PROBLEMA DE MULTIPLICAÇÃO DE CADEIAS DE MATRIZES: UMA IMPLEMENTAÇÃO EM JAVA UTILIZANDO O ALGORITMO DE PROGRAMAÇÃO DINÂMICA[1]

Joel Siqueira Marques[2], Paulo Roberto Bastos de Almeida Júnior[2], Leonardo Monteiro dos Santos[2]

Abstract *– This article presents a Java implementation of the algorithms MATRIX-CHAIN-ORDER, which determines the optimal number of scalar multiplications needed to compute the product of chain matrices, and PRINT-OPTIMAL-Parens, which prints an optimal placement of the brackets for the product chain matrices. The objective of this study is show how the problem Chain Matrix Multiplication was solved using concepts of dynamic programming, significantly reducing the complexity. It will be shown that the complexity of this problem is exponential (2^n) is solved by brute force, and polynomial (n^3) is solved by dynamic programming.*

Keywords: Matrix Multiplication Chains. Dynamic Programming.

Resumo *– Este artigo realiza uma implementação em Java dos algoritmos MATRIX-CHAIN-ORDER, que determina o número ótimo de multiplicações escalares necessárias para calcular o produto de cadeia de matrizes, e PRINT-OPTIMAL-PARENS, que imprime uma colocação ótima dos parênteses para o produto de cadeias de matrizes. O objetivo do trabalho é mostrar como o Problema de Multiplicação de Cadeias de Matrizes foi solucionado utilizando conceitos de programação dinâmica, diminuindo significativamente a complexidade. Será mostrado que a complexidade desse problema é exponencial (2^n) se resolvido por força bruta, e polinomial (n^3) se resolvido por programação dinâmica.*

Palavras-chave: Multiplicação de cadeias de matrizes. Programação Dinâmica.

1. Introdução

O Problema de Multiplicação de Cadeias de Matrizes pode ser enunciado da forma a seguir:

"dada uma cadeia $<A_1, A_2, ..., A_n>$ de n matrizes na qual, para i = 1, 2, ..., n, a matriz A_i tem dimensão $p_{i-1}p_i$, coloque completamente entre parênteses o produto $A_1A_2...A_n$ de um modo que minimize o número de multiplicações escalares (CORMEN, 2002).".

[1]Trabalho desenvolvido durante a disciplina Análise e Projeto de Algoritmos, como parte da avaliação referente ao 7º semestre, orientado pelo Profº Bruno Moutinho

[2]Graduandos em Ciência da Computação, Universidade da Amazônia UNAMA
"Campus" Alcindo Cacela - Av. Alcindo Cacela, 287, Belém, PA Brasil
joelmarques2008@hotmail.com, prbajunior@yahoo.com.br, leomsp@gmail.com

Segundo CORMEN, um produto de matrizes é completamente colocado entre parênteses se ele for uma única matriz ou o produto de dois produtos de matrizes completamente colocado entre parênteses, cercado por parênteses. A multiplicação de matrizes é associativa, e assim todas as colocações entre parênteses resultam no mesmo produto. Por exemplo, se a cadeia de matrizes é $<A_1, A_2, A_3, A_4>$, o produto $A_1A_2A_3A_4$ pode ser completamente colocado entre parênteses de cinco modos distintos:

- ✓ $(A_1(A_2(A_3A_4)))$,
- ✓ $(A_1((A_2A_3)A_4))$,
- ✓ $((A_1A_2)(A_3A_4))$,
- ✓ $((A_1(A_2A_3))A_4)$,
- ✓ $(((A_1A_2)A_3)A_4)$.

Vale resaltar que, no problema de multiplicação de cadeia de matrizes, não é feita nenhuma multiplicação de matrizes, o objetivo é apenas determinar uma ordem para multiplicar cadeia de matrizes que tenha o custo mais baixo. Desse modo, a entrada num programa de computador que implemente o problema discutido nesse trabalho seria somente um vetor p com as dimensões das matrizes, e a saída seria a ordem com que as matrizes seriam multiplicadas com um número mínimo de multiplicação escalares. Esse é o produto final desse artigo.

2. Uma solução por força bruta

Antes de estudar os algoritmos de programação dinâmica, vale mostrar um exemplo prático resolvido por verificação exaustiva de todas as possíveis colocações entre parênteses. Essa solução resulta em um algoritmo não eficiente.

Se a cadeia de matrizes é $<A_1, A_2, A_3>$ cujas dimensões é dado pelo vetor p = <10,100,5,50>.

Note que, pela definição do problema as posições no vetor p são p_0, p_1, p_2, ..., p_n, onde n é o número de matrizes. Portanto, o tamanho do vetor p é tamanho(p) = n +1.

As dimensões das matrizes A_1, A_2 e A_3 são:

$A_i = p_{i-1}p_i$

- ✓ $A_1 = p_{1-1} * p_1 = p_0 * p_1 = 10 \times 100$ (linha x coluna);
- ✓ $A_2 = p_{2-1} * p_2 = p_1 * p_2 = 100 \times 5$ (linha x coluna);
- ✓ $A_3 = p_{3-1} * p_3 = p_2 * p_3 = 5 \times 50$ (linha x coluna).

Cada valor em p, a partir do segundo, participa da próxima dimensão. Isso permite dizer que só se pode multiplicar duas matrizes se elas forem compatíveis: o número de colunas (C_1) da primeira matriz (M_1) deve ser igual ao número de linhas (L_2) da segunda matriz (M_2). Esse produto M_1M_2 resulta em uma terceira matriz M com dimensões $L_1 \times C_2$.

A Equação 2.1 mostra a condição para que seja possível o produto entre duas matrizes.

$$M_1 = (L_1 \; x \; C_1) \; e \;\; M_2 = (L_2 \; x \; C_2)$$
$$M = (M_1 \; x \; M_2)$$
$$M = (L_1 \; x \; C_1) \; (L_2 \; x \; C_2)$$
$$M = (L_1 \; x \; C_2) \qquad \textit{Equação 2.1}$$

O tempo (T) para calcular M é dado pelo número de multiplicações escalares, que é dado pela Equação 2.2:

$$T = (L_1 * \; C_1 * \; C_2) \qquad \textit{Equação 2.2}$$

Na cadeia $<A_1, A_2, A_3>$, o produto $A_1A_2A_3$ pode ser completamente colocado entre parênteses de dois modos distintos:

- ✓ $((A_1A_2)A_3)$ ➔ 7500 multiplicações escalares (ver *Equação 2.2*);
- ✓ $(A_1(A_2A_3))$ ➔ 75000 multiplicações escalares (ver *Equação 2.2*).

Desse modo, produto de acordo com a 1ª colocação é 10 vezes mais rápido que a 2ª opção.

CORMEN mostra uma equação de recorrência que determina o número de possibilidades de colocações entre parênteses para uma cadeia de n matrizes. Essa equação é mostrada na Equação 2.3.

$$P(n) = \begin{cases} 1 & se \; n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & se \; n \geq 2 \end{cases}$$

Equação 2.3

Onde P(n) é o número de alternativas para colocações dos parênteses de uma sequência de n matrizes. Quando n = 1, há apenas uma matriz e portanto somente um modo de colocar totalmente entre parênteses o produto de matrizes. Quando n ≥ 2, um produto de matrizes totalmente entre parênteses é o produto de dois subprodutos de matrizes totalmente entre parênteses, e a divisão entre os dois subprodutos pode ocorrer entre a k-ésima e a (k+1)-ésima matrizes para qualquer k = 1, 2, ..., n-1.

A Tabela 2.1 usa o método de substituição na recorrência *Equação 2.3* para 8 matrizes.

Tabela 2.1 Quantidades de colocações entre parênteses conforme Equação 2.3.

Número de matrizes(n)	K=1	K=2	K=3	K=4	K=5	K=6	K=7	Quantidade de colocações entre parênteses [P(n)]
1								1
2	1							1
3	1	1						2
4	2	1	2					5
5	5	2	2	5				14
6	16	5	4	5	16			46
7	46	16	10	10	16	46		144
8	144	46	32	25	32	46	144	469

Fonte: Os próprios autores.

Conforme mostra a Tabela 2.1 o número de soluções é exponencial em n (complexidade $\Omega(2^n)$ e, consequentemente, o método de força bruta para pesquisa exaustiva é uma estratégia pobre para se determinar a colocação ótima dos parênteses de uma cadeia de matrizes.

3. Solução por programação dinâmica

A programação dinâmica, como o método de dividir e conquistar, resolve problemas combinando as soluções para subproblemas. (Nesse contexto, o termo “programação” se refere a um método tabular, não ao processo de escrita de código de computador.). Os algoritmos de dividir e conquistar particionam o problema em subproblemas independentes, resolvem os subproblemas recursivamente, e então combinam suas soluções para resolver o problema original. Em contraste, a programação dinâmica é aplicável quando quando os subproblemas não são independentes, isto é, quando os

subproblemas compartilham subsubproblemas. Nesse contexto, um algoritmo de dividir e conquistar trabalha mais que o necessário, resolvendo repetidamente os subsubproblemas comuns. Um algoritmo de programação dinâmica resolve cada subsubproblema uma vez só e então grava sua resposta em uma tabela, evitando assim o trabalho de recalcular a resposta toda vez que o subsubproblema é encontrado (CORMEN, 2002).

Segundo CORMEN, o desenvolvimento de um algoritmo de programação dinâmica pode ser desmembrado em uma sequência de quatro etapas:

1. Caracterizar a estrutura de uma solução ótima;
2. Definir recursivamente o valor de uma solução ótima;
3. Calcular o valor de uma solução ótima em um processo de baixo para cima (bottom-up);
4. Construir uma solução ótima a partir de informações calculadas.

As seções que seguem tratam das etapas da programação dinâmica para a resolução do problema de multiplicação de cadeias de matrizes.

3.1. A estrutura de uma colocação ótima dos parênteses

O primeiro passo é caracterizar a estrutura de uma solução ótima. Será adotado a notação Ai..j para a matriz que resulta da avaliação do produto Ai.Ai+1...Aj. Para obter a solução do problema proposto, devemos obter A1..n, que pode ser obtido pelo produto de A1..k.Ak+1..n, cujo custo ótimo é obtido pela soma do custo de A1..k com Ak+1..n, mais o custo do produto delas. A subcadeia A1..k deve ter parentização ótima, do contrário poderíamos substituí-la, por outra com o custo menor que o ótimo.

Logo, uma solução ótima para uma instância do problema contém soluções ótimas para as sub-instâncias do mesmo problema, o que permite o emprego da programação dinâmica.

3.2. Uma solução recursiva

Deve-se definir uma expressão recursiva para a solução ótima em função das sub-instâncias. Será utilizada uma tabela m[i,j] para $1 \leq i \leq j \leq n$,

para armazenar a solução ótima para Ai..j. O valor m[i,i] = 0, pois Ai..i = Ai, não havendo a necessidade de qualquer cálculo.

Para i < j, podemos usar a estrutura ótima delineada na etapa 1. Assim Ai..j pode ser dividido em duas partes Ai..k e Ak+1..j, onde i ≤ k < j. Então m[i,j] é igual ao menor custo para calcular Ai..k e Ak+1..j, mais o custo para multiplicar essas duas matrizes. O custo para multiplicar Ai..k.Ak+1..j exige Pi-1.Pk.Pj multiplicações escalares.

A Equação 3.2.1 mostra a definição recursiva para o custo mínimo de colocar entre parênteses o produto AiAi+1...Aj.

$$m[i,j] = \begin{cases} 0 & se\ i = j \\ min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & se\ i < j \end{cases}$$

Equação 3.2.1

Os valores de m[i,j] fornecem os custos de soluções ótimas para subproblemas, mas não informações para a construção de uma solução ótima. Para facilitar a indicação de uma parentização ótima, basta armazenar na matriz s[i,j] o valor de k usado para o valor ótimo de m[i,j], ou seja m[i,j] = m[i,k] + m[k+1,j] + Pi-1.Pk.Pj.

3.3. Como calcular os custos ótimos

Usando a programação dinâmica têm-se $\Theta(n^2)$ subproblemas (basta observar que m[i,j] armazena os valores ótimos desses subproblemas), bem mais eficiente que o algoritmo de força bruta que é exponencial em n.

Neste ponto deve-se elaborar um algoritmo para a solução do problema, fazendo os cálculos de tal forma que nenhuma solução seja requisitada antes que a mesma já tenha sido calculada. O Procedimento 3.3.1 a seguir utiliza uma tabela auxiliar m[1..n,1..n] para armazenar os custos de m[i,j] e uma tabela auxiliar s[1..n,1..n] que registra qual índice de k alcançou o custo ótimo no cálculo de m[i,j]. Na construção de uma solução ótima será usada a tabela s.

```
MATRIX-CHAIN-ORDER(p)
1   n ← comprimento[p] -1
2   for i ← 1 to n
3       do m[i,j] ← 0
```

```
4   for L ← 2 to n                    // L é o comprimento da cadeia
5       do for i ← 1 to n - L + 1
6          do j ← i + L - 1
7             m[i,j] ← ∞
8             for k ← i to j - 1
9                do q ← m[i,k] + m[k+1,j] + p_{i-1}p_k p_j
10                  if q < m[i,j]
11                     then m[i,j] ← q
12                          s[i,j] ← k
13  return m  e s
```

Procedimento 3.3.1

O tempo de execução do *Procedimento 3.3.1* é $O(n^3)$, que pode ser determinado simplesmente observando a estrutura de loop aninhados.

3.4. A construção de uma solução ótima

A construção de uma solução ótima para o problema de multiplicação de cadeias de matrizes se baseia na tabela s. Cada entrada s[i,j] registra o valor de k tal que a colocação ótima de parênteses de AiAi+1..Aj divide o produto entre Ak e Ak+1. Desse modo, sabemos que a multiplicação de matrizes final no cálculo ótimo de A1..n, é A1..s[1,n]As[1,n]+1..n. As multiplicações de matrizes anteriores podem ser calculadas recursivamente, pois s[1,s[1,n]] determina a última multiplicação de matrizes no cálculo de A1..s[1,n], e s[s[1,n]+1,n] determina a última multiplicação de matrizes no cálculo de As[1,n]+1..n. O *Procedimento 3.4.1* abaixo mostra isso.

```
PRINT-OPTIMAL-PARENS(s,i,j)
1   if i = j
2     then print "A"i
3     else print "("
4          PRINT-OPTIMAL-PARENS(s,i,s[i,j])
5          PRINT-OPTIMAL-PARENS(s,s[i,j]+1,j)
6          print ")"
```

Procedimento 3.4.1

4. Implementação do Problema de Multiplicação de Cadeias de Matrizes em Java

Nesta seção será mostrado as implementações em Java dos algoritmos MATRIX-CHAIN-ORDER e PRINT-OPTIMAL-PARENS que resolvem o problema estudado nesse artigo.

Abaixo, o *Código fonte 4.1* da classe CadeiaMatriz.java com os métodos dos algoritmos acima mencionados. O código é fiel aos algoritmos, portanto não usou-se recursos próprios da linguagem Java, como List, Set ou Vector.

Código fonte 4.1

```
package br.com.multiplicacaoCadeiaMatrizes.modelo;

public class CadeiaMatriz {
	/**matriz para armazenar o número mínimo de multiplicações escalares(custos ótimos)**/
	private Integer m[][];

	/**matriz auxiliar que registra qual índice de k alcançou o custo ótimo no cálculo de m[i,j]**/
	private Integer s[][];

	/**variável para guardar como as matrizes deverão ser multiplicadas com o número mínimo de
	 * multiplicações escalares. Essa é a resposta para o Problema de Multiplicação de Cadeias de Matrizes
	 */
	StringBuilder colocacaoOtimaDosParenteses = new StringBuilder();

	private Integer numeroMinimoDeMultiplicacoesEscalares;

	public void MATRIX_CHAIN_ORDER(Integer p[]){

		/**sendo n=número de matrizes à multiplicar, como tamanho do vetor p=n+1, então n=p-1**/
		final Integer n = p.length - 1;//constante local

		m = new Integer[n][n];

		s = new Integer[n][n];


		for(int i=0; i<n; i++){
		    m[i][i]=0;
		}

		for(int L=2; L<=n; L++){
			for(int i=1; i<=n-L+1; i++){
				int j=i+L-1;
				m[i-1][j-1]=Integer.MAX_VALUE;
				for(int k=i; k<=j-1; k++){
					Integer q = m[i-1][k-1] + m[k][j-1] + p[i-1]*p[k]*p[j];
					if(q<m[i-1][j-1]){
						m[i-1][j-1] = q;
						numeroMinimoDeMultiplicacoesEscalares = q;
						s[i-1][j-1] = k;
					}
				}
			}
		}
//		return m,s;
	}

//	public void PRINT_OPTIMAL_PARENS(Integer s[][], Integer i, Integer j){
//
//		if(i==j){
//			System.out.print("A"+i);
//		}
//		else{
//			System.out.print("(");
```

```
//                              PRINT_OPTIMAL_PARENS(s, i, s[i-1][j-1]);
//                              PRINT_OPTIMAL_PARENS(s, s[i-1][j-1]+1, j);
//                              System.out.print(")");
//                      }
//              }

        public void PRINT_OPTIMAL_PARENS(Integer s[][], Integer i, Integer j){
                if(i==j){
                    colocacaoOtimaDosParenteses.append("A"+i);
                }
                else{
                        colocacaoOtimaDosParenteses.append("(");
                        PRINT_OPTIMAL_PARENS(s, i, s[i-1][j-1]);
                        PRINT_OPTIMAL_PARENS(s, s[i-1][j-1]+1, j);
                        colocacaoOtimaDosParenteses.append(")");
                }
        }


        /**somente get para ler as duas matrizes*/

        public Integer[][] getM() {
                return m;
        }

        public Integer[][] getS() {
                return s;
        }

        public StringBuilder getColocacaoOtimaDosParenteses() {
                return colocacaoOtimaDosParenteses;
        }

        public void setColocacaoOtimaDosParenteses(
                        StringBuilder colocacaoOtimaDosParenteses) {
                this.colocacaoOtimaDosParenteses = colocacaoOtimaDosParenteses;
        }

        public Integer getNumeroMinimoDeMultiplicacoesEscalares() {
                return numeroMinimoDeMultiplicacoesEscalares;
        }

        public void setNumeroMinimoDeMultiplicacoesEscalares(
                        Integer numeroMinimoDeMultiplicacoesEscalares) {
                this.numeroMinimoDeMultiplicacoesEscalares = numeroMinimoDeMultiplicacoesEscalares;
        }

}
```

Para construir uma interface com o usuário final, o software foi desenvolvido na tecnologia web. A *Tabela 4.1* lista todas as tecnologias usadas para deixar um ambiente bem harmonizado para o usuário que usará o software aqui produzido.

Tabela 4.1. Tecnologias usadas no software

Tecnologia	Versão
Java	Java EE 6
Linux	10.4 - 32 bits
Apache Maven	2.0.10
Eclipse Indigo	3.7
JSF	2.0.2

Facelet	1.1.15
Ajax	
CSS	
Primefaces	3.0.M3
Apache Tomcat	7.0.8

Fonte: Os próprios autores.

A *Figura 4.1* mostra a interface do sistema com o usuário. Apresenta como entrada o campo “Ordens das matrizes (entre vírgulas)” no qual será informado as dimensões das matrizes como visto nesse artigo: $p<p_0,p_1,p_2,...,p_n>$, e terá como saída a resposta ao Problema de Multiplicação de Cadeias de Matrizes, que é o objetivo desse trabalho, no campo “Colocação Ótima dos Parênteses”.

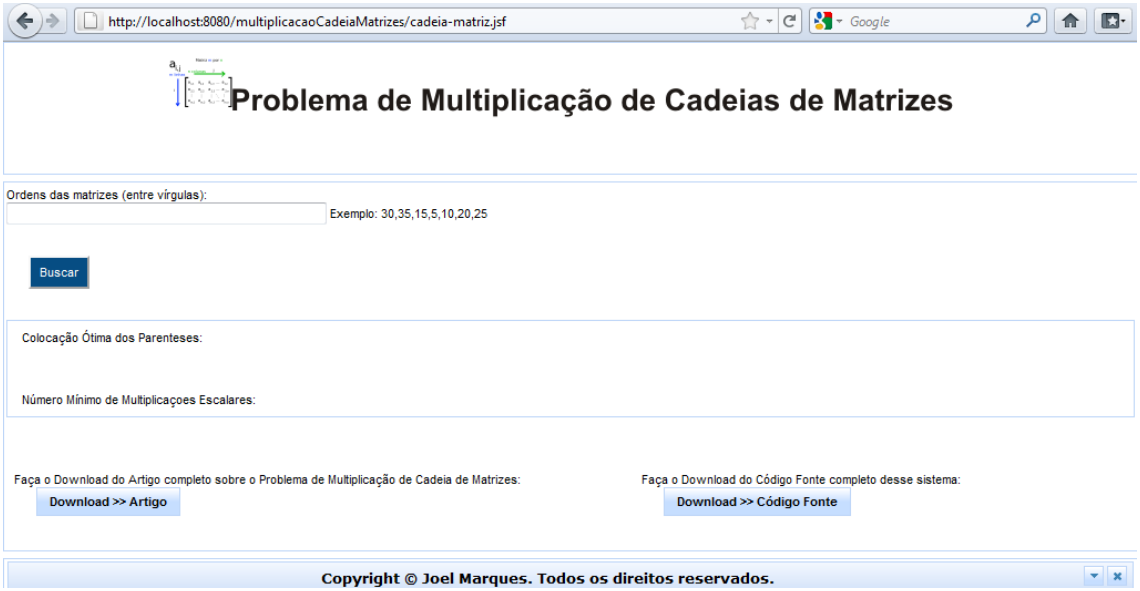

***Figura 4.1* Interface do sistema**

O exemplo de entrada mostrado na *Figura 4.1* é o mesmo exemplo estudado por CORMEN.

O *Código fonte 4.2* mostra o código fonte em JSF da interface.

Código fonte 4.2

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:p="http://primefaces.prime.com.tr/ui">

<ui:composition template="/facelet-principal.xhtml">
```

```
        <ui:define name="titulo">Multiplicação de Cadeia de Matrizes</ui:define>

        <ui:define name="conteudo">

  <h:form>
        <h:panelGroup>
                <h:outputLabel for="ordensDasMatrizes" value="Ordens das matrizes (entre vírgulas): " /><br />
                <h:inputText id="ordensDasMatrizes" value="#{controladorCadeiaMatriz.ordensDasMatrizes}"
label="Ordens das matrizes (entre vírgulas)"  size="30" style="width:283px;" />
                <h:outputLabel value="Exemplo: 30,35,15,5,10,20,25" /><br />
        </h:panelGroup>
        <br /><br />

        <h:panelGroup>
                <h:commandButton id="buscar" styleClass="botao" reRender="resultadoBusca" value="Buscar"
actionListener="#{controladorCadeiaMatriz.buscarColocacaoOtima}"/>
        </h:panelGroup>
        <br /><br /><br />

        <p:panel id="resultadoBusca">
                <h:panelGroup>
                    <h:outputLabel for="colocacaoOtimaDosParenteses" value="Colocação Ótima dos Parenteses: " /><br />
                        <h:outputText id="colocacaoOtimaDosParenteses"
value="#{controladorCadeiaMatriz.cadeiaMatriz.colocacaoOtimaDosParenteses}" label="Colocação Ótima dos Parenteses"
                                                                size="30" style="width:283px;" />
                </h:panelGroup>
                <br /><br /><br />
                <h:panelGroup>
                        <h:outputLabel for="numeroMinimoDeMultiplicacoesEscalares" value="Número Mínimo de
Multiplicaçoes Escalares: " /><br />
                        <h:outputText id="numeroMinimoDeMultiplicacoesEscalares"
value="#{controladorCadeiaMatriz.cadeiaMatriz.numeroMinimoDeMultiplicacoesEscalares}" label="Número Mínimo de
Multiplicaçoes Escalares" size="30" style="width:283px;" />
                </h:panelGroup>
        </p:panel>
        <br /><br /><br />

        <h:panelGrid columns="2" cellpadding="5">
                <h:panelGroup>
                        <h:outputLabel for="botaoDownloadArtigo" value="Faça o Download do Artigo completo sobre o
Problema de Multiplicação de Cadeia de Matrizes: " />
                        <p:commandButton id="botaoDownloadArtigo" value="Download >> Artigo" styleClass="botao"
ajax="false">
                        <p:fileDownload value="#{controladorCadeiaMatriz.fileArtigo}" />
                        </p:commandButton>
                </h:panelGroup>

                <h:panelGroup>
                        <h:outputLabel for="botaoDownloadCodigoFonte" value="Faça o Download do Código Fonte
completo desse sistema: " />
                        <p:commandButton id="botaoDownloadCodigoFonte" value="Download >> Código Fonte"
styleClass="botao" ajax="false">
                        <p:fileDownload value="#{controladorCadeiaMatriz.fileCodigoFonte}" />
                        </p:commandButton>
                </h:panelGroup>
        </h:panelGrid>

</h:form>

        </ui:define>
</ui:composition>

</html>
```

Para gerenciar a tela foi implementado o controlador ControladorCadeiaMatriz.java, mostrado no *Código fonte 4.3*.

Código fonte 4.3

```
package br.com.multiplicacaoCadeiaMatrizes.controlador;

import java.io.InputStream;
```

```
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import javax.servlet.ServletContext;
import org.primefaces.model.DefaultStreamedContent;
import org.primefaces.model.StreamedContent;
import br.com.multiplicacaoCadeiaMatrizes.modelo.CadeiaMatriz;


@ManagedBean(name = "controladorCadeiaMatriz")
@RequestScoped
public class ControladorCadeiaMatriz {

	CadeiaMatriz cadeiaMatriz = new CadeiaMatriz();

	private String ordensDasMatrizes;

	private StreamedContent fileArtigo;

	private StreamedContent fileCodigoFonte;

	private List<Integer> listaMatrizM = new ArrayList<Integer>();

	private List<Integer> listaMatrizS = new ArrayList<Integer>();

	public ControladorCadeiaMatriz(){
		InputStream streamArtigo =
((ServletContext)FacesContext.getCurrentInstance().getExternalContext().getContext()).getResourceAsStream("/arquivos/Problem
aDeMultiplicacaoDeCadeiasDeMatrizes.pdf");
		fileArtigo = new
DefaultStreamedContent(streamArtigo,"file/pdf","ProblemaDeMultiplicacaoDeCadeiasDeMatrizes.pdf");

		InputStream streamCodigoFonte =
((ServletContext)FacesContext.getCurrentInstance().getExternalContext().getContext()).getResourceAsStream("/arquivos/Problem
aDeMultiplicacaoDeCadeiasDeMatrizes.zip");
		fileCodigoFonte = new
DefaultStreamedContent(streamCodigoFonte,"file/zip","ProblemaDeMultiplicacaoDeCadeiasDeMatrizes.zip");

	}


	public void buscarColocacaoOtima(ActionEvent ae){
		Integer p[] = colocarOrdensDasMatrizesNoVetor();
		cadeiaMatriz.MATRIX_CHAIN_ORDER(p);

		Integer n = p.length-1;
		cadeiaMatriz.PRINT_OPTIMAL_PARENS(cadeiaMatriz.getS(), 1, n);

		preencherListaMatrizM();
		preencherListaMatrizS();
	}

	private Integer[] colocarOrdensDasMatrizesNoVetor(){
		String m[];
		m=ordensDasMatrizes.split(",");
		Integer p[] = new Integer[m.length];

		for(int i=0; i<m.length; i++){
			p[i] = Integer.parseInt(m[i]);
		}

		return p;
	}

	public void preencherListaMatrizM(){
		Integer m[][] = cadeiaMatriz.getM();
		for(int i=0; i<m.length; i++){
			for(int j=0; j<m.length; j++){
				listaMatrizM.add(m[i][j]);
			}
		}
	}
```

```
	public void preencherListaMatrizS(){
		Integer s[][] = cadeiaMatriz.getS();
		for(int i=0; i<s.length; i++){
			for(int j=0; j<s.length; j++){
				listaMatrizS.add(s[i][j]);
			}
		}
	}


	/**get e set*/

	public CadeiaMatriz getCadeiaMatriz() {
		return cadeiaMatriz;
	}

	public void setCadeiaMatriz(CadeiaMatriz cadeiaMatriz) {
		this.cadeiaMatriz = cadeiaMatriz;
	}

	public String getOrdensDasMatrizes() {
		return ordensDasMatrizes;
	}

	public void setOrdensDasMatrizes(String ordensDasMatrizes) {
		this.ordensDasMatrizes = ordensDasMatrizes;
	}

	public StreamedContent getFileArtigo() {
		return fileArtigo;
	}

	public StreamedContent getFileCodigoFonte() {
		return fileCodigoFonte;
	}

	public List<Integer> getListaMatrizM() {
		return listaMatrizM;
	}

	public void setListaMatrizM(List<Integer> listaMatrizM) {
		this.listaMatrizM = listaMatrizM;
	}

	public List<Integer> getListaMatrizS() {
		return listaMatrizS;
	}

	public void setListaMatrizS(List<Integer> listaMatrizS) {
		this.listaMatrizS = listaMatrizS;
	}

}
```

5. Como executar o software

Para executar o sistema é necessário um servidor web ou mesmo um contêiner web como o tomcat, este pode ser baixado no site oficial < http://tomcat.apache.org/>. Após perfeitamente configurado, realiza-se o deploy do arquivo multiplicacaoCadeiaMatrizes.war que acompanha o código fonte do resultado desse trabalho. Depois, basta acessar pelo navegador digitanto o mesmo nome do arquivo .war.

6. Conclusão

O presente artigo alcançou seu objetivo, que foi a implementação em Java dos algoritmos MATRIX-CHAIN-ORDER e PRINT-OPTIMAL-PARENS que determinam uma solução ótima para o Problema de Multiplicação de Cadeias de Matrizes utilizando programação dinâmica. Através dessa técnica foi possível identificar como o problema foi otimizado de forma bastante significativa, de um custo exponencial (2^n) para um custo polinomial (n^3).

REFERÊNCIAS

CORMEN, T., LEISERSON, C., RIVEST, R., STEIN, C. Algoritmos (Teoria e Prática) - Tradução da 2ª edição [americana]. Vandenberg D. de Souza. Rio de Janeiro: Elsevier, 2002 – 4ª Reimpressão.