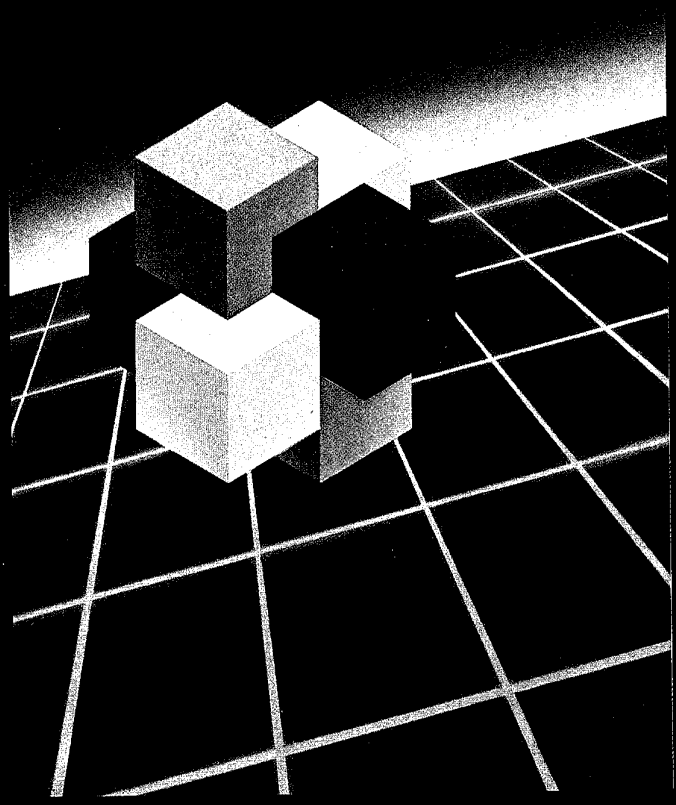
PANEL '85
EXPODATA

20 a 27 de Julho de 1985
UFRGS - Porto Alegre - Brasil

V Congresso da Sociedade Brasileira de Computação
XI Conferência Latino-Americana de Informática



ANAIS vol. I

* L.V. TOSCANI
** P.A.S. VELOSO

SUMARIO

A programação dinâmica é uma técnica de desenvolvimento de algoritmos muito útil para uma certa classe de problemas. Este artigo apresenta uma formalização da programação dinâmica através de tipos abstratos de dados e um estudo da complexidade intrínseca dessa técnica.

ABSTRACT

Dynamic programming is a technique for algorithm development which is very useful for a certain class of problems. This paper presents a formalization of dynamic programming by means of abstract data types, as well as a study of the intrinsic complexity of this technique.

* Mestre em Informática (PUC/RJ, 73); desenvolvimento e complexidade de algoritmos; Prof. adjunto da UFRGS; Cx. Postal 1501, 90000, Porto Alegre, RS.

** Ph.D. em Ciências da Comp.(Univ. da Califórnia - Berkeley, 75); teoria e metodologia de programação; Prof. associado na PUC/RJ; Depto. de Informática PUC/RJ; Rua Marques de São Vicente 225, 22.453, Rio de Janeiro, RJ.

1. INTRODUÇÃO

A formalização de métodos computacionais busca seu aproveitamento mais eficaz e abrangente. Além disso, permite uma melhor compreensão e uma maior clareza para os métodos, facilitando o estudo dos seus desempenhos.

Este trabalho descreve um método de desenvolvimento de algoritmos, conhecido por programação dinâmica, apresentando as definições informal e formal, com diagrama sintático, exemplo e um estudo da complexidade de tempo intrínseca do método.

2. APRESENTAÇÃO INFORMAL

A programação dinâmica é um método de desenvolvimento de algoritmos, que consiste em, dado um problema, dividí-lo em subproblemas, solucionar os subproblemas, guardar os subresultados, combinar subproblemas menores e subresultados para obter e resolver problemas maiores, até recompor e resolver o problema original. O diagrama sintático da figura 1 descreve essa situação (P = domínio de problemas, R = domínio de resultados).

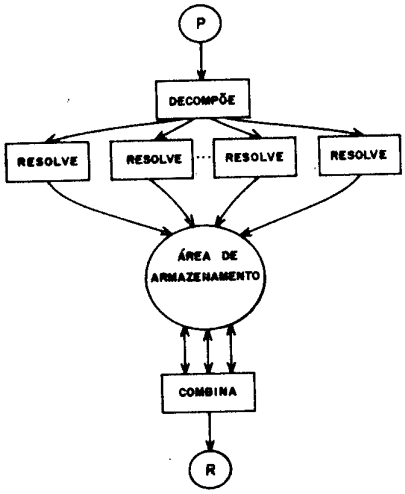


Figura 1

O problema é decomposto uma única vez, assim os subproblemas menores são gerados antes dos maiores. Por esse motivo esse método é chamado ascendente, ao contrário dos metodos recursivos que são descendentes.

3. DOMÍNIO DE APLICAÇÃO

A programação dinâmica é usada para resolver problemas cuja solução pode ser obtida através de uma seqüência ótima de decisões, conforme o diagrama sintático da figura 2.

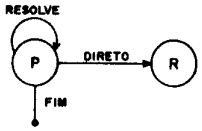


Figura 2

Decisões são tomadas sucessivamente até que uma certa condição fim (solução encontrada) seja atingida, quando é obtida a resposta final.

A seção seguinte apresenta um exemplo de aplicação desta técnica. Outros problemas que podem ser resolvidos por programação dinâmica são: árvore binária de busca ótima, problema do caixeiro viajante, etc |TER 82|.

4. EXEMPLO ILUSTRATIVO

Em |TER 82| é apresentado um exemplo simples que caracteriza bem o método. Esse exemplo será transcrito aqui.

Deseja-se multiplicar n matrizes, i.é, calcular $M=M_1 \times M_2 x \ldots x M_n$ onde cada matriz M_i tem b_{i-1} linhas e b_i colunas $1 \leqslant i \leqslant n$. O algoritmo trivial requer p x q x r operações para multiplicar uma matriz p x q por outra q x r. A multiplicação de matrizes é associativa, existem portanto várias maneira possíveis de se realizar essa multiplicação com diferentes números de operações correspondentes. O problema consiste, então, em determinar a seqüência de multiplicações que requer o número mínimo de operações.

Um algoritmo que enumere todas as seqüências possíveis, calcule os respectivos números totais de operações requeridas e em seguida escolha a seqüência ótimo, tem complexidade exponencial em n (número de matrizes), o que é inviável na prática, quando n é relativamente grande.

O algoritmo apresentado aqui, num primeiro passo decompõe o problema em n subproblemas de tamanho 1, resolve-os guarda o resultado na diagonal de uma matriz; passo a passo combina as soluções de n-u+1 problemas de tamanho u, para resolver n-u problemas de tamanho u+1 (u=1,2,...,n-1) e guarda os resultados numa diagonal superior da matriz. O processo termina com a solução de 1 problema de tamanho n.

Chame de m_{ij} o custo do produto, isto é o número mínimo de operações necessárias para calcular $M_i \times M_{i+1} \times \ldots \times M_j$. Se $M' = M_i \times \ldots \times M_k$ e $M'' = M_{k+1} \times \ldots \times M_j$ o custo mínimo de $M' \times M''$ é $m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j$, o custo mínimo de $M_i \times M_{i+1} \times \ldots \times M_j$ é m_{ij} =

$$\min_{i \leq k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j).$$

Entrada: $(b_0, b_1, \ldots, b_n)$

Saída: m_{1n}

```
1. para i ← 1 até n faça m_ii ← 0 fim-para;
2. para u ← 1 até n-1 faça
3.     para i ← 1 até n-u faça
4.         j ← i+u   (* u = j-1 *)
5.         m_ij ← min (m_ik + m_k+1,j + b_i-1 x b_k x b_j)
                  i≤k<j
6.         fim-para
7.     fim-para
8. para-com-saída (m_in)
```

A seqüência ótima das multiplicações pode ser obtida guardando os valores k obtidos na linha 5 do algoritmo.

A complexidade do algoritmo é $O(n^3)$; a programação dinâmica tornou possível resolver o problema com uma ordem de complexidade polinomial.

5. FORMALIZAÇÃO DO MÉTODO

Nesta secção a programação dinâmica é definida formalmente como um tipo de dado abstrato.

5.1 Diagrama Sintático

Domínios: P, domínio das instâncias de problemas, R, domínio de resultados, N, conjuntos dos números naturais, Q, conjunto das seqüências de elementos de P, M, conjunto das seqüências de elementos de R.

$$Q = P^{+} = \bigcup_{i \in N-\{0\}} P^{i} \qquad M = R^{+} = \bigcup_{i \in N-\{0\}} R^{i}$$

Funções: decompõe (P → Q), decompõe o problema original em subproblemas de tamanho mínimo; inicializa (Q → M), resolve os problemas de tamanho mínimo; combina (Q x M → Q) combina os problemas atuais para criar um novo conjunto de problemas de tamanho uma unidade maior; atualiza (Q x M → M), resolve os problemas atuais, usando as soluções dos problemas do nível anterior; recupera (M → R), recupera a solução do problema; tamanho (P → M), dá o tamanho do problema.

Predicados: resolve (P x R), resolve (p,r) = r é solução de p; subinst (P x P), subinst (p, q) = q é subinstância de p.

A partir dessas funções e predicados foram definidos a função tamanho^{+} e o predicado resolve^{+}, como segue: tamanho^{+}: Q → N, $\text{tamanho}^{+}(p_1, p_2, \ldots, p_i) = \max \{\text{tamanho}(p_1), \ldots, \text{tamanho}(p_i)\}$ e resolve^{+} $(PxR)^{+}$, $\text{resolve}^{+}((p_1, p_2, \ldots, p_i), (r_1, r_2, \ldots, r_i)) = \bigwedge_{j=1,\ldots,i}$ resolve (p_j, r_j), onde $(PxR)^{+} = \bigcup_{i \in N-\{0\}} P^{i} \times R^{i}$, $\text{subinst}^{+}(p,(q_1,\ldots,q_i)) = \bigwedge_{j=1,\ldots i} \text{subinst}(p,q_j)$

O relacionamento entre os domínios funções e predicados é mostrado na figura 3.

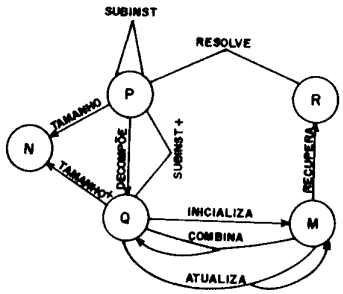


Figura 3

5.2 Axiomatização

Os seis axiomas seguintes definem a semântica do tipo

AX1. $(\forall p)$ (resolve$^+$ (decompõe (p), inicializa (decompõe (p))))

AX2. $(\forall q)$ $(\forall m)$ [resolve$^+$ (q,m) $\rightarrow$ resolve$^+$ (combina (q,m), atualiza (q,m))]

AX3. $(\forall p)$ subsinst (p, decompõe (p))

AX4. $(\forall p)$ $(\forall q)$ $(\forall m)$ [subinst$^+$(p,q) $\rightarrow$ subinst$^+$(p, combina (q,m))]

AX5. $(\forall q)$ $(\forall m)$(tamanho$^+$ (combina (q,m)) = tamanho$^+$ (q)+1)

AX6. $(\forall p)$ $(\forall q)(\forall m)$[(subinst$^+$(p,q) $\wedge$ tamanho (p) = tamanho$^+$ (q) $\wedge$
resolve$^+$ (q,m)) $\rightarrow$ resolve (p, recupera(m))]

Observações:

O axioma 1 diz que a função inicializa calcula os resultados dos subproblemas resultantes da decomposição. No axioma 2 é dito que as funções combina e atualiza mantêm a relação resolve. A função decompõe, decompõe uma instância de um problema em subinstância do problema inicial, é o que diz o axioma 3. Pelo axioma 4 a função combina mantêm a relação subsinstância. O axioma 5 define o tamanho do problema gerado pela função combina, como uma unidade maior que o tamanho do problema que lhe deu origem. E pelo axioma 6 toda subinstância q de mesmo tamanho do problema inicial p é equivalente ao problema p, no sentido de que a solução do problema q implica na solução do problema inicial p.

6. PROGRAMA ABSTRATO

Nesta secção a programação dinâmica é apresentada como um programa abstrato, utilizando as funções auxiliares já definidas e as seguintes variáveis:

p - problema entrada
r - solução do problema
n - tamanho da entrada
q - seqüência de problemas parciais
m - seqüência de soluções parciais
mínimo - tamanho dos problemas resultantes da decomposição
k - tamanho dos problemas atuais

Programa:

1. entrada p
2. n $\leftarrow$ tamanho (p)
3. q $\leftarrow$ decompõe (p)
4. mínimo $\leftarrow$ tamanho$^+$ (q)
5. m $\leftarrow$ inicializa (q)
6. para k = mínimo até n-1 faça

```
7.      q,m ← (combina, atualiza) (q,m)
8.      k ← k+1
9. fim-para
10. r ← recupera (m)
11. saída: r
```

Nas linhas 1 a 5 são inicializadas as variáveis p,n,q, mínimo e m. A variável mínimo recebe o valor do tamanho dos problemas q, resultantes da decomposição da entrada. As linhas de 6 a 9 contêm uma iteração que é executada exatamente n-mínimo vezes. A variável k guarda o tamanho dos problemas que estão sendo resolvidos. Cada iteração resolve a sequência q de problemas, guarda os resultados em m e cria uma nova seqüência q de problemas de tamanho k + 1. Na linha 10 é recuperado o resultado final.

A verificação da correção do programa pode ser encontrada em |TOS 85|.

7. APLICAÇÃO AO EXEMPLO

Usando esse formalismo o exemplo anterior toma a seguinte forma:
Tipos de dados: P, conjunto de seqüências de matrizes reais multiplicáveis; R = N; $Q=P^+$.

Variáveis: $p = < M_1,\ldots, M_n > \varepsilon P$; n, constante inteira cujo conteúdo é o número de elementos de p; m, matriz n x n.

Funções: tamanho, número de matrizes consideradas no problema; decompõe, decompõe o problema de tamanho n em n problemas de tamanho 1, isto é ao invés de considerar o produto $M_1 \times M_2 \times \ldots \times M_n$ são consideradas as matrizes $M_1, M_2, \ldots M_n$; inicializa, inicializa a diagonal da matriz m com zeros; combina considera todos n-u produtos parciais de tamanho u + 1 i. é, $M_i \times M_{i+1} \times \ldots \times M_{i+u}$ $i=1,\ldots, n-u$; atualiza, preenche a próxima diagonal superior da matriz da seguinte maneira $m_{ij} \leftarrow \min_{i \leqslant k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j)$ para

$i=1,2,\ldots, n-u$ e $j = i+u$; recupera, recupera o valor contido em m_{1n}.

Predicados: resolve (p,r), r é o número de operações necessários para multiplicar as matrizes de p; subsinst (p,q), todo elemento de q é uma subinstância de p "sem buracos", isto é se $p = < M_1, M_2, \ldots, M_n >$ um elemento de q é uma seqüência do tipo $< M_i, M_{i+1}, \ldots, M_{i+k} >$

Comparando os programas da secção 4 e secção 6 verifica-se a seguinte correspondência.

programa da secção 4

Entrada $(b_0, b_1, \ldots, b_n)$

1. para i ← 1 até n faça $m_{ij} \leftarrow 0$ fim-para
2. para u ← 1 até n-1 faça
3. para i ← 1 até n-u faça
4. j ← i+u
5. $m_{ij} \leftarrow \min_{i \leq k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j)$
6. fim-para
7. fim-para
8. para com saída (m_{1n})

programa da secção 6

1. entrada: p
2. n ← tamanho (p)
3. q ← decompõe (p)
4. mínimo ← tamanho$^+$ (q)
5. m ← inicializa (q)
6. para k= mínimo até n-1 faça
7. q, m ← (combina, atualiza) (q,m)
8. k ← k+1
9. fim-para
10. r ← recupera (m)
11. saída: r

A verificação de que o exemplo satisfaz os 6 axiomas pode ser encontrado em |TOS 85|

8. COMPLEXIDADE DE TEMPO INTRÍNSECA DO MÉTODO

A complexidade das linhas 2 a 5 é a soma das complexidades de cada linha, que depende do tamanho n do problema cnsiderado e varia com as funções (tamanho, decompõe, tamanho$^+$, inicializa) executadas. As linha 6 a 9 definem uma iteração de n-mínimo passos. O tempo de uma execução dessa iteração depende do tamanho k dos problemas considerados e da complexidade da linha 7. A complexidade da iteração é a soma das complexidade da cada execução. A complexidade da linha 7 é o número de problemas da seqüência q vezes a complexidade de solução de um problema de tamanho k.

Usando a seguinte notação: c(g) para complexidade da função g e Comp para complexidade total do algoritmo conclui-se:

$$\text{Comp} = c(\text{tamanho}) + c(\text{decompõe}) + c(\text{tamanho}^{+}) + c(\text{inicializa}) + \sum_{k=\text{mínimo}}^{n-1} c(\text{combina, atualiza}) + c(\text{recupera})$$

Fazendo algumas suposições com respeito à complexidade das funções primitivas é possível obter-se algumas regras de comportamento do método.

Suponha que as funções tamanho, decompõe, tamanho^{+} e recupera tem complexidade constante e a função inicializa tem complexidade de O (n) (Esta suposição é bem razoável!). Pode-se ser um pouco mais geral e supor que a soma das complexidades das cinco funções em questão é de O (n). Partindo desta suposição a complexidade do algoritmo fica:

$$\text{Comp} = c'n + \sum_{k=\text{mínimo}}^{n-1} c(\text{combina, atualiza})$$

Além disso, suponha que mínimo = 1 e que c(combina, atualiza) é polinomial em k. Então tem-se $\text{Comp} = c'n + \sum_{k=1}^{n-1} k^{i}$, mas esse somatório é $O(n^{i+1})$, logo a complexidade do algoritmo é polinomial e de ordem i+1 [TOS 85] (mínimo = 1 é irrelevante e não altera a complexidade total).

Por outro lado, se c(combina, atualiza) é não polinomial $\sum_{k=\text{mínimo}}^{n-1} c(\text{combina, atualiza})$ é não polinomial e a complexidade total do algoritmo é também não polinomial.

Para o exemplo apresentado tem-se: c(tamanho) é constante, c (tamanho^{+}) é constante, c (inicializa) é O(n), c (recupera) é constante, c (combina, atualiza)) é O((n-k) x k), mínimo=1. Logo,

$$\text{Comp} = c_1' n + \sum_{k=1}^{n-1} ((n-k) \times k) = O(n^3)$$

9. CONCLUSÃO

A programação dinâmica é especialmente útil no desenvolvimento de algoritmos, quando a solução do problema pode ser alcançada após uma seqüência de decisões. A seqüência ótima de decisões é obtida evitando as seqüências que sabidamente não resultam na seqüência ótima.

A abstração focaliza alguns aspectos do problema, considerados mais importantes e obscurece os detalhes menos importantes para o enfoque desejado. A abstração do método apresentado nesse trabalho evidencia o processo iterativo ascendente e o aspecto de armazenamento de resultados parciais, mas obscurece o processo de rejeição de seqüências de deci

sões que sabidamente não resultariam na seqüência ótima. Esse enfoque foi muito útil no estudo da complexidade intrínseca da programação dinâmica. Outra abstração do método que evidencia outros aspectos pode ser encontrada em |WAG 84|.

A programação dinâmica é não só uma técnica a mais de desenvolvimento de algoritmos, mas uma técnica valiosa no desenvolvimento de algoritmos eficientes. É especialmente útil no projeto de algoritmos para problemas NP-difíceis, como o problema do caixeiro viajante, por exemplo, cujo algoritmo trivial é $O(n!)$ e o algoritmo obtido através dessa técnica é $O(n^2\ 2^n)$ |TER 82|e|TOS 85|. Outros exemplos de algoritmos para problemas NP-difíceis obtidos pela programação dinâmica podem ser encontrados em |HOR 78|.

Outros métodos de desenvolvimento de algoritmos foram estudados e formalizados em |VEL 80| e |WAG 84|.

BIBLIOGRAFIA

|AHO 74| - AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN. "The Design and Analysis of Computer Algoritms". Reading, Mass., Addison-Wesley, 1974.

|GRE 81| - GREENE, D.H. & D.E. KNUTH. "Mathematics for the Analysis of Algorithms". Boston, Birkhauser, 1981.

|HOR 78| - HOROWITZ, E. & S. SAHNI. "Fundaments of Computer Algorithms". Comp. Sci. Press. 1978.

|TER 82| - TERADA, R. "Desenvolvimento de Algoritmos e Complexidade de Computação". Terceira Escola de Computação. Depto. de Informática, PUC/RJ, 1982.

|TOS 85| - TOSCANI, L.V. & VELOSO, P.A.S. Programação Dinâmica via tipos abstrato de dados. Porto Alegre, PGCC da UFRGS. (a publicar).

|VEL 80| - VELOSO, P.A.S. "Divide - and - Couquer via Data Types". Proc. III Latin-American Conf. on Informatics, Caracas, Venezuela, 1980.

|VEL 83| - VELOSO, P.A.S. "Problems and Solutions: Towards a General Formulation" - 3rd International Conference in Computer Science. Santiago, Chile, 1983.

|WAG 84| - WAGA, C.F.E. "Métodos para resolver problemas". Depto. Informática da PUC/RJ, Dissertação de mestrado. Rio, 1984.