Técnicas e Análise de Algoritmos: programação dinâmica

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados II (AE23CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

Sumário

- Programação Dinâmica
  - Exemplo 1: multiplicação de cadeias de matrizes
  - Exemplo 2: problema da mochila binária

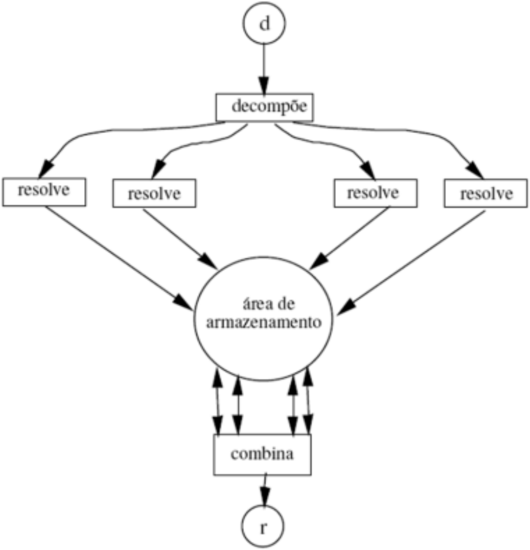
Programação Dinâmica

- Quando um algoritmo recursivo tem complexidade exponencial, a programação dinâmica pode levar a um algoritmo mais eficiente
- A técnica de programação dinâmica consiste em dividir o problema original em subproblemas mais simples e resolvê-los, armazenando os resultados em uma tabela
- A ideia básica da programação dinâmica é construir por etapas
- Inicialmente, a entrada é decomposta em partes mínimas, para as quais são obtidas respostas

Programação Dinâmica

- Em cada passo, sub-resultados são combinados obtendo-se respostas para partes maiores
- O processo é feito até que seja obtida uma resposta para o problema original
- A decomposição é feita uma única vez e os casos menores são tratados antes dos maiores
- Suas soluções são armazenadas para serem usadas quantas vezes forem necessárias

- Estrutura geral da programação dinâmica



Programação Dinâmica

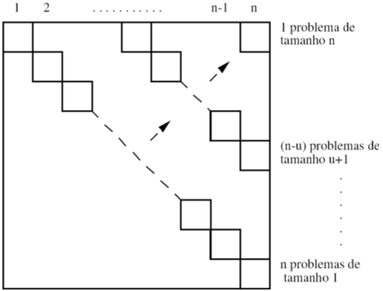
- A programação dinâmica é aplicada em diversos problemas famosos, tais como:
  - Multiplicação de cadeias de matrizes
  - Mochila binária

- Cálculo de multiplicação de cadeias de matrizes: minimizar as operações escalares
- Considere o produto:
  $M = M_1[10,20] \times M_2[20,50] \times M_3[50,1] \times M_4[1,100]$
  - $M = M_1 \times (M_2 \times (M_3 \times M_4))$ requer 125.000 operações
  - $M = (M_1 \times (M_2 \times M_3)) \times M_4$ requer 2.200 operações
- O produto de duas matrizes $p \times q$ e $q \times r$ requerem $O(pqr)$ operações

- Resolver esse problema por tentativa e erro é um processo exponencial em n ($f(n) \geq 2^{n-2}$)
- Usando programação dinâmica é possível obter um algoritmo $O(n^3)$!!

- Seja $m_{i,j}$ o menor custo para calcular o produto $M_i \times M_{i+1} ... \times M_j$ para $1 \leq i \leq j \leq n$
  - $m_{i,j} = 0$ se $i = j$
  - $m_{i,j} = Min_{i \leq k \leq j}(m_{i,k} + m_{k+1,j} + b_{i-1}b_kb_j)$ se $j > i$
  - onde:
    - $m_{i,k}$ representa o custo mínimo para calcular $M' = M_i \times M_{i+1} \times ... \times M_k$
    - $m_{k+1,j}$ representa o custo mínimo para calcular $M'' = M_{k+1} \times M_{k+2} \times ... \times M_j$
    - $b_{i-1}b_kb_j$ é o custo para multiplicar $M'[b_{i-1}, b_k]$ por $M''[b_k, b_j]$
    - $b_{i-1} \times b_i$ são as dimensões da matriz M_i
    - $m_{i,j}$, $j > i$, representa o custo mínimo de todos os valores possíveis de k entre i e $j - 1$ da soma dos três termos

- Os valores m_{ij} são calculados na ordem crescente das diferenças nos subscritos
- O cálculo inicia com m_{ij} para todo i, depois $m_{i,i+1}$ para todo i
- Assim, esse método é chamado ascendente, ao contrário dos métodos recursivos, que são chamados descendentes



- Considerando o exemplo anterior: $M = M_1[10,20] \times M_2[20,50] \times M_3[50,1] \times M_4[1,100]$. Como minimizar as operações escalares?

$m_{11} = ?$	$m_{12} = ?$	$m_{13} = ?$	$m_{14} = ?$
	$m_{22} = ?$	$m_{23} = ?$	$m_{24} = ?$
		$m_{33} = ?$	$m_{34} = ?$
			$m_{44} = ?$

- Custos para as multiplicações (em ordem por diagonal)
  - $m_{11} = m_{22} = m_{33} = m_{44} = 0$ (entre apenas uma matriz, não há multiplicação)
  - $m_{12} = M_1 \times M_2$
  - $m_{23} = M_2 \times M_3$
  - $m_{34} = M_3 \times M_4$
  - $m_{13} = (M_1 \times M_2) \times M_3$ ou $m_{13} = M_1 \times (M_2 \times M_3)$
    - $m_{13} = m_{12} \times M_3$ ou $m_{13} = M_1 \times m_{23}$
  - $m_{24} = (M_2 \times M_3) \times M_4$ ou $m_{13} = M_2 \times (M_3 \times M_4)$
    - $m_{24} = m_{23} \times M_4$ ou $m_{13} = M_2 \times m_{34}$
  - $m_{14} = m_{12} \times m_{34}$ ou $m_{14} = m_{13} \times M_4$ ou $m_{14} = M_1 \times m_{24}$

- Considerando o exemplo anterior: $M = M_1[10, 20] \times M_2[20, 50] \times M_3[50, 1] \times M_4[1, 100]$. Como minimizar as operações escalares?

$m_{11} = ?$	$m_{12} = ?$	$m_{13} = ?$	$m_{14} = ?$
	$m_{22} = ?$	$m_{23} = ?$	$m_{24} = ?$
		$m_{33} = ?$	$m_{34} = ?$
			$m_{44} = ?$

$b_0 = 10$	$b_1 = 20$	$b_2 = 50$	$b_3 = 1$	$b_4 = 100$

- Relembrando:
  - $m_{i,j} = 0$ se $i = j$
  - $m_{i,j} = Min_{i \leq k \leq j}(m_{i,k} + m_{k+1,j} + b_{i-1} b_k b_j)$ se $j > i$

- Considerando o exemplo anterior: $M = M_1[10, 20] \times M_2[20, 50] \times M_3[50, 1] \times M_4[1, 100]$. Como minimizar as operações escalares?

$m_{11} = 0$	$m_{12} = ?$	$m_{13} = ?$	$m_{14} = ?$
	$m_{22} = 0$	$m_{23} = ?$	$m_{24} = ?$
		$m_{33} = 0$	$m_{34} = ?$
			$m_{44} = 0$

$b_0 = 10$	$b_1 = 20$	$b_2 = 50$	$b_3 = 1$	$b_4 = 100$

- Relembrando:
  - $m_{i,j} = 0$ se $i = j$
  - $m_{i,j} = Min_{i \leq k \leq j}(m_{i,k} + m_{k+1,j} + b_{i-1} b_k b_j)$ se $j > i$

- Considerando o exemplo anterior: $M = M_1[10,20] \times M_2[20,50] \times M_3[50,1] \times M_4[1,100]$. Como minimizar as operações escalares?

$m_{11} = 0$	$m_{12} = 10.000$	$m_{13} = ?$	$m_{14} = ?$
	$m_{22} = 0$	$m_{23} = 1.000$	$m_{24} = ?$
		$m_{33} = 0$	$m_{34} = 5.000$
			$m_{44} = 0$

$b_0 = 10$	$b_1 = 20$	$b_2 = 50$	$b_3 = 1$	$b_4 = 100$

- $m_{12} = m_{11} + m_{22} + b_0 * b_1 * b_2 = 0 + 0 + 10 * 20 * 50 = 10.000$
- $m_{23} = m_{22} + m_{33} + b_1 * b_2 * b_3 = 0 + 0 + 20 * 50 * 1 = 1.000$
- $m_{34} = m_{33} + m_{44} + b_2 * b_3 * b_4 = 0 + 0 + 50 * 1 * 100 = 5.000$

- Considerando o exemplo anterior: $M = M_1[10, 20] \times M_2[20, 50] \times M_3[50, 1] \times M_4[1, 100]$. Como minimizar as operações escalares?

$m_{11} = 0$	$m_{12} = 10.000$	$m_{13} = 1.200$	$m_{14} = ?$
	$m_{22} = 0$	$m_{23} = 1.000$	$m_{24} = 3.000$
		$m_{33} = 0$	$m_{34} = 5.000$
			$m_{44} = 0$

$b_0 = 10$	$b_1 = 20$	$b_2 = 50$	$b_3 = 1$	$b_4 = 100$

- $m_{13} =$
  - $m_{11} + m_{23} + b_0 * b_1 * b_3 = 0 + 1.000 + 10 * 20 * 1 = \boldsymbol{1.200}$
  - $m_{12} + m_{33} + b_0 * b_2 * b_3 = 10.000 + 10 * 50 * 1 = 10.500$
- $m_{24} =$
  - $m_{22} + m_{34} + b_1 * b_2 * b_4 = 0 + 5.000 + 20 * 50 * 100 = 105.000$
  - $m_{23} + m_{44} + b_1 * b_3 * b_4 = 1.000 + 0 + 20 * 1 * 100 = \boldsymbol{3.000}$

- Considerando o exemplo anterior: $M = M_1[10,20] \times M_2[20,50] \times M_3[50,1] \times M_4[1,100]$. Como minimizar as operações escalares?

$m_{11} = 0$	$m_{12} = 10.000$	$m_{13} = 1.200$	$m_{14} = \mathbf{2.200}$
	$m_{22} = 0$	$m_{23} = 1.000$	$m_{24} = 3.000$
		$m_{33} = 0$	$m_{34} = 5.000$
			$m_{44} = 0$

$b_0 = 10$	$b_1 = 20$	$b_2 = 50$	$b_3 = 1$	$b_4 = 100$

  - $m_{14} =$
    - $m_{11} + m_{24} + b_0 * b_1 * b_4 = 0 + 3.000 + 10 * 20 * 100 = 23.000$
    - $m_{12} + m_{34} + b_0 * b_2 * b_4 = 10.000 + 5.000 + 10 * 50 * 100 = 65.000$
    - $m_{13} + m_{44} + b_0 * b_3 * b_4 = 1.200 + 0 + 10 * 1 * 100 = \boldsymbol{2.200}$

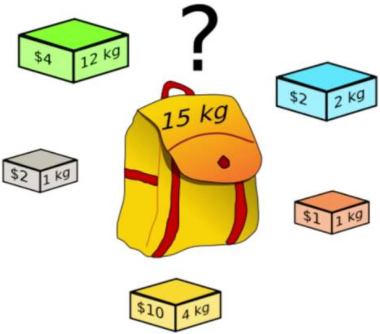
- Melhor custo: valor canto superior da matriz

```
int multiplicacao_minima(int** m, int n, int b[]){
   int h, i, j, k, aux;
   for (i = 0; i < n; i++)
      m[i][i] = 0;
   for (h = 1; h < n; h++) { // indica qual diagonal será processada
      for (i = 1; i <= n - h; i++){ // n - h é a quantidade de elementos na diagonal
         j = i + h; // i => linhas; j => colunas
         m[i - 1][j - 1] = INT_MAX;
         for (k = i; k <= j - 1; k++){
            aux = m[i - 1][k - 1] + m[k][j - 1] + b[i - 1] * b[k] * b[j];
            if (aux < m[i - 1][j - 1])
               m[i - 1][j - 1] = aux;
         }
      }
   }
   return m[0][n - 1];

}
```

- Complexidade: $O(n^3)$

- Lembra do problema da mochila binária?
  - Considere n itens a serem levados para uma viagem, dentro de uma mochila de capacidade b
  - Cada item x_j tem um peso a_j e um valor c_j. Quais itens escolher, que modo que o valor total dos itens levados seja o maior possível?



- O problema da mochila é definido em programação matemática:

$$P : z = \max \sum_{j=1}^{n} c_j x_j$$

- sujeito a

$$\sum_{j=1}^{n} a_j x_j \leq b, \; x_j \in \{0, 1\}$$

- **Princípio para aplicação de programação dinâmica**: imagine que o lado direito da desigualdade assume um valor λ que varia de 0 até b e as soluções ótimas são $x_0, ..., x_k, ..., x_b$

- Isto nos leva a definir o problema $P_k(\lambda)$, a sua respectiva solução ótima $x_k(\lambda)$ e o seu respectivo valor objetivo $f_k(\lambda)$

$$P_k(\lambda) : f_k(\lambda) = \max \sum_{j=1}^{k} c_j x_j$$

- sujeito a

$$\sum_{j=1}^{k} a_j x_j \leq \lambda,\ x_j \in \{0,1\},\ \{a\}_{j=1}^{n},\ j = 1, \ldots, k$$

- $P_k(\lambda)$ é o problema da mochila restrito aos k primeiros itens e uma mochila de capacidade λ
- Resta-nos obter um procedimento que permita calcular $f_k(\lambda)$ em termos dos valores $f_s(u)$ com $s \leq k$ e $u \leq \lambda$
- O que podemos dizer sobre a solução ótima $x*$ para o problema $P_k(\lambda)$ com valor $f_k(\lambda)$?
    - $x_k^* = 0$ ou $x_k^* = 1$

- Considerando cada caso, temos:
  1. Se $x_k^* = 0$, então a solução ótima satisfaz $f_k(\lambda) = f_{k-1}(\lambda)$
  2. Se $x_k^* = 1$, então a solução ótima satisfaz $f_k(\lambda) = c_k + f_{k-1}(\lambda - a_k)$
- Combinando os casos (1) e (2), obtêm-se:

$$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\} \tag{1}$$

- Definindo-se os valores iniciais como $f_0(\lambda) = 0$ para $\lambda \geq 0$, pode-se utilizar a Equação 1 para calcular sucessivamente os valores de $f_1, f_2, \ldots, f_n$ para todos os valores inteiros de $\lambda \in \{0, \ldots, b\}$

λ	f_0	f_1	f_2	f_3	f_4
0	0				
1	0				
2	0				
3	0				
4	0				
5	0				
6	0				
7	0				

$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\}$

$a_1 = 2,\ a_2 = 1,\ a_3 = 6,\ a_4 = 5,\ c_1 = 10,\ c_2 = 7,\ c_3 = 25,\ c_4 = 24,\ b = 7$

λ	f_0	f_1	f_2	f_3	f_4
0	0	0			
1	0	0			
2	0	10			
3	0	10			
4	0	10			
5	0	10			
6	0	10			
7	0	10			

$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\}$

$a_1 = 2$, $a_2 = 1$, $a_3 = 6$, $a_4 = 5$, $c_1 = 10$, $c_2 = 7$, $c_3 = 25$, $c_4 = 24$, $b = 7$

f1(0) = max{f0(0), c1 + f0(0 - a1)} = max{0, $\boxed{\text{10 + f0(0 - 2)}}$} = 0

- Note que, para peso negativo (0 - 2), o resultado da operação é zero, já não podemos extrapolar o peso da mochila de capacidade λ

f1(1) = max{f0(1), c1 + f0(1 - a1)} = max{0, 10 + f0(1 - 2)}= 0

f1(2) = max{f0(2), c1 + f0(2 - a1)} = max{0, 10 + f0(2 - 2)} = 10

f1(3) = max{f0(3), c1 + f0(3 - a1)} = max{0, 10 + f0(3 - 2)} = 10

f1(4) = max{f0(4), c1 + f0(4 - a1)} = max{0, 10 + f0(4 - 2)} = 10

f1(5) = max{f0(5), c1 + f0(5 - a1)} = max{0, 10 + f0(5 - 2)} = 10

f1(6) = max{f0(6), c1 + f0(6 - a1)} = max{0, 10 + f0(6 - 2)} = 10

f1(7) = max{f0(7), c1 + f0(7 - a1)} = max{0, 10 + f0(7 - 2)} = 10

λ	f_0	f_1	f_2	f_3	f_4
0	0	0	0		
1	0	0	7		
2	0	10	10		
3	0	10	17		
4	0	10	17		
5	0	10	17		
6	0	10	17		
7	0	10	17		

$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\}$

$a_1 = 2,\ a_2 = 1,\ a_3 = 6,\ a_4 = 5,\ c_1 = 10,\ c_2 = 7,\ c_3 = 25,\ c_4 = 24,\ b = 7$

f2(0) = max{f1(0), c2 + f1(0 - a2)} = max{0, 7 + f1(0 - 1)} = 0
f2(1) = max{f1(1), c2 + f1(1 - a2)} = max{0, 7 + f1(1 - 1)} = 7
f2(2) = max{f1(2), c2 + f1(2 - a2)} = max{10, 7 + f1(2 - 1)} = 10
f2(3) = max{f1(3), c2 + f1(3 - a2)} = max{10, 7 + f1(3 - 1)} = 17
f2(4) = max{f1(4), c2 + f1(4 - a2)} = max{10, 7 + f1(4 - 1)} = 17
f2(5) = max{f1(5), c2 + f1(5 - a2)} = max{10, 7 + f1(5 - 1)} = 17
f2(6) = max{f1(6), c2 + f1(6 - a2)} = max{10, 7 + f1(6 - 1)} = 17
f2(7) = max{f1(7), c2 + f1(7 - a2)} = max{10, 7 + f1(7 - 1)} = 17

λ	f_0	f_1	f_2	f_3	f_4
0	0	0	0	0	
1	0	0	7	7	
2	0	10	10	10	
3	0	10	17	17	
4	0	10	17	17	
5	0	10	17	17	
6	0	10	17	25	
7	0	10	17	32	

$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\}$

$a_1 = 2,\ a_2 = 1,\ a_3 = 6,\ a_4 = 5,\ c_1 = 10,\ c_2 = 7,\ c_3 = 25,\ c_4 = 24,\ b = 7$

f3(0) = max{f2(0), c3 + f2(0 - a3)} = max{0, 25 + f2(0 - 6)} = 0
f3(1) = max{f2(1), c3 + f2(1 - a3)} = max{7, 25 + f2(1 - 6)} = 7
f3(2) = max{f2(2), c3 + f2(2 - a3)} = max{10, 25 + f2(2 - 6)} = 10
f3(3) = max{f2(3), c3 + f2(3 - a3)} = max{17, 25 + f2(3 - 6)} = 17
f3(4) = max{f2(4), c3 + f2(4 - a3)} = max{17, 25 + f2(4 - 6)} = 17
f3(5) = max{f2(5), c3 + f2(5 - a3)} = max{17, 25 + f2(5 - 6)} = 17
f3(6) = max{f2(6), c3 + f2(6 - a3)} = max{17, 25 + f2(6 - 6)} = 25
f3(7) = max{f2(7), c3 + f2(7 - a3)} = max{17, 25 + f2(7 - 6)} = 32

λ	f_0	f_1	f_2	f_3	f_4
0	0	0	0	0	0
1	0	0	7	7	7
2	0	10	10	10	10
3	0	10	17	17	17
4	0	10	17	17	17
5	0	10	17	17	24
6	0	10	17	25	31
7	0	10	17	32	34

$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\}$

$a_1 = 2,\ a_2 = 1,\ a_3 = 6,\ a_4 = 5,\ c_1 = 10,\ c_2 = 7,\ c_3 = 25,\ c_4 = 24,\ b = 7$

f4(0) = max{f3(0), c4 + f3(0 - a4)} = max{0, 24 + f2(0 - 5)} = 0
f4(1) = max{f3(1), c4 + f3(1 - a4)} = max{7, 24 + f2(1 - 5)} = 7
f4(2) = max{f3(2), c4 + f3(2 - a4)} = max{10, 24 + f2(2 - 5)} = 10
f4(3) = max{f3(3), c4 + f3(3 - a4)} = max{17, 24 + f2(3 - 5)} = 17
f4(4) = max{f3(4), c4 + f3(4 - a4)} = max{17, 25 + f3(4 - 5)} = 17
f4(5) = max{f3(5), c4 + f3(5 - a4)} = max{17, 24 + f3(5 - 5)} = 24
f4(6) = max{f3(6), c4 + f3(6 - a4)} = max{25, 24 + f3(6 - 5)} = 31
f4(7) = max{f3(7), c4 + f3(7- a4)} = max{32, 24 + f3(7 - 5)} = 34

- Conforme apresentado no exemplo, o custo total colocado na mochila é 34

Programação Dinâmica

- Outros exemplos de aplicação:
  - Algoritmo de Dijkstra
  - Subsequência crescente máxima
  - Sequência de Fibonacci
  - ...

Programação Dinâmica

- Vantagens:
  - Explora todas as alternativas de maneira eficiente
  - A cada iteração, a decisão pode ser reconsiderada
  - Tem prova de correção simples
- Desvantagens:
  - A solução pode ser lenta
  - Necessita de maior espaço de memória

Referências I

- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Clifford, S.
  Algoritmos: teoria e prática.
  Elsevier, 2012.

- Horowitz, E., Sahni, S. Rajasekaran, S.
  Computer Algorithms.
  Computer Science Press, 1998.

- Szwarcfiter, J.; Markenzon, L.
  Estruturas de Dados e Seus Algoritmos.
  LTC, 2010.

- Ziviani, N.
  Projeto de Algoritmos - com implementações em Java e C++.
  Thomson, 2007.