PRINT-MSC(b, X, comprimento[X], comprimento[Y])

```
PRINT-MSC(b, X, i, j)
if i=0 or j=0
  then return
if b[i,j] = ↖
  then PRINT-MSC(b, X, i-1, j-1)
    print x_i
else if b[i,j] = ↑
  then PRINT-MSC(b, X, i-1, j)
else PRINT-MSC(b, X, i, j-1)
```

O procedimento imprime "BCBA"