**Implementação em Java**

**package** Classes;

***/\*\****

***\****

***\****

***\*/***

**public** **class** Matriz {

*/\*Atributos da classe\*/*

*/\*string: atributo que recebe os dados de saída de printOptmalParents()*

*\* para poder exibir o resultado da parentização. \*/*

**private** **static** String string;

*/\*m: atributo que recebe os valores das multiplicações feitas para o melhor custo\*/*

**private** **int** m[][];

*/\*s: atributo que recebe o valor das posições de melhor multiplicação\*/*

**private** **int** s[][];

*/\*linhas: recebe o valor total das linhas das matrizes.\*/*

**private** **int** linhas;

*/\*colunas: recebe ovalor total das colunas das matrizes\*/*

**private** **int** colunas;

*/\*inicoMatriz: atributo tipo flag, para marcar a inicialização de matrizes*

*\* no método recursiveMatrixChain(int p[], int i, int j)\*/*

**private** **boolean** inicioMatriz;

*/\*Métodos geters e setters para entrada e saída de dados nos atributos\*/*

**public** **int**[][] getM() {

**return** m;

}

**public** **void** setM(**int**[][] m) {

**this**.m = m;

}

**public** **int**[][] getS() {

**return** s;

}

**public** **void** setS(**int**[][] s) {

**this**.s = s;

}

**public** **int** getLinhas() {

**return** linhas;

}

**public** **void** setLinhas(**int** linhas) {

**this**.linhas = linhas;

}

**public** **int** getColunas() {

**return** colunas;

}

**public** **void** setColunas(**int** colunas) {

**this**.colunas = colunas;

}

**public** Matriz() {

string = "";

}

*/\*Métodos da Classe\*/*

*/\* Método para calcular o melhor custo.*

*\*Parametros: p é um vetor com as posições das matrizes.*

*\*Retorno: int[][].\*/*

**public** **static** **int**[][] MatrixChainOrder(**int** p[]) {

**int** retorno[][] = **new** **int**[p.length - 1][p.length - 1];

**try** {

**int** i = 0; *//linhas*

**int** j = 0; *//colunas*

**int** k = 0;

**int** q = 0;

**int** infinito = Integer.MAX\_VALUE; *// tipo infinito positivo (para simular o infinito)*

**int** n = p.length - 1;

**int** m[][] = **new** **int**[n][n]; *// ixj*

**int** s[][] = **new** **int**[n][n];

**for** (i = 0; i < n; i++) {

m[i][i] = 0;

}

**for** (**int** l = 1; l < n; l++) {

**for** (i = 0; i < n - l; i++) {

j = i + l;

m[i][j] = infinito;

**for** (k = i; k < j; k++) {

q = m[i][k] + m[k + 1][j] + p[i] \* p[k + 1] \* p[j + 1];

**if** (q < m[i][j]) {

m[i][j] = q;

s[i][j] = k + 1;

}

}

}

}

retorno = s;

} **catch** (Exception e) {

System.out.println("Erro: " + e);

e.printStackTrace();

}

**return** retorno;

}

*/\*Método para alocar os parênteses de uma forma ótima para a multiplicação.*

*\*Parametros: s é a matriz que contém as posições de melhor multiplicação;*

*\* i é o índice inical;*

*\* j é o índice final.*

*\*Retorno: String.\*/*

**public** String printOptmalParents(**int** s[][], **int** i, **int** j) {

**if** (i == j) {

**this**.string += "A" + (i + 1) + " ";

} **else** {

**this**.string += " ( ";

printOptmalParents(s, i, s[i][j] - 1);

printOptmalParents(s, s[i][j], j);

**this**.string += " ) ";

}

**return** **this**.string;

}

*/\*Método para inicializar os atributos da classe que serão utilizados em métodos.*

*\*Parâmetros: p é um vetor com as posições das matrizes.*

*\*Retorno: não há.\*/*

**public** **void** inicializaRecursiveMatrixChain(**int** p[]) {

**int** n = p.length - 1;

**this**.m = **new** **int**[n][n]; *// ixj*

**this**.s = **new** **int**[n][n];

**this**.inicioMatriz = **true**;

}

*/\*Método para calcular o melhor custo; porém com chamadas recursivas.*

*\*Parâmetros: p é um vetor com as posições das matrizes;*

*\* i é o índice inicial;*

*\* j é o índice final.*

*\*Retonro: int.\*/*

**public** **int** recursiveMatrixChain(**int** p[], **int** i, **int** j) {

**int** retorno = 0;

**if** (**this**.inicioMatriz) {

**if** (i == j) {

retorno = 0;

} **else** {

**this**.m[i][j] = Integer.MAX\_VALUE;

**for** (**int** k = i; k <= j - 1; k++) {

**int** q = recursiveMatrixChain(p, i, k) + recursiveMatrixChain(p, k + 1, j) + p[i] \* p[k + 1] \* p[j + 1];

**if** (q < **this**.m[i][j]) {

**this**.m[i][j] = q;

**this**.s[i][j] = k + 1;

}

}

retorno = **this**.m[i][j];

}

}

**return** retorno;

}

}