AULA 15

Programação dinâmica

CLRS 15.1–15.3

Programacão dinâmica

= “recursão–com–tabela”
= transformação inteligente de recursão em iteração

EXEMPLO 1: **Números de Fibonacci**

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

Algoritmo recursivo para F_n:

```
FIBO-REC (n)
se n ≤ 1
   então devolva n
   senão a ← FIBO-REC (n − 1)
         b ← FIBO-REC (n − 2)
         devolva a + b
```

Consumo de tempo?
Número de somas:

$$\begin{aligned} T(0) &= 0 \\ T(1) &= 0 \\ T(n) &= T(n-1) + T(n-2) + 1 \quad \text{se } n \geq 2 \end{aligned}$$

Solução: $T(n) \geq 3^n/2^n$ para $n \geq 6$.

Prova: $T(6) = 12 > (3/2)^6$ e $T(7) = 20 > (3/2)^7$.
Se $n \geq 8$ então

$$\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\geq (3/2)^{n-1} + (3/2)^{n-2} + 1 \\
&= (3/2 + 1)\,(3/2)^{n-2} + 1 \\
&\geq (5/2)\,(3/2)^{n-2} \\
&\geq (9/4)\,(3/2)^{n-2} \\
&= (3/2)^2 (3/2)^{n-2} \\
&= (3/2)^n\,.
\end{aligned}$$

Consumo de tempo: $\Omega((\frac{3}{2})^n)$
Exponencial!

Algoritmo resolve o mesmo subproblema muitas vezes:

$$\begin{array}{ccccccccccccc}
 & & & & & & F_5 & & & & & & \\
 & & & & & / & & \backslash & & & & & \\
 & & & F_4 & & & & & & F_3 & & & \\
 & & / & & \backslash & & & & / & & \backslash & & \\
 & F_3 & & & & F_2 & & F_2 & & & & F_1 & \\
 / & & \backslash & & / & & \backslash & / & & \backslash & & & \\
F_2 & & F_1 & & F_1 & & F_0 & F_1 & & F_0 & & &
\end{array}$$

Algoritmo de programação dinâmica:

```
FIBO (n)
f[0] ← 0
f[1] ← 1
para i ← 2 até n
    faça f[i] ← f[i − 1] + f[i − 2]
devolva f[n]
```

Note a tabela $f[0\,..\,n{-}1]$

Consumo de tempo: $\Theta(n)$

Soma de subconjunto (subset-sum)

Dados inteiros não-negativos $w_1, \ldots, w_n$, W, encontrar subconjunto K de $\{1, \ldots, n\}$ tal que $\sum_{k \in K} w_k = W$.

Exemplo

$w = 100, 30, 90, 35, 40, 30, 10$ e $W = 160$

Motivação: Algum subconjunto dos cheques $w_1, \ldots, w_n$ (w_i é o valor do cheque i) tem soma W?

Propriedade da subestrutura ótima:

Suponha que K é solução do problema (n, W).
Se $K \ni n$
então $K - \{n\}$ é solução do problema $(n-1, W - w_n)$
senão K é solução do problema $(n-1, W)$.

Simplificação:
diga 1 se problema tem solução e 0 em caso contrário.

Solução recursiva:

```
Rec (w, n, W)
se W = 0
    então devolva 1
se n = 0
    então devolva 0
se Rec (w, n − 1, W) = 1
    então devolva 1
se w_n > W
    então devolva 0
    senão devolva Rec (w, n−1, W−w_n)
```

Consumo de tempo: $\Omega(2^n)$

Por que demora tanto?
O mesmo subproblema é resolvido muitas vezes.

Programação dinâmica

$$s[i,Y] := \begin{array}{ll} 1 & \text{se problema } (i,Y) \text{ tem solução} \\ 0 & \text{caso contrário} \end{array}$$

Recorrência:

$s[i,0] = 1$

$s[0,Y] = 0$ se $Y > 0$

$s[i,Y] = s[i-1,Y]$ se $w_i > Y$

$s[i,Y] = \max\{s[i-1,Y], s[i-1,Y-w_i]\}$

i \ Y	0	1	2	3	4
0	0	0	0	0	0
1					
2					
3	⋆	⋆	⋆	⋆	
4				?	
5					

$W = 4$
$n = 5$

Lembrete: W e $w_1, \ldots, w_n$ são inteiros

Cada subproblrema (i, Y) resolvido uma só vez:

SOMA-DE-SUBCONJ (w, n, W)

para $i \leftarrow 0$ até n faça
    $s[i, 0] \leftarrow 1$
para $Y \leftarrow 1$ até W faça
    $s[0, Y] \leftarrow 0$
    para $i \leftarrow 1$ até n faça
        $s[i, Y] \leftarrow s[i-1, Y]$
        se $s[i, Y] = 0$ e $w_i \leq Y$
            então $s[i, Y] \leftarrow s[i-1, Y - w_i]$
devolva $s[n, W]$

Consumo de tempo: $\Theta(nW)$

NOTA: O consumo $\Theta(n2^{\lg W})$ é exponencial!
Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w_1, \ldots, w_n$ e W por 1000)

EXERCÍCIO: Escreva uma versão que devolva K tal que $\sum_{k \in K} w_k = W$

Programacão dinâmica, EXEMPLO 3:

Multiplicação iterada de matrizes

Preliminares:
Se A é $p \times q$ e B é $q \times r$ então AB é $p \times r$

$(AB)[i,j] = \sum_k A[i,k]\, B[k,j]$

Número de multiplicações escalares $= p \cdot q \cdot r$

MULT-MAT (p, A, q, B, r)

para $i \leftarrow 1$ até p faça
    para $j \leftarrow 1$ até r faça
        $AB[i,j] \leftarrow 0$
        para $k \leftarrow 1$ até q faça
            $AB[i,j] \leftarrow AB[i,j] + A[i,k] \cdot B[k,j]$

Multiplicação iterada: $A_1 \cdot A_2 \cdot A_3$

$$^{10}\; A_1 \;^{100}\; A_2 \;^{5}\; A_3 \;^{50}$$

$((A_1\, A_2)\, A_3)$	7500	mults escalares
$(A_1\, (A_2\, A_3))$	75000	mults escalares

Problema: Encontrar número mínimo de multiplicações escalares necessário para calcular produto $A_1A_2 \cdots A_n$.

$$
\begin{array}{ccccccccccc}
p_0 & & p_1 & & p_2 & \cdots & p_{n-1} & & p_n \\
& A_1 & & A_2 & & \ldots & & A_n &
\end{array}
$$

cada A_i é $p_{i-1} \times p_i$

Soluções ótimas contêm soluções ótimas: se

$$(A_1A_2)\,(A_3((A_4A_5)A_6))$$

é ordem ótima de multiplicação então

$$(A_1A_2) \quad \text{e} \quad (A_3((A_4A_5)A_6))$$

também são ordens ótimas.

$m[i,j] =$ número mínimo de multiplicações escalares para calcular $A_i \cdots A_j$

Decomposição: $(A_i \cdots A_k)\,(A_{k+1} \cdots A_j)$

Recorrência:

se $i = j$ então $m[i,j] = 0$

se $i < j$ então $m[i,j] =$

$$\min_{i \le k < j} \{\, m[i,k] + p_{i-1} p_k p_j + m[k{+}1,j] \,\}$$

Exemplo:

$$m[3,7] = \min_{3 \le k < 7} \{\, m[3,k] + p_2 p_k p_7 + m[k{+}1,7] \,\}$$

Algoritmo recursivo:
recebe $p_{i-1}, \ldots, p_j$ e devolve $m[i,j]$

```
REC-MAT-CHAIN (p, i, j)
se i = j
    então devolva 0
m[i, j] ← ∞
para k ← i até j − 1 faça
    q1 ← REC-MAT-CHAIN (p, i, k)
    q2 ← REC-MAT-CHAIN (p, k + 1, j)
    q ← q1 + p_{i−1} p_k p_j + q2
    se q < m[i, j]
        então m[i, j] ← q
devolva m[i, j]
```

Consumo de tempo: $\Omega(2^n)$
onde $n = j - i + 1$.

Por que demora tanto?
O mesmo subproblema é resolvido muitas vezes.

Cálculo do consumo de tempo de REC-MAT-CHAIN:

$T(n) =$ número comparações entre q e $m[*,*]$
na linha 6 quando $n := j - i + 1$

$$
\begin{aligned}
T(1) &= 0 \\
T(n) &= \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \\
&= 2\sum_{k=2}^{n-1} T(h) \; + \; (n-1) \\
&= 2(T(2) + \cdots + T(n-1)) \; + \; (n-1)
\end{aligned}
$$

para $n \geq 2$

Fácil verificar: $T(n) \geq 2^{n-2}$ para $n \geq 2$

Programação dinâmica:
cada subproblema $A_i \cdots Aj$ resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2,6]$ preciso de
$m[2,2]$, $m[2,3]$, $m[2,4]$, $m[2,5]$ e de
$m[3,6]$, $m[4,6]$, $m[5,6]$, $m[6,6]$.

$i \backslash j$	1	2	3	4	5	6	7	8
1	0							
2		0	$\star$	$\star$	$\star$	?		
3			0			$\star$		
4				0		$\star$		
5					0	$\star$		
6						0		
7							0	
8								0

Calcule todos os $m[i,j]$ com $j - i + 1 = 2$,
depois todos com $j - i + 1 = 3$,
depois todos com $j - i + 1 = 4$,
etc.

Algoritmo de programação dinâmica:
recebe $p_0, p_1, \ldots, p_n$ e devolve $m[1, n]$

MATRIX-CHAIN-ORDER (p, n)

02 para $i \leftarrow 1$ até n faça

03 $\quad m[i,i] \leftarrow 0$

04 para $l \leftarrow 2$ até n faça

05 $\quad$ para $i \leftarrow 1$ até $n - l + 1$ faça

06 $\quad\quad j \leftarrow i + l - 1$

07 $\quad\quad m[i,j] \leftarrow \infty$

08 $\quad\quad$ para $k \leftarrow i$ até $j - 1$ faça

09 $\quad\quad\quad q \leftarrow m[i,k] + p_{i-1} p_k p_j + m[k+1, j]$

10 $\quad\quad\quad$ se $q < m[i,j]$

11 $\quad\quad\quad\quad$ então $m[i,j] \leftarrow q$

12 devolva $m[1,n]$

Linhas 4–11: tratam das subcadeias $A_i \cdots A_j$
de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$

Curioso verificar que consumo é $\Omega(n^3)$:

Número de execuções da linha 9
(para cada i, linha 9 é executada $l-1$ vezes)

l	i	execs linha 9
2	$1,\ldots,n-1$	$(n-1)\cdot 1$
3	$1,\ldots,n-2$	$(n-2)\cdot 2$
4	$1,.,n-3$	$(n-3)\cdot 3$
$n-1$	$1,2$	$2\cdot(n-2)$
n	1	$1\cdot(n-1)$
	total	$\sum_{h=1}^{n-1} h(n-h)$

Para $n \geq 6$, $\sum_{h=1}^{n-1} h(n-h) =$

$$
\begin{aligned}
&= n\sum_{h=1}^{n-1} h - \sum_{h=1}^{n-1} h^2 \\
&= n\tfrac{1}{2}n(n-1) - \tfrac{1}{6}(n-1)n(2n-1) \qquad \text{(CLRS p.1060)} \\
&\geq \tfrac{1}{2}n^2(n-1) - \tfrac{1}{6}2n^3 \\
&\geq \tfrac{1}{2}n^2\tfrac{5n}{6} - \tfrac{1}{3}n^3 \\
&= \tfrac{5}{12}n^3 - \tfrac{1}{3}n^3 \\
&= \tfrac{1}{12}n^3
\end{aligned}
$$

Consumo de tempo é $\Omega(n^3)$

TAREFA 15

Exercício 15.A [CLRS 15.2-1]
Encontre a maneira ótima de fazer a multiplicação iterada das matrizes cujas dimensões são $(5, 10, 3, 12, 5, 50, 6)$.

Exercício 15.B [CLRS 15.2-5]
Mostre que são necessários exatamente $n - 1$ pares de parênteses para especificar exatamente a ordem de multiplicação de $A_1 \cdot A_2 \cdots A_n$.

Exercício 15.C [CLRS 15.3-2]
Desenhe a árvore de recursão para o algoritmo MERGE-SORT aplicado a um vetor de 16 elementos. Por que a técnica de programação dinâmica não é capaz de acelerar o algoritmo?

Exercício 15.D [CLRS 15.3-5 expandido]
Considere o seguinte algoritmo para determinar a ordem de multiplicação de uma cadeia de matrizes $A_1, A_2, \ldots, A_n$ de dimensões $p_0, p_1, \ldots, p_n$: primeiro, escolha k que minimize p_k; depois, determine recursivamente as ordens de multiplicação de $A_1, \ldots, A_k$ e $A_{k+1}, \ldots, A_n$. Esse algoritmo produz uma ordem que minimiza o número total de multiplicações escalares? E se k for escolhido de modo a maximizar p_k? E se k for escolhido de modo a minimizar p_k?

Exercício 15.E
Prove que o número de execuções da linha 09 em MATRIX-CHAIN-ORDER é O(n^3).

Exercício 15.F [Subset-sum. CLRS 16.2-2 simplificado]
Escreva um algoritmo de programação dinâmica para o seguinte problema: dados números inteiros não-negativos $w_1, \ldots, w_n$ e W, encontrar um subconjunto K de $\{1, \ldots, n\}$ que satisfaça $\sum_{k \in K} w_k \leq W$ e maximize $\sum_{k \in K} w_k$. (Imagine que $w_1, \ldots, w_n$ são os tamanhos de arquivos digitais que você deseja armazenar em um disquete de capacidade W.)

Exercício 15.G [Mochila 0-1. CLRS 16.2-2]
O problema da mochila 0-1 consiste no seguinte: dados números inteiros não-negativos $v_1, \ldots, v_n$, $w_1, \ldots, w_n$ e W, queremos encontrar um subconjunto K de $\{1, \ldots, n\}$ que

$$\text{satisfaça } \sum_{k \in K} w_k \leq W \text{ e maximize } \sum_{k \in K} v_k .$$

(Imagine que w_i é o *peso* e v_i é o *valor* do objeto i.) Resolva o problema usando programação dinâmica.

Exercício 15.H [Partição equilibrada]
Seja S o conjunto das raízes raízes quadradas dos números $1, 2, \ldots, 500$. Escreva e teste um programa que determine uma partição (A, B) de S tal que a soma dos números em A seja tão próxima quanto possível da soma dos números em B. Seu algoritmo resolve o problema? ou só dá uma solução "aproximada"?

Uma vez calculados A e B, seu programa deve imprimir a diferença entre a soma de A e a soma de B e depois imprimir a lista dos quadrados dos números em um dos conjuntos.