Autores: *Francisco Javier Marqués Gaona, Germán Ruano García y David Subires Esparra*

Ejercicios Tema 2

Ejercicio 1. Multiplicación rápida de matrices.

- Presentar el esquema algorítmico detallado (pseudocódigo).
- Seleccionar las estructuras de datos adecuadas para almacenar los datos.
- Implementar el correspondiente código en Java.
- Evaluar experimentalmente la eficiencia utilizando diversos juegos de prueba generados aleatoriamente (dado el tamaño del problema).
- Almacenar juegos de prueba, resultados y tiempos de ejecución.
- Analizar la eficiencia obtenida empíricamente frente a la teórica.
- Proponer si es de interés un método híbrido y sus umbrales.

Pseudocódigo algoritmo de Fuerza Bruta

```
public function fuerzaBruta(m1, m2) returns una matriz cuadrada
//Comentario: Siendo m1 y m2 matrices cuadradas
INICIO

                    VARIABLES
                      int[][] m <-- new int[m1.length][m1.length]

                      For(i = 1; i < m.lenght; i++) do
                           For(j = 0; j < m.length; j++) do
                                  int  sum <-- 0
                                  For(k = 0; k <m.length; k++) do
                                        sum <-- sum + m1[i][k] * m2[k][j]
                                  Fin for
                                  m[i][j] <-- sum
                           Fin for
                    Fin for
        Return m
FIN
```

Pseudocódigo algoritmo de DyV

```
public function divideYVenceras(m1, m2) returns una matriz cuadrada
//Comentario: Siendo m1 y m2 matrices cuadradas
INICIO

        VARIABLES
        int[][] m <-- divideYVenceras(m1, m2, m1.length)
        Return m
FIN

////////////////////////////////MÉTODO PRIVADO////////////////////////////////

private function divideYVenceras(a, b, tama) returns una matriz cuadrada
//Comentario: Siendo a y b matrices cuadradas y tama un entero
INICIO

    if(tama == 2) do
        int[][] m <-- new int[tama][tama]
        m[0][0] <-- a[0][0]*b[0][0] + a[0][1]*b[1][0]
        m[0][1] <-- a[0][0]*b[0][1] + a[0][1]*b[1][1]
        m[1][0] <-- a[1][0]*b[0][0] + a[1][1]*b[1][0]
        m[1][1] <-- a[1][0]*b[0][1] + a[1][1]*b[1][1]
        return m
    int tamaNuevo <-- tama/2
    int[][] a11 <-- new int[tamaNuevo][tamaNuevo]
    int[][] b11 <-- new int[tamaNuevo][tamaNuevo]
    int[][] a12 <-- new int[tamaNuevo][tamaNuevo]
    int[][] b12 <-- new int[tamaNuevo][tamaNuevo]
    int[][] a21 <-- new int[tamaNuevo][tamaNuevo]
    int[][] b21 <-- new int[tamaNuevo][tamaNuevo]
    int[][] a11 <-- new int[tamaNuevo][tamaNuevo]
    int[][] a22 <-- new int[tamaNuevo][tamaNuevo]
    int[][] b22 <-- new int[tamaNuevo][tamaNuevo]
    for(i = 1; i < tamaNuevo; i++) do
        for(j = 0; j < tamaNuevo; j++) do
            a11[i][j] <-- a[i][j];
            b11[i][j] <-- b[i][j];
            a12[i][j] <-- a[i][j+tamaNuevo]
            b12[i][j] <-- b[i][j+tamaNuevo]
            a21[i][j] <-- a[i+tamaNuevo][j]
            b21[i][j] <-- b[i+tamaNuevo][j]
            a22[i][j] <-- a[i+tamaNuevo][j+tamaNuevo]
            b22[i][j] <-- b[i+tamaNuevo][j+tamaNuevo]
        fin for
    fin for
```

```
        int [][]izq <-- divideYVenceras (a11, b11, tamaNuevo)
        int [][]der <-- divideYVenceras (a12, b21, tamaNuevo)
        int [][]c11 <-- sumar (izq, der)
        izq <-- divideYVenceras (a11, b12, tamaNuevo)
        der <-- divideYVenceras (a12, b22, tamaNuevo)
        int [][]c12 <-- sumar (izq, der)
        izq <-- divideYVenceras (a21, b11, tamaNuevo)
        der <-- divideYVenceras (a22, b21, tamaNuevo)
        int [][]c21 <-- sumar (izq, der)
        izq <-- divideYVenceras (a21, b12, tamaNuevo)
        der <-- divideYVenceras (a22, b22, tamaNuevo)
        int [][]c22 <-- sumar (izq, der)
        int [][]m <-- new int [tama][tama]
        for (int i=0; i<tamaNuevo; i++) do
                for (int j=0; j<tamaNuevo; j++) do
                    m[i][j] <-- c11[i][j]
                    m[i][j+tamaNuevo] <-- c12[i][j]
                    m[i+tamaNuevo][j] <-- c21[i][j]
                    m[i+tamaNuevo][j+tamaNuevo] <-- c22[i][j]
                 fin for
         fin for
         return m
FIN
```

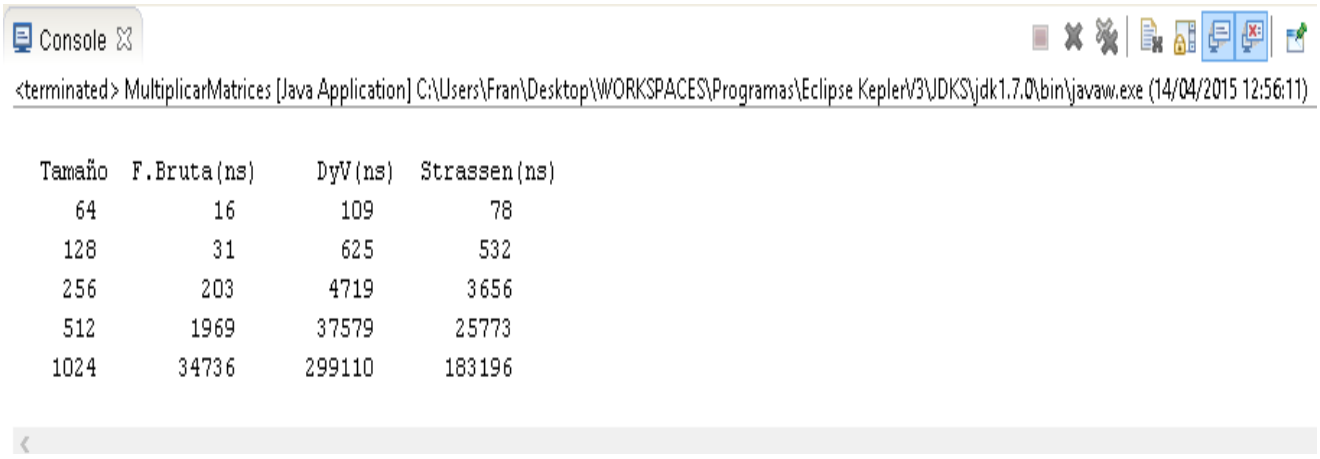
Pseudocódigo algoritmo de Strassen

```
public function strassen(m1, m2) returns una matriz cuadrada
//Comentario: Siendo m1 y m2 matrices cuadradas
INICIO

        VARIABLES
        int[][] m <-- strassen(m1, m2, m1.length)
        Return m
FIN

////////////////////////////////MÉTODO PRIVADO////////////////////////////////

private functions strassen(a, b, tama) returns una matriz cuadrada
//Comentario: Siendo a y b matrices cuadradas y tama un entero
INICIO

    if(tama == 2) do
        int [][]m <-- new int [tama][tama];
        m[0][0] <-- a[0][0]*b[0][0] + a[0][1]*b[1][0]
        m[0][1] <-- a[0][0]*b[0][1] + a[0][1]*b[1][1]
        m[1][0] <-- a[1][0]*b[0][0] + a[1][1]*b[1][0]
        m[1][1] <-- a[1][0]*b[0][1] + a[1][1]*b[1][1]
        return m
    tamaNuevo <-- tama/2
    int [][] a11 <-- new int [tamaNuevo][tamaNuevo]
    int [][] b11 <-- new int [tamaNuevo][tamaNuevo]
    int [][] a12 <-- new int [tamaNuevo][tamaNuevo]
    int [][] b12 <-- new int [tamaNuevo][tamaNuevo]
    int [][] a21 <-- new int [tamaNuevo][tamaNuevo]
    int [][] b21 <-- new int [tamaNuevo][tamaNuevo]
    int [][] a22 <-- new int [tamaNuevo][tamaNuevo]
    int [][] b22 <-- new int [tamaNuevo][tamaNuevo]
    for(i = 0; i < tamaNuevo; i++) do
        for(j = 0; j < tamaNuevo; j++) do
            a11[i][j] <-- a[i][j]
            b11[i][j] <-- b[i][j]
            a12[i][j] <-- a[i][j+tamaNuevo]
            b12[i][j] <-- b[i][j+tamaNuevo]
            a21[i][j] <-- a[i+tamaNuevo][j]
            b21[i][j] <-- b[i+tamaNuevo][j]
            a22[i][j] <-- a[i+tamaNuevo][j+tamaNuevo]
            b22[i][j] <-- b[i+tamaNuevo][j+tamaNuevo]
        fin for
    fin for
    int [][]m1 <-- strassen (restar(a12, a22), sumar(b21,b22), tamaNuevo)
    int [][]m2 <-- strassen (sumar(a11,a22), sumar(b11,b22), tamaNuevo)
```

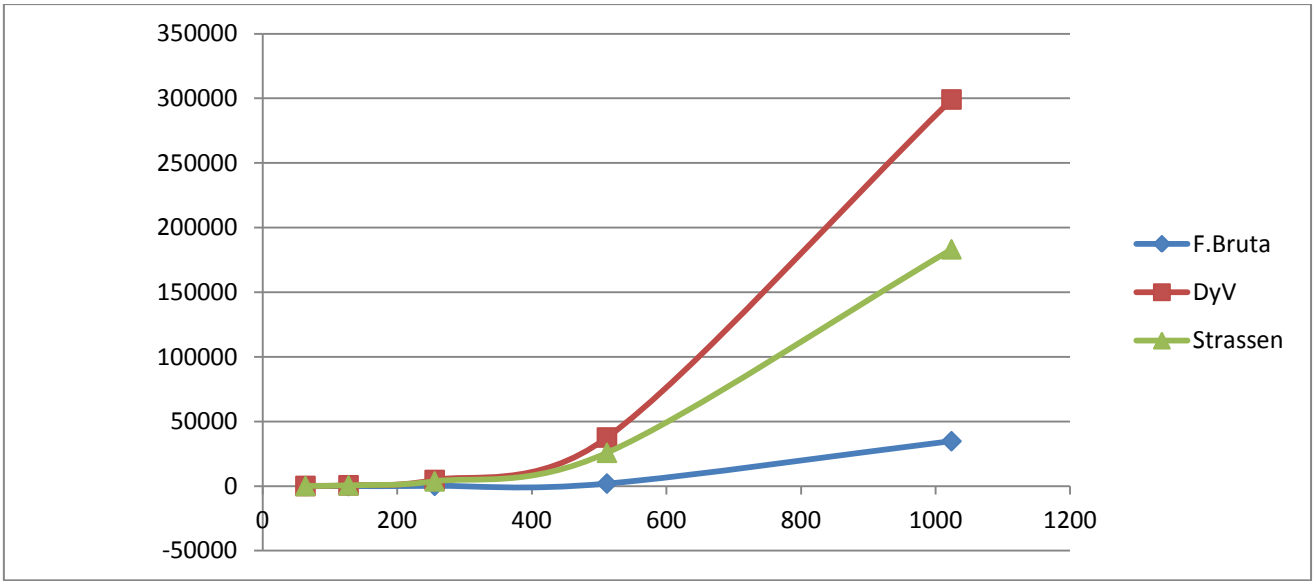
```
    int [][]m3 <-- strassen (restar(a11, a21), sumar (b11, b21), tamaNuevo)
    int [][]m4 <-- strassen (sumar(a11, a12), b22, tamaNuevo)
    int [][]m5 <-- strassen (a11, restar(b12, b22), tamaNuevo)
    int [][]m6 <-- strassen (a22, restar(b21, b11), tamaNuevo)
    int [][]m7 <-- strassen (sumar(a21, a22), b11, tamaNuevo)
    int [][]c11 <-- sumar(restar(sumar (m1, m2), m4), m6)
    int [][]c12 <-- sumar (m4, m5)
    int [][]c21 <-- sumar (m6, m7)
    int [][]c22 <-- restar (sumar (restar(m2, m3), m5), m7)
    int [][]m <-- new int [tama][tama]
    for(i = 0; i < tamaNuevo; i++) do
        for(j = 0; j < tamaNuevo; j++) do
            m[i][j] = c11[i][j]
            m[i][j+tamaNuevo] = c12[i][j]
            m[i+tamaNuevo][j] = c21[i][j]
            m[i+tamaNuevo][j+tamaNuevo] = c22[i][j]
        fin for
    fin for
    return m
FIN
```

Tiempos de ejecución de los distintos algoritmos

Tiempos resultantes al ejecutar los distintos algoritmos sobre matrices con diversos tamaños, empezando por una matriz de 64x64 para consecutivamente ir duplicando su tamaño.



Tamaño	F.Bruta(ns)	DyV(ns)	Strassen(ns)
64	16	109	78
128	31	625	532
256	203	4719	3656
512	1969	37579	25773
1024	34736	299110	183196



Eficiencia de los algoritmos analizados teóricamente

- *Algoritmo de Fuerza Bruta:*

Se trata de un algoritmo iterativo formado por tres bucles en el que las operaciones dentro de cada uno de ellos son constantes por lo tanto tendría orden t(n) ϵ $\Theta(n^3)$

- Algoritmo DyV:

Resolvemos mediante división:

- Resolución del problema mediante división:

$t(n) = cn^k$ Para $0 \leq n < b$

$t(n) = at(n/b) + cn^k$ En $n \geq b$

➢ Solución:

$$t(n) \in \begin{cases} \Theta(n^k) & \text{si } a < b^k \\ \Theta(n^k \log n) & \text{si } a = b^k \\ \Theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Sabemos que el número de llamadas recursivas es 8, el tamaño del problema lo dividimos en 2 y la parte no recursiva es $2n^2$ ya que tenemos dos grupos de dos bucles for y las operaciones dentro de ellos son constantes. Es decir tendríamos:

a = 8; b = 2 ; c = 2 y k = 2
Como a = 8 y $b^k = 2^2 = 4$, tendríamos que $a > b^2$, por lo tanto el tiempo del algoritmo sería:
t(n) ϵ $\Theta(n^{\log_b a})$ ϵ $\Theta(n^{\log_2 8})$ ϵ $\Theta(n^3)$

- Algoritmo Strassen:

Resolvemos mediante división:

El número de llamadas es recursivas es 7, el tamaño del problema lo dividimos en 2 y la parte no recursiva sería $2n^2$

a = 7; b = 2 ; c = 2 y k = 2
Como a = 7 y $b^k = 2^2 = 4$, tendríamos que $a > b^2$, por lo tanto el tiempo del algoritmo sería:
t(n) ϵ $\Theta(n^{\log_b a})$ ϵ $\Theta(n^{\log_2 7})$ ϵ $\Theta(n^{2,8})$
Se observa que los tiempos experimentales son notablemente inferiores a los teóricos.

Método híbrido

La utilización del método híbrido es útil en la resolución del problema de multiplicación de matrices ya que consigue una pequeña mejoría de tiempo en la resolución del mismo.

Además se han conseguido sucesivas mejoras a lo largo de los años, las distintas cotas de los algoritmos son las siguientes:

- Algoritmo clásico: $O(n^3)$
- V.Strassen (1969): $O(n^{2'807})$
- V.Pan (1984): $O(n^{2'795})$
- D.Coppersmith y S.Winograd (1990): $O(n^{2'376})$