MC458 — Projeto e Análise de Algoritmos I

C.C. de Souza   C.N. da Silva   O. Lee

Antes de mais nada...

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- O que vocês tem em mãos é uma versão modificada preparada para atender a meus gostos.
- Nunca é demais enfatizar que o material é apenas um guia e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial o CLR ou CLRS).

Orlando Lee

Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram direta ou indiretamente com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Uma lista destes "colaboradores" (em ordem alfabética) é dada abaixo:
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

Programação Dinâmica

Resumo

Para projetar um algoritmo de programação dinâmica seguimos os seguintes passos:

Resumo

Para projetar um algoritmo de programação dinâmica seguimos os seguintes passos:

1. Subestrutura ótima: caracterizamos a estrutura de uma solução ótima.

Resumo

Para projetar um algoritmo de programação dinâmica seguimos os seguintes passos:

1. Subestrutura ótima: caracterizamos a estrutura de uma solução ótima.
2. Recorrência: recursivamente definimos o valor de uma solução ótima.

Resumo

Para projetar um algoritmo de programação dinâmica seguimos os seguintes passos:

1. Subestrutura ótima: caracterizamos a estrutura de uma solução ótima.
2. Recorrência: recursivamente definimos o valor de uma solução ótima.
3. Algoritmo: computamos o valor de uma solução ótima, de modo bottom-up.

Resumo

Para projetar um algoritmo de programação dinâmica seguimos os seguintes passos:

1. Subestrutura ótima: caracterizamos a estrutura de uma solução ótima.
2. Recorrência: recursivamente definimos o valor de uma solução ótima.
3. Algoritmo: computamos o valor de uma solução ótima, de modo bottom-up.
4. Reconstrução: construímos uma solução ótima, a partir da informação computada.

Quando queremos apenas o valor ótimo, omitimos o passo (4). Se quisermos executar o passo (4) temos que adaptar o passo (3) para lembrar da escolha que leva a uma solução ótima de cada subproblema.

Problema: Multiplicação de Matrizes

Calcular o número mínimo de multiplicações escalares necessários para computar a matriz M dada por:

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

onde M_i é uma matriz de b_{i-1} linhas e b_i colunas, para todo $i \in \{1, \ldots, n\}$.

Multiplicação de Cadeia de Matrizes

Problema: Multiplicação de Matrizes

Calcular o número mínimo de multiplicações escalares necessários para computar a matriz M dada por:

$$M = M_1 \times M_2 \times \dots M_i \dots \times M_n$$

onde M_i é uma matriz de b_{i-1} linhas e b_i colunas, para todo $i \in \{1, \dots, n\}$.

- Matrizes são multiplicadas aos pares sempre. Então, é preciso encontrar uma parentização ótima para a cadeia de matrizes.

Multiplicação de Cadeia de Matrizes

Problema: Multiplicação de Matrizes

Calcular o número mínimo de multiplicações escalares necessários para computar a matriz M dada por:

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

onde M_i é uma matriz de b_{i-1} linhas e b_i colunas, para todo $i \in \{1, \ldots, n\}$.

- Matrizes são multiplicadas aos pares sempre. Então, é preciso encontrar uma parentização ótima para a cadeia de matrizes.
- O cálculo de $M_i \times M_{i+1}$ pode ser feito com $b_{i-1} b_i b_{i+1}$ multiplicações envolvendo os elementos de M_i e M_{i+1} (usando o algoritmo tradicional de multiplicação de matrizes).

Multiplicação de Cadeia de Matrizes

- **Exemplo:** Qual é o número mínimo de multiplicações escalares necessárias para computar $M = M_1 \times M_2 \times M_3 \times M_4$ com $b = \{200, 2, 30, 20, 5\}$?

Multiplicação de Cadeia de Matrizes

- **Exemplo:** Qual é o número mínimo de multiplicações escalares necessárias para computar $M = M_1 \times M_2 \times M_3 \times M_4$ com $b = \{200, 2, 30, 20, 5\}$?
- As possibilidades de parentização são:

$$
\begin{array}{lcl}
M = (M_1 \times (M_2 \times (M_3 \times M_4))) & \rightarrow & 5.300 \text{ multiplicações} \\
M = (M_1 \times ((M_2 \times M_3) \times M_4)) & \rightarrow & 3.400 \text{ multiplicações} \\
M = ((M_1 \times M_2) \times (M_3 \times M_4)) & \rightarrow & 4.500 \text{ multiplicações} \\
M = ((M_1 \times (M_2 \times M_3)) \times M_4) & \rightarrow & 29.200 \text{ multiplicações} \\
M = (((M_1 \times M_2) \times M_3) \times M_4) & \rightarrow & 152.000 \text{ multiplicações}
\end{array}
$$

Multiplicação de Cadeia de Matrizes

- **Exemplo:** Qual é o número mínimo de multiplicações escalares necessárias para computar $M = M_1 \times M_2 \times M_3 \times M_4$ com $b = \{200, 2, 30, 20, 5\}$?
- As possibilidades de parentização são:

$$
\begin{array}{lcl}
M = (M_1 \times (M_2 \times (M_3 \times M_4))) & \rightarrow & 5.300 \text{ multiplicações} \\
M = (M_1 \times ((M_2 \times M_3) \times M_4)) & \rightarrow & 3.400 \text{ multiplicações} \\
M = ((M_1 \times M_2) \times (M_3 \times M_4)) & \rightarrow & 4.500 \text{ multiplicações} \\
M = ((M_1 \times (M_2 \times M_3)) \times M_4) & \rightarrow & 29.200 \text{ multiplicações} \\
M = (((M_1 \times M_2) \times M_3) \times M_4) & \rightarrow & 152.000 \text{ multiplicações}
\end{array}
$$

- A ordem das multiplicações faz **muita** diferença.

- Poderíamos calcular o número de multiplicações para todas as possíveis parentizações.

Multiplicação de Cadeia de Matrizes

- Poderíamos calcular o número de multiplicações para todas as possíveis parentizações.
- O número de possíveis parentizações é dado pela recorrência:

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & n > 1, \end{cases}$$

$$(M_1 \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_n)$$

Multiplicação de Cadeia de Matrizes

- Poderíamos calcular o número de multiplicações para todas as possíveis parentizações.
- O número de possíveis parentizações é dado pela recorrência:

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & n > 1, \end{cases}$$

$$(M_1 \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_n)$$

- Entretanto, pode-se mostrar que $P(n) \in \Omega(4^n / n^{\frac{3}{2}})$.

- Poderíamos calcular o número de multiplicações para todas as possíveis parentizações.
- O número de possíveis parentizações é dado pela recorrência:

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & n > 1, \end{cases}$$

$$(M_1 \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_n)$$

- Entretanto, pode-se mostrar que $P(n) \in \Omega(4^n / n^{\frac{3}{2}})$.
- Assim, a estratégia de força bruta é impraticável!

Subestrutura ótima (Passo 1)

- Para todo (i, j) tal que $1 \leq i \leq j \leq n$, considere o seguinte produto de matrizes:

$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

Subestrutura ótima (Passo 1)

- Para todo (i, j) tal que $1 \leq i \leq j \leq n$, considere o seguinte produto de matrizes:
$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

- Dada uma parentização ótima, existem dois pares de parênteses que identificam o último par de matrizes que serão multiplicadas. Ou seja, existe k tal que $M = (M_{1,k}) \times (M_{k+1,n})$.

Subestrutura ótima (Passo 1)

- Para todo (i, j) tal que $1 \leq i \leq j \leq n$, considere o seguinte produto de matrizes:
$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

- Dada uma parentização ótima, existem dois pares de parênteses que identificam o último par de matrizes que serão multiplicadas.
Ou seja, existe k tal que $M = (M_{1,k}) \times (M_{k+1,n})$.
- Como a parentização de M é ótima, as parentizações no cálculo de $M_{1,k}$ e $M_{k+1,n}$ devem ser ótimas. (Por quê?)

Subestrutura ótima (Passo 1)

- Para todo (i, j) tal que $1 \leq i \leq j \leq n$, considere o seguinte produto de matrizes:
$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

- Dada uma parentização ótima, existem dois pares de parênteses que identificam o último par de matrizes que serão multiplicadas. Ou seja, existe k tal que $M = (M_{1,k}) \times (M_{k+1,n})$.
- Como a parentização de M é ótima, as parentizações no cálculo de $M_{1,k}$ e $M_{k+1,n}$ devem ser ótimas. (Por quê?)
- Eis a subestrutura ótima do problema: a parentização ótima de $M_{1,n}$ inclui a parentização ótima de $M_{1,k}$ e $M_{k+1,n}$.

Subestrutura ótima (Passo 1)

- Seja $m[i, j]$ o número (mínimo) de multiplicações em uma parentização ótima de

$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

Assim, o que queremos determinar é $m[1, n]$.

Subestrutura ótima (Passo 1)

- Seja $m[i, j]$ o número (mínimo) de multiplicações em uma parentização ótima de

$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

  Assim, o que queremos determinar é $m[1, n]$.

- Se uma parentização ótima de $M_{1,n}$ for da forma $M = (M_{1,k}) \times (M_{k+1,n})$, então:

$$m[1, n] = m[1, k] + m[k + 1, n] + b_0 b_k b_n.$$

Subestrutura ótima (Passo 1)

- Seja $m[i, j]$ o número (mínimo) de multiplicações em uma parentização ótima de
$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$
Assim, o que queremos determinar é $m[1, n]$.
- Se uma parentização ótima de $M_{1,n}$ for da forma $M = (M_{1,k}) \times (M_{k+1,n})$, então:
$$m[1, n] = m[1, k] + m[k + 1, n] + b_0 b_k b_n.$$
- A dificuldade é que não conhecemos k, o ponto em que é feita a última multiplicação entre matrizes. Assim, é preciso testar todos os valores $k = 1, 2, \ldots, n - 1$. Logo,

Subestrutura ótima (Passo 1)

- Seja $m[i, j]$ o número (mínimo) de multiplicações em uma parentização ótima de
$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$
Assim, o que queremos determinar é $m[1, n]$.
- Se uma parentização ótima de $M_{1,n}$ for da forma $M = (M_{1,k}) \times (M_{k+1,n})$, então:
$$m[1, n] = m[1, k] + m[k + 1, n] + b_0 b_k b_n.$$
- A dificuldade é que não conhecemos k, o ponto em que é feita a última multiplicação entre matrizes. Assim, é preciso testar todos os valores $k = 1, 2, \ldots, n - 1$. Logo,
$$m[1, n] = \min_{1 \le k < n} \{m[1, k] + m[k + 1, n] + b_0 b_k b_n\}.$$

Recorrência (Passo 2)

- De forma geral, precisamos calcular $m[i, j]$ para todo (i, j) tal que $1 \leq i \leq j \leq n$. Esses são os subproblemas do problema original.

Recorrência (Passo 2)

- De forma geral, precisamos calcular $m[i, j]$ para todo (i, j) tal que $1 \leq i \leq j \leq n$. Esses são os subproblemas do problema original.
- Note que $m[i, i] = 0$ para $i = 1, 2, \ldots, n$.

Recorrência (Passo 2)

- De forma geral, precisamos calcular $m[i,j]$ para todo (i,j) tal que $1 \le i \le j \le n$. Esses são os subproblemas do problema original.
- Note que $m[i,i] = 0$ para $i = 1, 2, \ldots, n$.
- Para $i < j$, existe um índice k, com $i \le k < j$, tal que:

$$m[i,j] = m[i,k] + m[k+1,j] + b_{i-1} b_k b_j.$$

- De forma geral, precisamos calcular $m[i,j]$ para todo (i,j) tal que $1 \leq i \leq j \leq n$. Esses são os subproblemas do problema original.
- Note que $m[i,i] = 0$ para $i = 1, 2, \ldots, n$.
- Para $i < j$, existe um índice k, com $i \leq k < j$, tal que:

$$m[i,j] = m[i,k] + m[k+1,j] + b_{i-1} b_k b_j.$$

- Portanto,

$$m[i,j] = \min_{i \leq k < j} \{m[i,k] + m[k+1,j] + b_{i-1} b_k b_j\}.$$

Algoritmos baseados na recorrência

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_kb_j\}, & \text{se } i < j. \end{cases}$$

Veremos três tipos de algoritmos:

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1} b_k b_j\}, & \text{se } i < j. \end{cases}$$

Veremos três tipos de algoritmos:

- bottom-up eficiente (programação dinâmica – Passo 3)

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_kb_j\}, & \text{se } i < j. \end{cases}$$

Veremos três tipos de algoritmos:

- bottom-up eficiente (programação dinâmica – Passo 3)
- recursivo ineficiente (topdown com sobreposição)

Algoritmos baseados na recorrência

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1} b_k b_j\}, & \text{se } i < j. \end{cases}$$

Veremos três tipos de algoritmos:

- bottom-up eficiente (programação dinâmica – Passo 3)
- recursivo ineficiente (topdown com sobreposição)
- recursivo eficiente (topdown com memorização)

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_k b_j\}, & \text{se } i < j. \end{cases}$$

Nos algoritmos que veremos, usaremos uma variável $s[i,j]$ para armazenar o índice k que minimiza a expressão:

$$\min_{i \le k \le j} m[i,k] + m[k+1,j] + b_{i-1}b_k b_j,$$

ou seja, tomamos $s[i,j] = k$ se

Algoritmos baseados na recorrência

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_k b_j\}, & \text{se } i < j. \end{cases}$$

Nos algoritmos que veremos, usaremos uma variável $s[i,j]$ para armazenar o índice k que minimiza a expressão:

$$\min_{i \leq k \leq j} m[i,k] + m[k+1,j] + b_{i-1}b_k b_j,$$

ou seja, tomamos $s[i,j] = k$ se

$$m[i,j] = m[i,k] + m[k+1,j] + b_{i-1}b_k b_j.$$

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1} b_k b_j\}, & \text{se } i < j. \end{cases}$$

- O número de subproblemas $m[i,j]$, $1 \le i \le j \le n$ é $\Theta(n^2)$.

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_k b_j\}, & \text{se } i < j. \end{cases}$$

- O número de subproblemas $m[i,j]$, $1 \le i \le j \le n$ é $\Theta(n^2)$.
- Queremos resolver os subproblemas em uma ordem que evite o recálculo de subproblemas.

Programação dinâmica – Passo 3

A recorrência geral é:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_k b_j\}, & \text{se } i < j. \end{cases}$$

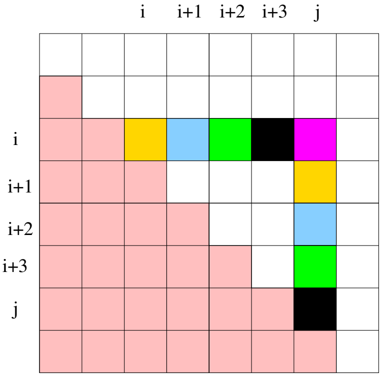
- O número de subproblemas $m[i,j]$, $1 \le i \le j \le n$ é $\Theta(n^2)$.
- Queremos resolver os subproblemas em uma ordem que evite o recálculo de subproblemas.
- Observando a recorrência, vemos que se ℓ é o tamanho da cadeia $M_{i,j}$, então $m[i,j]$ depende de subproblemas cujas cadeias tem tamanho menor que ℓ.

PD-MIN-MULT-MATRIZ(b, n)

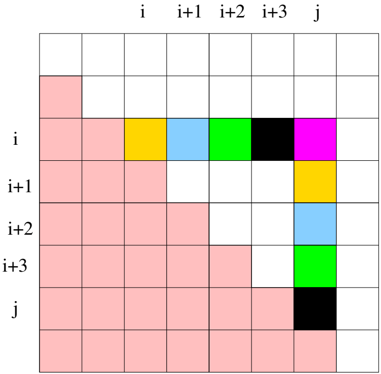
1. **para** $i \leftarrow 1$ **até** n **faça**
2 $\quad m[i,i] \leftarrow 0$ $\qquad \triangleright$ subproblemas de tamanho 1
3. **para** $\ell \leftarrow 2$ **até** n **faça** $\qquad \triangleright$ ℓ é o tamanho da cadeia $M_{i,j}$
4. $\quad$ **para** $i \leftarrow 1$ **até** $n - \ell + 1$ **faça**
5. $\quad\quad j \leftarrow i + \ell - 1$ $\qquad \triangleright$ $\ell = j - i + 1$
6. $\quad\quad m[i,j] \leftarrow \infty$
7. $\quad\quad$ **para** $k \leftarrow i$ **até** $j - 1$ **faça**
8. $\quad\quad\quad q \leftarrow m[i,k] + m[k+1,j] + b_{i-1} b_k b_j$
9. $\quad\quad\quad$ **se** $q < m[i,j]$ **então**
10. $\quad\quad\quad\quad m[i,j] \leftarrow q$
11. $\quad\quad\quad\quad s[i,j] \leftarrow k$
12. **devolva** (m, s)

Invariante: no início de cada iteração da linha 2, o valor $m[i,j]$ é o valor ótimo do subproblema $M_{i,j}$ se $j - i + 1 < \ell$ (tamanho $< \ell$).

Programação Dinâmica - Exemplo

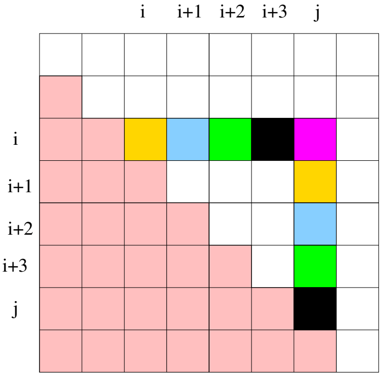


Programação Dinâmica - Exemplo

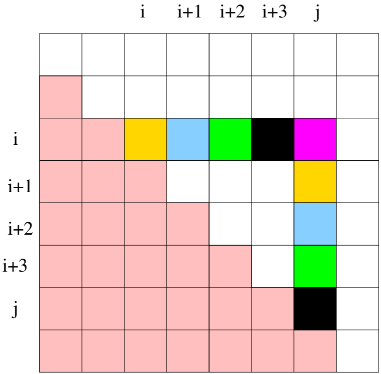


- Posições "abaixo" da diagonal principal não são usadas pela matriz m.

Programação Dinâmica - Exemplo

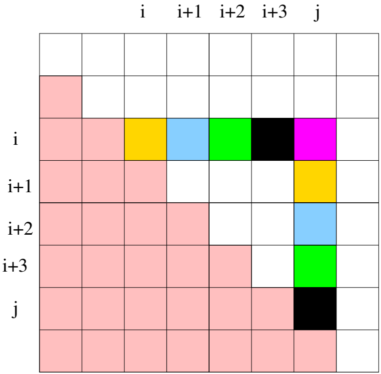


- Posições "abaixo" da diagonal principal não são usadas pela matriz m.
- A diagonal principal contém as soluções $m[i, i] = 0$ dos subproblemas $M_{i,i}$ de tamanho 1.

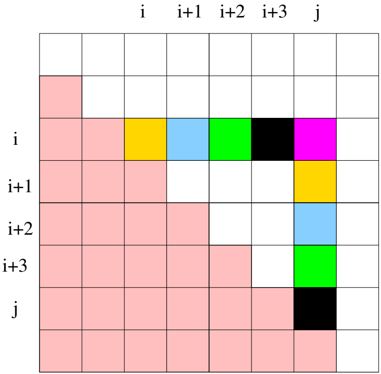


- Posições "abaixo" da diagonal principal não são usadas pela matriz m.
- A diagonal principal contém as soluções $m[i,i] = 0$ dos subproblemas $M_{i,i}$ de tamanho 1.
- A diagonal que vai de $m[1,2]$ a $m[n-1,n]$ contém os valores $m[i,i+1]$ dos subproblemas $M_{i,i+1}$ de tamanho 2.

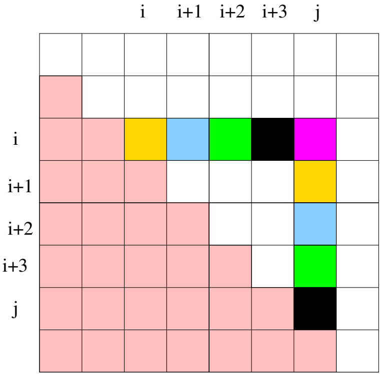
Programação Dinâmica - Exemplo



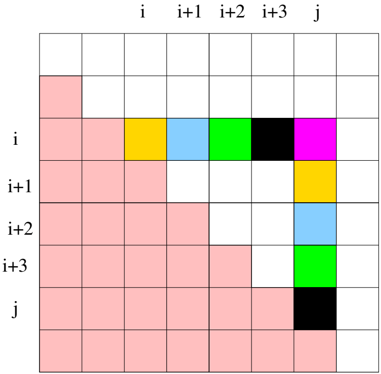
Programação Dinâmica - Exemplo



- A diagonal que vai de $m[1,3]$ a $m[n-2,n]$ contém os valores $m[i,i+2]$ dos subproblemas $M_{i,i+2}$ de tamanho 3.

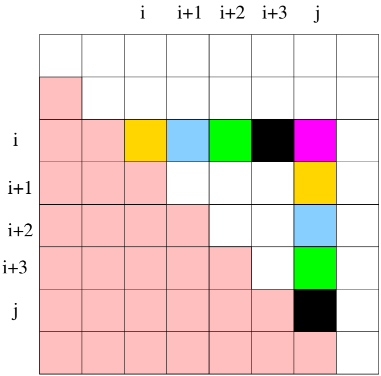


- A diagonal que vai de $m[1,3]$ a $m[n-2,n]$ contém os valores $m[i,i+2]$ dos subproblemas $M_{i,i+2}$ de tamanho 3.
- O mesmo vale para as outras diagonais.



- A diagonal que vai de $m[1,3]$ a $m[n-2,n]$ contém os valores $m[i,i+2]$ dos subproblemas $M_{i,i+2}$ de tamanho 3.
- O mesmo vale para as outras diagonais.
- O valor $m[1,n]$ está na última diagonal.

Programação Dinâmica - Exemplo



$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1}b_k b_j\}, & \text{se } i < j. \end{cases}$$

Programação Dinâmica - Exemplo

m

	1	2	3	4
1	0			
2		0		
3			0	
4				0

s

	1	2	3	4
1	_			
2		_		
3			_	
4				_

{ 200, 2, 30, 20, 5 }

Programação Dinâmica - Exemplo

	1	2	3	4
1	0	12000		
2		0	1200	
3			0	3000
4				0

m

	1	2	3	4
1	–	1		
2		–	2	
3			–	3
4				–

s

{ 200, 2, 30, 20, 5 }

Programação Dinâmica - Exemplo

	1	2	3	4
1	0	12000	9200	
2		0	1200	
3			0	3000
4				0

m

	1	2	3	4
1	_	1	1	
2		_	2	
3			_	3
4				_

s

{ 200, 2, 30, 20, 5 }

b0*b1*b3=200*2*20=8000

b0*b2*b3=200*30*20=120000

Programação Dinâmica - Exemplo

	1	2	3	4
1	0	12000	9200	
2		0	1200	1400
3			0	3000
4				0

m

	1	2	3	4
1	_	1	1	
2		_	2	3
3			_	3
4				_

s

{ 200, 2, 30, 20, 5 }

b1*b2*b4=2*30*5=300

b1*b3*b4=2*20*5=200

	1	2	3	4
1	0	12000	9200	3400
2		0	1200	1400
3			0	3400
4				0

m

	1	2	3	4
1	_	1	1	1
2		_	2	3
3			_	3
4				_

s

{ 200, 2, 30, 20, 5 }

b0*b1*b4=200*2*5=2000

b0*b2*b4=200*30*5=30000

b0*b3*b4=200*20*5=20000

	1	2	3	4
1	0	12000	9200	3400
2		0	1200	1400
3			0	3000
4				0

m

	1	2	3	4
1	_	1	1	1
2		_	2	3
3			_	3
4				_

s

M1 ((M2 . M3) . M4)

PD-Min-Mult-Matriz(b, n)

1. **para** $i \leftarrow 1$ **até** n **faça**
2 $\quad m[i,i] \leftarrow 0$ $\qquad \triangleright$ subproblemas de tamanho 1
3. **para** $\ell \leftarrow 2$ **até** n **faça** $\qquad \triangleright$ ℓ é o tamanho da cadeia $M_{i,j}$
4. $\quad$ **para** $i = 1$ **até** $n - \ell + 1$ **faça**
5. $\qquad j \leftarrow i + \ell - 1$ $\qquad \triangleright \ell = j - i + 1$
6. $\qquad m[i,j] \leftarrow \infty$
7. $\qquad$ **para** $k \leftarrow i$ **até** $j - 1$ **faça**
8. $\qquad\quad q \leftarrow m[i,k] + m[k+1,j] + b_{i-1} b_k b_j$
9. $\qquad\quad$ **se** $q < m[i,j]$ **então**
10. $\qquad\qquad m[i,j] \leftarrow q$
11. $\qquad\qquad s[i,j] \leftarrow k$
12. **devolva** (m, s)

Os laços encaixados nas linhas 3, 4 e 7 mostram que a complexidade do algoritmo é $O(n^3)$.
O algoritmo usa espaço $\Theta(n^2)$ para armazenar m e s.

Construção de uma solução ótima – Passo 4

- PD-MIN-MULT-MATRIZ calcula o número mínimo de multiplicações para calcular M mas não nos fornece diretamente uma parentização ótima.

Construção de uma solução ótima – Passo 4

- PD-MIN-MULT-MATRIZ calcula o número mínimo de multiplicações para calcular M mas não nos fornece diretamente uma parentização ótima.
- Entretanto, a tabela $s[1\,..\,n-1][2\,..\,n]$ nos fornece informação suficiente para fazer isto.

Construção de uma solução ótima – Passo 4

- PD-MIN-MULT-MATRIZ calcula o número mínimo de multiplicações para calcular M mas não nos fornece diretamente uma parentização ótima.
- Entretanto, a tabela $s[1\,..\,n-1][2\,..\,n]$ nos fornece informação suficiente para fazer isto.
- Lembre-se que se $s[i,j]=k$ então

$$m[i,j] = m[i,k] + m[k+1,j] + b_{i-1}b_kb_j,$$

ou seja, k é a escolha que leva a uma solução ótima.

Construção de uma solução ótima – Passo 4

Imprime-Parentização-Ótima(s, i, j)
se $i = j$ **então**
    imprima "M_i"
senão imprima "("
        Imprime-Parentização-Ótima$(s, i, s[i, j])$
        Imprime-Parentização-Ótima$(s, s[i, j] + 1, j)$
senão imprima ")"

Para o exemplo apresentado, o algoritmo imprime $(M_1((M_2 M_3) M_4))$.

Subsestrutura ótima: como achar?

Subsestrutura ótima: como achar?

- No problema do corte de barra vimos que uma maneira ótima de cortar uma barra de comprimento n envolve fazer um corte ótimo em dois pedaços de tamanho i e $n - i$. A solução então era usar o pedaço de comprimento i e os pedaços de um corte ótimo da barra de comprimento $n - i$.

Subsestrutura ótima: como achar?

- No problema do corte de barra vimos que uma maneira ótima de cortar uma barra de comprimento n envolve fazer um corte ótimo em dois pedaços de tamanho i e $n - i$. A solução então era usar o pedaço de comprimento i e os pedaços de um corte ótimo da barra de comprimento $n - i$.
- No problema de multiplicação de cadeia de matrizes vimos que uma parentização ótima de $M_i \times \cdots \times M_j$ envolve separar o produto entre M_k e M_{k+1}. A solução então consistia em usar a parentização ótima de $M_i \times \cdots \times M_k$ e de $M_{k+1} \times \cdots \times M_j$.

Subsestrutura ótima: como achar?

Ao tentar resolver problemas deste tipo, você perceberá o seguinte padrão comum para descobrir a subestrutura ótima:

Subsestrutura ótima: como achar?

Ao tentar resolver problemas deste tipo, você perceberá o seguinte padrão comum para descobrir a subestrutura ótima:

- A solução ótima consiste em fazer uma escolha que leva a um ou mais subproblemas.

Subsestrutura ótima: como achar?

Ao tentar resolver problemas deste tipo, você perceberá o seguinte padrão comum para descobrir a subestrutura ótima:

- A solução ótima consiste em fazer uma escolha que leva a um ou mais subproblemas.
- Suponha que a escolha que leva a uma solução ótima é dada a você (ela cai do céu!).

Subsestrutura ótima: como achar?

Ao tentar resolver problemas deste tipo, você perceberá o seguinte padrão comum para descobrir a subestrutura ótima:

- A solução ótima consiste em fazer uma escolha que leva a um ou mais subproblemas.
- Suponha que a escolha que leva a uma solução ótima é dada a você (ela cai do céu!).
- Dada esta escolha, você determina quais subproblemas você deve resolver e como caracterizar o espaço de subproblemas.

Subsestrutura ótima: como achar?

Ao tentar resolver problemas deste tipo, você perceberá o seguinte padrão comum para descobrir a subestrutura ótima:

- A solução ótima consiste em fazer uma escolha que leva a um ou mais subproblemas.
- Suponha que a escolha que leva a uma solução ótima é dada a você (ela cai do céu!).
- Dada esta escolha, você determina quais subproblemas você deve resolver e como caracterizar o espaço de subproblemas.
- Mostre que as soluções dos subproblemas usadas "dentro" de uma solução ótima devem ser soluções ótimas desssas. Para isto, use uma técnica "cut-and-paste": "se a solução do subproblema não fosse ótima, eu poderia trocá-la pela solução ótima deste, obtendo uma solução melhor para o o problema original (contradição)."

Subsestrutura ótima: como achar?

- Como escolher o espaço de subproblemas? Uma ideia é tentar mantê-lo tão simples quanto possível e expandi-lo apenas se for necessário.

Subsestrutura ótima: como achar?

- Como escolher o espaço de subproblemas? Uma ideia é tentar mantê-lo tão simples quanto possível e expandi-lo apenas se for necessário.
- No problema do corte de barra usamos simplesmente o comprimento da barra: $\Theta(n)$ subproblemas.

Subsestrutura ótima: como achar?

- Como escolher o espaço de subproblemas? Uma ideia é tentar mantê-lo tão simples quanto possível e expandi-lo apenas se for necessário.
- No problema do corte de barra usamos simplesmente o comprimento da barra: $\Theta(n)$ subproblemas.
- No problema de multiplicação de cadeia de matrizes, foi necessário considerar os subproblemas M_{ij} para $1 \leq i \leq j \leq n$. Não era suficiente considerar apenas subproblemas em que $i = 1$ ou que $j = n$. Isto resulta em $\Theta(n^2)$ subproblemas.

Multiplicação de Cadeia de Matrizes (Recursivo)

Solução recursiva ineficiente:

$$m[i,j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + b_{i-1} b_k b_j\}, & \text{se } i < j. \end{cases}$$

$\textsc{Min-Mult-Matriz-Rec}(b, i, j)$

1. **se** $i = j$ **então devolva** 0
2. $m[i,j] := \infty$
3. **para** $k := i$ **até** $j - 1$ **faça**
4. $\quad q := \textsc{Min-Mult-Matriz-Rec}(b, i, k)$
   $\quad\quad + \textsc{Min-Mult-Matriz-Rec}(b, k+1, j)$
   $\quad\quad + b_{i-1} b_k b_j$
5. $\quad$ **se** $m[i,j] > q$ **então**
6. $\quad\quad m[i,j] := q$; $s[i,j] := k$
7. **devolva** $m[i,j]$.

Multiplicação de Cadeia de Matrizes (Recursivo)

$\textsc{Min-Mult-Matriz-Rec}(b, i, j)$

1. **se** $i = j$ **então devolva** 0
2. $m[i,j] := \infty$
3. **para** $k := i$ **até** $j - 1$ **faça**
4. $\quad q := \textsc{Min-Mult-Matriz-Rec}(b, i, k)$
 $\qquad + \textsc{Min-Mult-Matriz-Rec}(b, k+1, j)$
 $\qquad + b_{i-1} b_k b_j$
5. $\quad$ **se** $m[i,j] > q$ **então**
6. $\qquad m[i,j] := q$; $s[i,j] := k$
7. **devolva** $m[i,j]$.

Seja $T(n)$ o número de operações executadas pelo algoritmo no melhor caso. Então

Min-Mult-Matriz-Rec(b, i, j)

```
se i = j então devolva 0
m[i, j] := ∞
para k := i até j − 1 faça
    q := Min-Mult-Matriz-Rec(b, i, k)
         + Min-Mult-Matriz-Rec(b, k + 1, j)
         + b_{i−1} b_k b_j
    se m[i, j] > q então
        m[i, j] := q ; s[i, j] := k
devolva m[i, j].
```

Seja $T(n)$ o número de operações executadas pelo algoritmo no melhor caso. Então

$$T(n) \geq \begin{cases} 1, & \text{se } n = 1 \\ 1 + \sum_{k=1}^{n-1}[T(k) + T(n-k) + 1] & \text{se } n > 1, \end{cases}$$

- A complexidade de MIN-MULT-MATRIZ-REC no melhor caso é dada por:

$$T(n) \geq \begin{cases} 1, & \text{se } n = 1 \\ 1 + \sum_{k=1}^{n-1} [T(k) + T(n-k) + 1], & \text{se } n > 1, \end{cases}$$

Algoritmo Recursivo - Complexidade

- A complexidade de Min-Mult-Matriz-Rec no melhor caso é dada por:

$$T(n) \geq \begin{cases} 1, & \text{se } n = 1 \\ 1 + \sum_{k=1}^{n-1}[T(k) + T(n-k) + 1], & \text{se } n > 1, \end{cases}$$

- Portanto, $T(n) \geq 2\sum_{k=1}^{n-1} T(i) + n$, para $n > 1$.

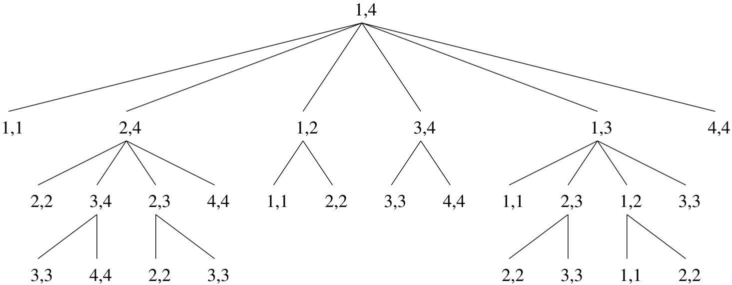
Algoritmo Recursivo - Complexidade

- A complexidade de MIN-MULT-MATRIZ-REC no melhor caso é dada por:

$$T(n) \geq \begin{cases} 1, & \text{se } n = 1 \\ 1 + \sum_{k=1}^{n-1}[T(k) + T(n-k) + 1], & \text{se } n > 1, \end{cases}$$

- Portanto, $T(n) \geq 2\sum_{k=1}^{n-1} T(i) + n$, para $n > 1$.
- Pode-se mostrar (por substituição) que $T(n) \geq 2^{n-1}$, ou seja, o algoritmo recursivo tem complexidade $\Omega(2^n)$ no melhor caso, o que é impraticável!

Algoritmo Recursivo



A árvore de recursão de MIN-MULT-MATRIZ-REC$(n, 1, 4)$ mostra que vários subproblemas são resolvidos mais de uma vez, causando a ineficiência do algoritmo. Por exemplo, $M_{1,2}$, $M_{2,3}$ e $M_{3,4}$.

Multiplicação de Cadeia de Matrizes (Memorização)

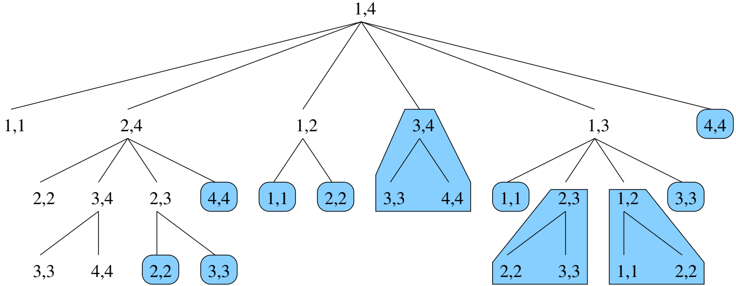
MEMO-MIN-MULT-MATRIZ(b, n)

para $i \leftarrow 1$ **até** n **faça**
  para $j \leftarrow 1$ **até** n **faça**
    $m[i, j] \leftarrow \infty$
devolva MEMO-MMM-AUX$(b, 1, n)$

MEMO-MMM-AUX(b, i, j)

se $m[i, j] < \infty$ **então devolva** $m[i, j]$
se $i = j$ **então** $m[i, j] := 0$
senão
  para $k := i$ **até** $j - 1$ **faça**
    $q :=$ MEMO-MMM-AUX(b, i, k)
      $+$ MEMO-MMM-AUX$(b, k+1, j) + b_{i-1} b_k b_j$
    se $m[i, j] > q$ **então** $m[i, j] \leftarrow q$; $s[i, j] \leftarrow k$
devolva $m[i, j]$

Algoritmo com Memorização



Na execução de MEMO-MIN-MULT-MATRIZ$(b, 4)$, as computações executadas nas árvores em azul são substituídas por uma simples consulta à tabela m.

$\textsc{Memo-MMM-Aux}(b, i, j)$

se $m[i, j] < \infty$ **então devolva** $m[i, j]$
se $i = j$ **então** $m[i, j] := 0$
senão
    para $k := i$ **até** $j - 1$ **faça**
        $q := \textsc{Memo-MMM-Aux}(b, i, k)$
            $+ \textsc{Memo-MMM-Aux}(b, k + 1, j) + b_{i-1} b_k b_j$
        se $m[i, j] > q$ **então** $m[i, j] \leftarrow q$; $s[i, j] \leftarrow k$
devolva $m[i, j]$

Multiplicação de Cadeia de Matrizes (Memorização)

$\textsc{Memo-MMM-Aux}(b, i, j)$

se $m[i, j] < \infty$ **então devolva** $m[i, j]$
se $i = j$ **então** $m[i, j] := 0$
senão
    para $k := i$ **até** $j - 1$ **faça**
        $q := \textsc{Memo-MMM-Aux}(b, i, k)$
            $+ \textsc{Memo-MMM-Aux}(b, k+1, j) + b_{i-1} b_k b_j$
        se $m[i, j] > q$ **então** $m[i, j] \leftarrow q$; $s[i, j] \leftarrow k$
devolva $m[i, j]$

Complexidade de $\textsc{Memo-MMM-Aux}$: ???.

Multiplicação de Cadeia de Matrizes (Memorização)

Memo-MMM-Aux(b, i, j)

se $m[i,j] < \infty$ **então devolva** $m[i,j]$
se $i = j$ **então** $m[i,j] := 0$
senão
  para $k := i$ **até** $j - 1$ **faça**
    $q :=$ Memo-MMM-Aux(b, i, k)
      $+$ Memo-MMM-Aux$(b, k+1, j) + b_{i-1} b_k b_j$
    se $m[i,j] > q$ **então** $m[i,j] \leftarrow q$; $s[i,j] \leftarrow k$
devolva $m[i,j]$

Há dois tipos de chamadas: (A) $m[i,j] = \infty$ e (B) $m[i,j] < \infty$

Multiplicação de Cadeia de Matrizes (Memorização)

$\textsc{Memo-MMM-Aux}(b, i, j)$

se $m[i,j] < \infty$ **então devolva** $m[i,j]$
se $i = j$ **então** $m[i,j] := 0$
senão
  para $k := i$ **até** $j - 1$ **faça**
    $q := \textsc{Memo-MMM-Aux}(b, i, k)$
      $+ \textsc{Memo-MMM-Aux}(b, k+1, j) + b_{i-1} b_k b_j$
    se $m[i,j] > q$ **então** $m[i,j] \leftarrow q$; $s[i,j] \leftarrow k$
devolva $m[i,j]$

Há dois tipos de chamadas: (A) $m[i,j] = \infty$ e (B) $m[i,j] < \infty$

Há $O(n^2)$ chamadas do tipo (A), uma para cada par i, j.

Multiplicação de Cadeia de Matrizes (Memorização)

$\text{MEMO-MMM-AUX}(b, i, j)$

1. **se** $m[i, j] < \infty$ **então devolva** $m[i, j]$
2. **se** $i = j$ **então** $m[i, j] := 0$
3. **senão**
4.   **para** $k := i$ **até** $j - 1$ **faça**
5. $q := \text{MEMO-MMM-AUX}(b, i, k)$
               $+ \text{MEMO-MMM-AUX}(b, k + 1, j) + b_{i-1} b_k b_j$
6.   **se** $m[i, j] > q$ **então** $m[i, j] \leftarrow q$; $s[i, j] \leftarrow k$
7. **devolva** $m[i, j]$

Há dois tipos de chamadas: (A) $m[i, j] = \infty$ e (B) $m[i, j] < \infty$

Há $O(n^2)$ chamadas do tipo (A), uma para cada par i, j.

As chamadas do tipo (B) são feitas dentro de chamadas do tipo (A). A função faz $O(n)$ chamadas dessas. Logo, há $O(n^3)$ chamadas do tipo (B).

Multiplicação de Cadeia de Matrizes (Memorização)

MEMO-MMM-AUX(b, i, j)

1. **se** $m[i,j] < \infty$ **então devolva** $m[i,j]$
2. **se** $i = j$ **então** $m[i,j] := 0$
3. **senão**
4. **para** $k := i$ **até** $j-1$ **faça**
5. $q :=$ MEMO-MMM-AUX(b, i, k) $+$ MEMO-MMM-AUX$(b, k+1, j) + b_{i-1} b_k b_j$
6. **se** $m[i,j] > q$ **então** $m[i,j] \leftarrow q$; $s[i,j] \leftarrow k$
7. **devolva** $m[i,j]$

Há dois tipos de chamadas: (A) $m[i,j] = \infty$ e (B) $m[i,j] < \infty$

Multiplicação de Cadeia de Matrizes (Memorização)

$\text{Memo-MMM-Aux}(b, i, j)$

1. **se** $m[i,j] < \infty$ **então devolva** $m[i,j]$
2. **se** $i = j$ **então** $m[i,j] := 0$
3. **senão**
4.   **para** $k := i$ **até** $j - 1$ **faça**
5. $q := \text{Memo-MMM-Aux}(b, i, k)$
               $+ \text{Memo-MMM-Aux}(b, k+1, j) + b_{i-1} b_k b_j$
6.   **se** $m[i,j] > q$ **então** $m[i,j] \leftarrow q$; $s[i,j] \leftarrow k$
7. **devolva** $m[i,j]$

Há dois tipos de chamadas: (A) $m[i,j] = \infty$ e (B) $m[i,j] < \infty$

$O(n^2)$ chamadas do tipo (A): cada uma custa $O(n)$ + recursão.

Multiplicação de Cadeia de Matrizes (Memorização)

$\textsc{Memo-MMM-Aux}(b, i, j)$

se $m[i,j] < \infty$ **então devolva** $m[i,j]$
se $i = j$ **então** $m[i,j] := 0$
senão
  para $k := i$ **até** $j - 1$ **faça**
    $q := \textsc{Memo-MMM-Aux}(b, i, k)$
      $+ \textsc{Memo-MMM-Aux}(b, k+1, j) + b_{i-1} b_k b_j$
    se $m[i,j] > q$ **então** $m[i,j] \leftarrow q$; $s[i,j] \leftarrow k$
devolva $m[i,j]$

Há dois tipos de chamadas: (A) $m[i,j] = \infty$ e (B) $m[i,j] < \infty$

$O(n^2)$ chamadas do tipo (A): cada uma custa $O(n)$ + recursão.

$O(n^3)$ chamadas do tipo (B): cada uma custa $O(1)$.

MEMO-MMM-AUX(b, i, j)
1. **se** $m[i,j] < \infty$ **então devolva** $m[i,j]$
2. **se** $i = j$ **então** $m[i,j] := 0$
3. **senão**
4.   **para** $k := i$ **até** $j-1$ **faça**
5. $q :=$ MEMO-MMM-AUX(b, i, k)
            $+$ MEMO-MMM-AUX$(b, k+1, j) + b_{i-1} b_k b_j$
6.   **se** $m[i,j] > q$ **então** $m[i,j] \leftarrow q$; $s[i,j] \leftarrow k$
7. **devolva** $m[i,j]$

Há dois tipos de chamadas: (A) $m[i,j] = \infty$ e (B) $m[i,j] < \infty$

$O(n^2)$ chamadas do tipo (A): cada uma custa $O(n)$ + recursão.

$O(n^3)$ chamadas do tipo (B): cada uma custa $O(1)$.

Complexidade: $O(n^3)$