Programação Dinâmica

prof. Fábio Luiz Usberti
prof. Cid Carvalho de Souza

MC521 - Desafios de Programação I

Instituto de Computação - UNICAMP - 2018

Sumário

Multiplicação de Matrizes

- Considere o problema de multiplicar duas matrizes $A_{m \times p}$ e $B_{p \times n}$, resultando na matriz $C_{m \times n}$.
- Um elemento c_{ij} da matriz C pode ser determinado pela seguinte equação:

$$c_{ij} = \sum_{k=1}^{p} a_{ik} \cdot b_{kj}$$

- Ou seja, c_{ij} corresponde à soma dos produtos dos elementos da i-ésima linha da matriz A com os elementos da j-ésima coluna da matriz B. Por exemplo:

$$\begin{bmatrix} 1 & 0 & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & \mathbf{2} \\ 2 & \mathbf{4} \\ 3 & \mathbf{6} \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 3 & \mathbf{6} \\ 5 & 10 \end{bmatrix}$$

- Quantas multiplicações são necessárias para obter a matriz C?

Implementação

```
#define MAX 100

int A[MAX][MAX];
int B[MAX][MAX];
int C[MAX][MAX];

/* multiplying two matrices A(m,p) and B(p,n)
to form matrix C(m,n) */
void matrix_mult(int m, int n, int p) {

  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) {
      C[i][j] = 0;
      for (int k = 0; k < p; ++k) {
        C[i][j] += A[i][k]*B[k][j];
      }
    }
  }

}
```

Multiplicação de Cadeias de Matrizes

Problema: Multiplicação de Matrizes

Calcular o número mínimo de operações de multiplicação (escalar) para computar a matriz M dada por:

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

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

- Como as matrizes são multiplicadas aos pares, o problema em questão trata-se de encontrar uma parentização ótima para a cadeia de matrizes.
- Para calcular a matriz M' dada por $M_i \times M_{i+1}$ são necessárias $b_{i-1} * b_i * b_{i+1}$ multiplicações entre os elementos de M_i e M_{i+1}.

Multiplicação de Cadeias de Matrizes (Cont.)

- **Exemplo:** Qual é o 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 pode afetar significativamente o tempo de processamento.

- Inicialmente, para todo (i, j) tal que $1 \leq i \leq j \leq n$, vamos definir as seguintes matrizes:
$$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j.$$

- Agora, 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 também, caso contrário, seria possível obter uma parentização de M ainda melhor.
- Eis a subestrututra ótima do problema: a parentização ótima de M inclui a parentização ótima de $M_{i,k}$ e $M_{k+1,n}$.

Multiplicação de Cadeias de Matrizes (Cont.)

- De forma geral, se $m[i,j]$ é número mínimo de multiplicações que deve ser efetuado para computar $M_i \times M_{i+1} \times \ldots \times M_j$, então $m[i,j]$ é dado por:

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

- Podemos então projetar um algoritmo recursivo de busca completa para resolver o problema.

MinimoMultiplicacoesRecursivo(b, i, j)

$\triangleright$ **Entrada:** Vetor b com as dimensões das matrizes e os índices i e j que delimitam o início e término da subcadeia.
$\triangleright$ **Saída:** O número mínimo de multiplicações escalares necessário para computar a multiplicação da subcadeia. Esse valor é registrado em uma tabela ($m[i, j]$), bem como o índice da divisão em subcadeias ótimas ($s[i, j]$).

1. **se** $i = j$ **então retorne**(0)
2. $m[i, j] := \infty$
3. **para** $k := i$ **até** $j - 1$ **faça**
4. $\quad q := MinimoMultiplicacoesRecursivo(b, i, k)+$
   $\quad\quad MinimoMultiplicacoesRecursivo(b, k + 1, j)+$
   $\quad\quad b[i - 1] * b[k] * b[j]$
5. $\quad$ **se** $m[i, j] > q$ **então**
6. $\quad\quad m[i, j] := q$; $s[i, j] := k$
7. **retorne**($m[i, j]$).

Efetuando a Multiplicação Ótima

- Para efetuar a multiplicação da cadeia de matrizes com o número mínimo de multiplicações escalares é possível aplicar a tabela s, que registra os índices ótimos de divisão em subcadeias.

MultiplicaMatrizes(M, s, i, j)

▷ **Entrada:** Cadeia de matrizes M, a tabela s e os índices i e j que delimitam a subcadeia a ser multiplicada.
▷ **Saída:** A matriz resultante da multiplicação da subcadeia entre i e j, efetuando o mínimo de multiplicações escalares.

1. **se** $i < j$ **então**
2. $\quad X := MultiplicaMatrizes(M, s, i, s[i, j])$
3. $\quad Y := MultiplicaMatrizes(M, s, s[i, j] + 1, j)$
4. $\quad$ **retorne**$(Multiplica(X, Y, b[i-1], b[s[i, j]], b[j]))$
5. **se não retorne**(M_i);

- O número mínimo de operações feita pelo algoritmo recursivo é dada pela recorrência:

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

- Portanto, $T(n) = 2\sum_{i=1}^{n-1} T(i) + n \geq 2T(n-1) + 1 \geq 2^n$, para $n > 1$.
- Portanto, o algoritmo recursivo tem complexidade $\Omega(2^n)$, o que é impraticável!

Sobreposição de subproblemas

- É possível notar que o algoritmo recursivo apresenta sobreposição de subproblemas: o cálculo do mesmo $m[i,j]$ pode ser requerido em vários subproblemas.
- Por exemplo, para $n = 4$, $m[1,2]$, $m[2,3]$ e $m[3,4]$ são computados duas vezes.
- O número de total de $m[i,j]$'s calculados é $O(n^2)$ apenas !
- Portanto, podemos obter um algoritmo mais eficiente se evitarmos recálculos de subproblemas.

Implementações Top-down x Bottom-up

- Existem duas implementações de programação dinâmica para evitar o recálculo de subproblemas:

1. **Top-down:** Consiste em manter a estrutura recursiva do algoritmo, registrando em uma tabela o valor ótimo para subproblemas já computados e verificando, antes de cada chamada recursiva, se o subproblema a ser resolvido já foi computado.
2. **Bottom-up:** Consiste em preencher uma tabela que registra o valor ótimo para cada subproblema de forma apropriada, isto é, a computação do valor ótimo de cada subproblema depende somente de subproblemas já previamente computados. Elimina completamente a recursão.

Multiplicação de matrizes

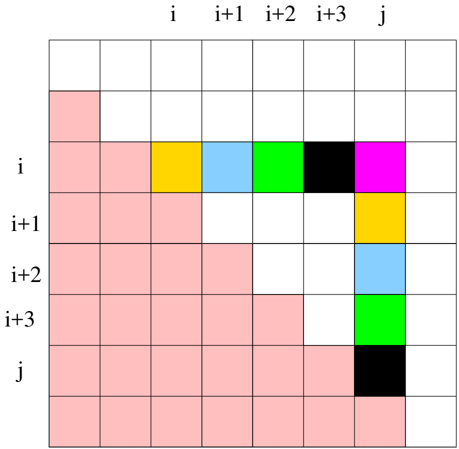
```
#define INF 1E9
#define MAX 1000
int m[MAX][MAX]; // DP table with optimum values (should be initialized with INF)
int s[MAX][MAX]; // Index table of the optimal subdivisions of the matrix chain
int b[MAX]; // vector containing matrix dimensions (should be initialized with data input)

/* Top-down DP algorithm for matrix chain multiplication.
INPUT:  i,j: indices of the matrix chain Mij = Mi * M(i+1) * ... * M(j-1) * Mj
        b: vector containing dimensions of matrix chain, such that
        b[i] and b[i+1] are the number of lines and columns for the i-th matrix.
OUTPUT: m[i][j]: minimum number of scalar multiplications for the matrix chain Mij
        s[i][j]: index of the optimal partition of the matrix chain Mij */
int matrix_chain_td(int i, int j) {
  // memoization
  if (m[i][j] < INF) return m[i][j];
  // base case
  if (i == j) return m[i][j] = 0;
  // recurrence
  for (int k = i; k < j; ++k) {
     int q = matrix_chain_td(i,k) + matrix_chain_td(k+1,j) + b[i]*b[k+1]*b[j+1];
     if (q < m[i][j]) {
        m[i][j] = q;
        s[i][j] = k;
     }
  }
  return m[i][j];
}
```

Implementação Bottom-Up

- A implementação bottom-up elimina completamente o uso de recursão.
- A implementação bottom-up para o problema da multiplicação de matrizes envolve computar, para valores crescentes de u, o valor ótimo de todas as subcadeias de tamanho u.

Implementação Bottom-Up - Exemplo



Implementação Bottom-Up - Exemplo

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

m

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

s

Implementação Bottom-Up - 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 }

Implementação Bottom-Up - 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

Implementação Bottom-Up - Exemplo

m

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

s

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

{ 200, 2, 30, 20, 5 }

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

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

Implementação Bottom-Up - Exemplo

	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

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

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

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

{ 200, 2, 30, 20, 5 }

Implementação Bottom-Up - Exemplo

	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)

Multiplicação de matrizes

```
#define INF 1E9
#define MAX 1000
int m[MAX][MAX]; // DP table with optimum values (should be initialized with INF)
int s[MAX][MAX]; // Index table of the optimal subdivisions of the matrix chain
int b[MAX]; // vector containing matrix dimensions (should be initialized with data input)

/* Bottom-up DP algorithm for matrix chain multiplication.
The algorithm starts by computing matrix chains of size one, then
computes chains of sizes u+1 (u = 1,...,n-1), in increasing order.
INPUT:  b: vector containing dimensions of matrix chain, such that
        b[i] and b[i+1] are the number of lines and columns for the i-th matrix.
OUTPUT:
        m[i][j]: minimum number of scalar multiplications for the matrix chain Mij
        s[i][j]: index of the optimal partition of the matrix chain Mij */
void matrix_chain_bu(int n) {
  // initializing base cases
  for (int i = 0; i < n; ++i) m[i][i] = 0;
  // u+1: size of the matrix subchain being computed
  for (int u = 1; u < n; ++u) {
    for (int i = 0; i < n-u; ++i) {
      int j = i + u;
      for (int k = i; k < j; ++k) {
        int q = m[i][k] + m[k+1][j] + b[i]*b[k+1]*b[j+1];
        if (q < m[i][j]) {
          m[i][j] = q;
          s[i][j] = k;
  } } } }
}
```

Implementação Bottom-Up - Complexidade

- A complexidade de tempo do algoritmo é $\Theta(n^3)$.
- A complexidade de espaço é $\Theta(n^2)$, já que é necessário armazenar a matriz com os valores ótimos dos subproblemas.

Referências

1. S. Halim e F. Halim. Competitive Programming 2, Second Edition Lulu (www.lulu.com), 2011. (IMECC – 005.1 H139c)
2. S. S. Skiena, M. A. Revilla. Programming Challenges: The Programming Contest Training Manual, Springer, 2003.
3. T.H. Cormen, C.E. Leiserson, R.L.Rivest e C. Stein. Introduction to Algorithms. 2nd Edition, McGraw-Hill, 2001. (no. chamada IMECC – 005.133 Ar64j 3.ed.)
4. U. Manber. Introduction to Algorithms: A Creative Approach. Addison-Wesley. 1989. (no. chamada IMECC – 005.133 Ec53t 2.ed.)