Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC3001- Algoritmos y Estructuras de Datos

Tarea N°2
Multiplicación de matrices

Alumno: José Pacheco Aguilera
Profesor: Patricio Poblete
Auxiliares: Gabriel Flores
Sven REissengger
Ayudantes: Gabriel Chandia
Fabian Mosso
Matias Risco
Matias Ramirez
Fecha de entrega: 30 de Abril de 2018

1. Introducción

En el presente informe se muestra un algoritmo que permite obtener la parentizacion que optimiza la multiplicación de n matrices con distintas dimensiones, es decir, dada una cantidad de matrices que se quieren multiplicar el algoritmo entrega como se deben multiplicar para que el espacio de memoria y tiempo utilizado sea el menor, para esto mediante un algoritmo se calcula donde debe ir el primer paréntesis, luego dado esa división, se calcula lo mismo para los dos subproblemas creados, esto quiere decir, para las matrices que se multiplican a la izquierda del primer paréntesis y las matrices que están a la derecha del primer paréntesis, y así recursivamente hasta que se hayan resuelto todos los subproblemas.

2. Diseño de la solución

La solución de este problema consistió en programación dinámica basada en el apunte, para la cual se creo la clase *parentizacion* que recibe un string el cual contiene números separados por espacios que indican las dimensiones de las matrices que se quieren multiplica. Luego se transforma ese string en una lista con el mismo orden de los números ingresados en el string. Esa lista se ingresa a una función auxiliar llamada *(*multmatrices) que calcula el orden de la multiplicación de las matrices de la lista, de todas las formas posibles de multiplicarlas, para esto se inicia colocando un paréntesis después de la segunda matriz y se calcula el orden total de las multiplicaciones, luego se coloca el paréntesis después de la tercera matriz y asi hasta colocar el paréntesis en todas las posiciones y se guarda la posición donde el orden es menor, luego de saber donde esta el primer paréntesis se hace el mismo procedimiento con las dos listas de matrices resultantes y así hasta que se colocan paréntesis en toda la lista de matrices y se guarda el valor, este valor se guarda en una matriz s que guarda el 'camino' para el cual se logra el orden menor de multiplicaciones. Luego se tiene otra función auxiliar llamada *parentizar* que recibe el matriz 's' guardada anteriormente y lee ese 'camino', lo que permite que en la pantalla se imprima en orden la parentizacion óptima.

3. Implementación

La primera función auxiliar utilizada (*multmatrices*) recibe una lista con los números de las dimensiones de las matrices que se quieren multiplicar y entrega una matriz con el 'camino' para la parentizacion óptima, para ello se crean dos matrices de $[n+1] \times [n+1]$ (con n el largo de la lista) y en una se guarda el orden de todas las posibles multiplicaciones y en otra la posición donde se debe colocar el paréntesis que optimiza el orden de la multiplicación.

```
public static int[][] multmatrices( int[] listdim) {
    int n=listdim.length-1;
    int[][] m= new int [n+1][n+1];
    int[][] s= new int [n+1][n+1];
    for (int i=1; 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][j]=Integer.MAX_VALUE;
            for (int k=i; k<=j-1;k++){
                int q=m[i][k] + m[k+1][j]+listdim[i-1]*listdim[k]*listdim[j];
                if (q<m[i][j]){
                    m[i][j]=q;
                    s[i][j]=k;
                }
            }
        }
    }
    return s;
    }
```

Figura 1: Codigo de la funcion auxiliar *multmatrices*

La segunda función auxiliar utilizada ((parentizar)) recibe una matriz y dos enteros, la matriz es la matriz resultante de *multmatrices* y los dos enteros que recibe son un 1 y el largo de la lista con dimensiones, con estos valores se hace una recursion en la cual el primer y el segundo valor que recibe la función van cambiando hasta que en algún momento estos son iguales y se imprime un ”.” que indica una matriz, y mientras no se llega a que son iguales se imprimen paréntesis que permite llegar a la parentizacion óptima.

```
public static void parentizar(int[][] s, int i, int j){
    if (i<j){
        System.out.print("(");
        parentizar(s,i,s[i][j]);
        parentizar(s,(s[i][j])+1,j);
        System.out.print(")");
        }
    else {
        System.out.print(".");
    }
}
```

Figura 2: Codigo de la funcion auxiliar *parentizar*

Por ultimo esta el *main* que es el que ejecuta todo el programa, ya que al iniciarlo, pregunta al usuario las dimensiones de las matrices que las debe escribir con un espacio entremedio, esos valores los recibe como un string que luego transforma en una lista de los mismos números sin espacio entre ellos, con esa lista se llama a la función *mulmatrices* que da como resultado el 'camino' que lleva a la parentizacion óptima de esas matrices, y con ese valor se llama a la función *parentizar* que imprime la parentizacion óptima, luego que lo imprime este proceso se puede repetir hasta que el usuario presione enter sin escribir nada.

```
public class Parentizacion {
    public static void main(String[] args) {
        String dim;
        Scanner d = new Scanner(System.in);
        System.out.println("Dimensiones de las matrices?: ");
        while (d.hasNextLine()){
            dim=d.nextLine();
            String[] Dim=dim.split(" ");
            int[] listdim= new int[Dim.length];
            if ("".equals(dim)){
                System.out.print("EOF");
                break;

            }
            else{
            for (int i=0; i<Dim.length ; i++ ){
                listdim[i]=Integer.parseInt(Dim[i]);
            }
            int[][] c=multmatrices(listdim);
            System.out.println("La parentizacion optima es: ");
            parentizar(c,1,(listdim.length)-1);
            System.out.println("");
    }
    }
    }
```

Figura 3: Codigo principal de la clase

4. Resultados

A continuación se muestran los resultados de ingresar ′2 3 4 5′ , ′2 3 4 5 1 2′ y ′32 74 124 1004 7′

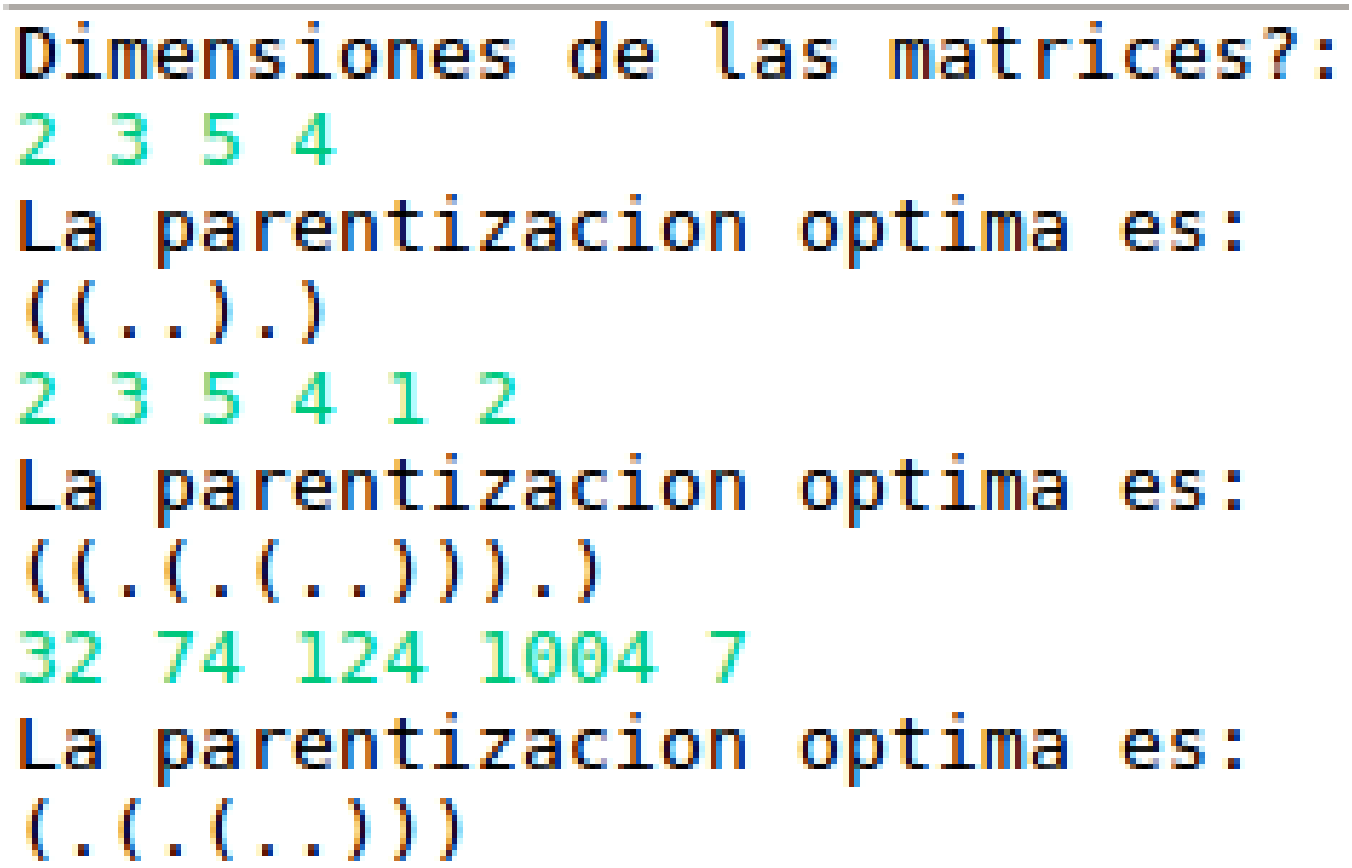

Figura 4: Resultados de la parentizacion optima

5. Conclusiones

Observando los resultados previamente expuestos se puede decir que el algoritmo utilizado para la parentizacion óptima cumple con su objetivo ya que los resultado son exactamente igual a los esperados.

6. Anexo

```
    import java.util.Scanner;

public class Parentizacion {
public static void main(String[] args) {
String dim;
Scanner d = new Scanner(System.in);
System.out.println("Dimensiones de las matrices?: ");
while (d.hasNextLine()){
     dim=d.nextLine();
         String[] Dim=dim.split(" ");
    int[] listdim= new int[Dim.length];
    if ("".equals(dim)){
         System.out.print("EOF");
         break;

   }
    else{
    for (int i=0; i<Dim.length ; i++ ){
         listdim[i]=Integer.parseInt(Dim[i]);
    }
    int[][] c=multmatrices(listdim);
        System.out.println("La parentizacion optima es: ");
         parentizar(c,1,(listdim.length)-1);
         System.out.println("");
}
}
}

public static void parentizar(int[][] s, int i, int j){
    if (i<j){
         System.out.print("(");
         parentizar(s,i,s[i][j]);
         parentizar(s,(s[i][j])+1,j);
            System.out.print(")");
        }
    else {
         System.out.print(".");
    }
}

public static int[][] multmatrices( int[] listdim) {
```

```
    int n=listdim.length-1;
    int[][] m= new int [n+1][n+1];
    int[][] s= new int [n+1][n+1];
    for (int i=1; 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][j]=Integer.MAX_VALUE;
             for (int k=i; k<=j-1;k++){
         int q=m[i][k] + m[k+1][j]+listdim[i-1]*listdim[k]*listdim[j];
         if (q<m[i][j]){
             m[i][j]=q;
             s[i][j]=k;
}
}
}
}
return s;
}

}
```