2.2.2 Multiplicación de una secuencia de matrices

Suponga el siguiente problema:

Dada una secuencia de n matrices $\{A_1, A_2, \ldots . A_n\}$, donde para cada $i\ (1 \leq i \leq n)$ la matriz A_i tiene dimensión $p_{i-1}xp_i$, encontrar una forma de multiplicar las matrices que minimice el número de multiplicaciones escalares a realizar.

1. Una forma óptima de multiplicar una secuencia de matrices está determinada por el número de multiplicaciones a realizar. Para multiplicar una matriz de pxq por una de qxr son necesarias pqr operaciones escalares de multiplicación.
2. Si se multiplican tres matrices de 10 × 100, 100 × 5 y 5 × 50, se puede hacer con 7500 $\big((A_1xA_2)xA_3\big)$ o 75000 $A_1x(A_2xA_3)$ operaciones.

Sin emplear Programación Dinámica

Implica calcular el número de operaciones para cada posible orden de multiplicación de matrices. Sea P(n) el número de ordenes posibles en una secuencia de n matrices. Es sencillo ver que:

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

La solución a la ecuación anterior es:

$$P(n) = \frac{1}{n}\binom{2n-2}{n-1} = \Omega\left(\frac{4^n}{n^{\frac{3}{2}}}\right)$$

Empleando Programación Dinámica

Suponga que encontramos la forma óptima de multiplicar las matrices $\{A_1, A_2, \ldots . A_n\}$ al nivel más alto, la solución se vería como la multiplicación de dos matrices que resultan de calcular los productos $A_1, \ldots ., A_k$ y $A_{k+1}, \ldots ., A_n$, ambos en forma óptima, para algún $k\ (1 \leq k \leq n)$.

Lo anterior implica que el problema tiene una sub-estructura óptima. Ahora Suponga que se llama $m[i,j]$ al número óptimo de multiplicaciones escalares a realizar al multiplicar $\{A_i, A_{i+1}, \ldots . A_j\}$ donde $m[i,j]$ se puede escribir recursivamente como:

$$m[i,j] = \begin{cases} 0 & si\ i = j \\ \min\limits_{i \leq j < k} \{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & si\ i < j \end{cases}$$

Suponga que $s[i,j]$ es la forma en que, al nivel más alto, se divide el producto de $\{A_i, A_{i+1}, \ldots . A_j\}$ de manera óptima. Entonces,

$$m[i,j] = \begin{cases} i & si\ i = j \\ \operatorname{argmin}\limits_{i \leq j < k} \{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & si\ i < j \end{cases}$$

Si se observa, estas expresiones, se dará cuenta que es posible resolver el problema de forma recursiva tradicional, pero muchos cálculos se deberán rehacer. Esto significa que existe superposición de problemas.

Estrategia Botton-Up

La estrategia bottom-up consiste en resolver primero los subproblemas más pequeños, almacenar su solución, y luego resolver los problemas más complejos, usando los resultados que se almacenaron anteriormente. Está claro que calcular $m[i,j]$ es sencillo, pero ¿cuál tipo de problema es el que le sigue en complejidad?, el misterio es calcular $m[i,j+1]$ con $i \leq i < n$.

Algoritmo

multiplicarSecuenciaDeMatrizDeOrden(p)
    n := length(p) – 1
    for i := 0 to n
        do m[i, i] := 0
    for l := 2 to n
        do for i := 1 to n – l + 1
            do j := i + l – 1
                $k := s[i,j] := \underset{i \leq j < k}{\operatorname{argmin}} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}$
                $m[i,j] := m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$
    return *m, s*

El tiempo de ejecución de este algoritmo es $\theta(n^3)$, ahora implementando el algoritmo tenemos el código 2.2.2.1 que al ejecutarlo puede producir lo siguiente

Ejemplo 1
Cuantas matrices tiene : 5
Introduzca los ordenes de cada matriz :
p[0] = 5
p[1] = 5
p[2] = 20
p[3] = 10
p[4] = 90
p[5] = 10

M =
0 500 1250 5750 10250
-1 0 1000 5500 10000
-1 -1 0 18000 11000
-1 -1 -1 0 9000
-1 -1 -1 -1 0
S =
-1 0 0 2 0
-1 -1 1 2 3
-1 -1 -1 2 2
-1 -1 -1 -1 3
-1 -1 -1 -1 -1

La secuencia optima de multiplicacion es:
(A[0] (((A[1] A[2]) A[3]) A[4]))

Ejemplo 2
Cuantas matrices tiene : 5
Introduzca los ordenes de cada matriz :
p[0] = 10
p[1] = 8
p[2] = 5
p[3] = 30
p[4] = 40
p[5] = 5

M =
0 400 1900 8400 7350
-1 0 1200 7600 6950
-1 -1 0 6000 6750
-1 -1 -1 0 6000
-1 -1 -1 -1 0
S =
-1 0 1 1 0
-1 -1 1 1 1
-1 -1 -1 2 2
-1 -1 -1 -1 3
-1 -1 -1 -1 -1
La secuencia optima de multiplicacion es:
(A[0] (A[1] (A[2] (A[3] A[4]))))

Código 2.2.2.1 Implementación del algoritmo anterior en Java

```
/**
 *
 * @author sdelaot
 */
public class SecuenciaMatricial {
    private int[][] MM;
    private int[][] SS;
    private int miN;
    private final int infinity;
    public SecuenciaMatricial() {
        this.infinity = Integer.MAX_VALUE;
        miN = 0;
        MM = null;
        SS = null;
    }
    void ejecutarAlgoritmo( int miN, int [] p ) {
        this.miN = miN;
        crearMatrices();
        procesarMatricesSecuencialesDeOrden(p);
```

```
    imprimirMatrices();
    System.out.println( "La secuencia optima de multiplicacion es:" );
    imprimirSecuenciaOptimaMaticial(0, miN - 1, SS, miN);
    System.out.println( "\n" );
}
public void crearMatrices() {
    MM = new int[miN][];
    for (int i = 0; i < miN; i++) {
        MM[i] = new int[miN];
    }
    SS = new int[miN][];//new intP[n];
    for (int i = 0; i < miN; i++) {
        SS[i] = new int[miN];
    }
    for (int row = 0; row < miN; row++) {
        for (int col = 0; col < miN; col++) {
            MM[row][col] = -1;
        }
    }
    for (int row = 0; row < miN; row++) {
        for (int col = 0; col < miN; col++) {
            SS[row][col] = -1;
        }
    }
}
public void imprimirMatrices() {
    System.out.println( "M = " );
    for (int row = 0; row < miN; row++) {
        for (int col = 0; col < miN; col++) {
            System.out.print( MM[row][col] + " " );
        }
        System.out.println();
    }
    System.out.println( "S = " );
    for (int row = 0; row < miN; row++) {
        for (int col = 0; col < miN; col++) {
            System.out.print( SS[row][col] + " " );
        }
        System.out.println();
    }
}
public int multiplicarCosto(int i, int k, int j, int [] p ) {
    return p[i] * p[k + 1] * p[j + 1];
}
public void procesarMatricesSecuencialesDeOrden(int [] p ) {
    for (int i = 0; i < miN; i++) {
        MM[i][i] = 0;
        }
    for (int l = 2; l <= miN; l++) {
        for (int i = 0; i < (miN - l + 1); i++) {
```

```
            int j = i + l - 1;
            MM[i][j] = infinity;
            for (int k = i; k <= j - 1; k++) {
               int q = MM[i][k] + MM[k + 1][j] +
                     multiplicarCosto(i, k, j, p);
               if (q < MM[i][j]) {
                  MM[i][j] = q;
                  SS[i][j] = k;
                  }
               }
            }
         }
   }
   public void imprimirSecuenciaOptimaMaticial(int i, int j, int [][] s,
         int N) {
      if (j > i) {
         System.out.print( "(" );
         imprimirSecuenciaOptimaMaticial(i, s[i][j], s, N);
         System.out.print( " " );
         imprimirSecuenciaOptimaMaticial(s[i][j] + 1, j, s, N);
         System.out.print( ")" );
         }
      else {
         System.out.print( "A[" + i + "]" );
         }
   }
}
import java.util.Scanner;
/**
 * @author sdelaot
 */
public class ProbadorDeSecuenciaMatricial {
   public static void main(String[] args) {
      int miN = 0;
      Scanner teclado = new Scanner( System.in );
      System.out.println();
      System.out.print( "Cuantas matrices tiene : " );
      miN = teclado.nextInt();
      System.out.println( "Introduzca los ordenes de cada matriz : " );
      int [] p = new int[miN + 1];
      for (int i = 0; i < miN + 1; i++) {
         System.out.print( "p[" + i + "] = " );
         p[i] = teclado.nextInt();
         }
      System.out.println();
      SecuenciaMatricial secuenciaM = new SecuenciaMatricial();
      secuenciaM.ejecutarAlgoritmo(miN, p);
   }
}
```

Ahora mostraremos dos algoritmos de multiplicación matricial, el normal y el algoritmo de Strassen, ambos se muestran en los códigos 2.2.2.2 y 2.2.2.3

Código 2.2.2.2 Multiplicación matricial normal

```
/**
 * @author sdelaot
 */
public class Vector {
   private int [] vector;

   public Vector( int cuantos ) {
      this( new int[cuantos] );
   }

   public Vector(int[] vector) {
      this.vector = vector;
   }
   public void setElemento( int x, int valor ) {
      if( x<0 || x>=vector.length ) {
         System.out.println( "Imposible asignar " + valor +
               " indice " + x + "  es negativo" );
         }
      else {
         vector[x] = valor;
         }
   }
   public int getElemento( int x ) {
      if( x<0 ) {
         System.out.println( "Imposible devolver valor indice " +
               x + " es negativo" );
         return 0;
         }
      return vector[x];
   }

   @Override
   public String toString() {
      StringBuffer resultado = new StringBuffer();
      for( int n=0; n<vector.length; n++ ) {
         switch( calcularLongitudDe(vector[n]) ) {
            case 1:
               resultado.append( "    " );
               break;
            case 2:
               resultado.append( "   " );
               break;
            case 3:
               resultado.append( "  " );
               break;
            case 4:
```

```
                resultado.append( " " );
                break;
            }
            resultado.append( vector[n] );
            }

        return resultado.toString();
    }
    public int length() {
        return vector.length;
    }
    private int calcularLongitudDe(int numero) {
        String numeroStr = "" + numero;
        return numeroStr.length();
    }
}
/**
 * @author sdelaot
 */
public class Matriz {
    private Vector [] matriz;
    public Matriz( int y, int x ) {
      this( new Vector[y], x );
    }
    public Matriz( Vector [] matriz, int x ) {
        this.matriz = matriz;
        for( int y=0; y<matriz.length; y++ ) {
            matriz[y] = new Vector(x);
            }
    }
    public Matriz( Matriz m1 ) {
        this.matriz = m1.matriz;
    }
    public void addElemento( int y, int x, int valor ) {
        if( x<0 || y<0 ) {
            System.out.println( "Imposible asignar " + valor +
                    " indice x o y negativo" );
            return;
            }
        matriz[y].setElemento(x, valor);
    }
    public int getElemento( int x, int y ) {
        if( x<0 || y<0 ) {
            System.out.println(
                    "Imposible devolver valor indice x o y negativo" );
            return 0;
            }
        return matriz[y].getElemento(x);
    }
    @Override
```

```
    public String toString() {
        StringBuilder resultado = new StringBuilder();
        for( int y=0; y<matriz.length; y++ ) {
            resultado.append(matriz[y]);
            resultado.append( "\n" );
            }
        return resultado.toString();
    }
    public int [] length() {
        int [] xy = {
            matriz.length,      // filas y
            matriz[0].length()  // columnas x
            };
        return xy;
    }
}
/**
 *
 * @author sdelaot
 */
public class AritmeticaMatricial {
    private Matriz matriz;
    public Matriz sumar( Matriz M1, Matriz M2 ) {
        int [] xy1 = M1.length();
        int [] xy2 = M2.length();
        if( xy1[0]!=xy2[0] || xy1[1]!=xy2[1] ) {
            System.out.println( "Las matrices no pueden sumarse\n" +
                    " No son de la misma dimension " + xy1[0] + " x " + xy1[1] +
                    " y " + xy2[0] + " x " + xy2[1] );
            return null;
            }
        matriz = new Matriz(xy1[0], xy1[1]);
        for( int y=0; y<xy1[0]; y++ ) {
            for( int x=0; x<xy1[1]; x++ ) {
                int valor = M1.getElemento(x, y) + M2.getElemento(x, y);
                matriz.addElemento(y, x, valor);
                }
            }
        return matriz;
    }
    public Matriz restar( Matriz M1, Matriz M2 ) {
        int [] xy1 = M1.length();
        int [] xy2 = M2.length();
        if( xy1[0]!=xy2[0] || xy1[1]!=xy2[1] ) {
            System.out.println( "Las matrices no pueden restarse\n" +
                    " No son de la misma dimension " + xy1[0] + " x " + xy1[1] +
                    " y " + xy2[0] + " x " + xy2[1] );
            return null;
            }
        matriz = new Matriz(xy1[0], xy1[1]);
```

```
    for( int y=0; y<xy1[0]; y++ ) {
        for( int x=0; x<xy1[1]; x++ ) {
            int valor = M1.getElemento(x, y) - M2.getElemento(x, y);
            matriz.addElemento(y, x, valor);
            }
        }
    return matriz;
}
public Matriz multiplicarPorEscalar(Matriz M1, int escalar) {
    int [] xy = M1.length();
    matriz = new Matriz(xy[0], xy[1]);
    for( int y=0; y<xy[0]; y++ ) {
        for( int x=0; x<xy[1]; x++ ) {
            int valor = M1.getElemento(x, y) * escalar;
            matriz.addElemento(y, x, valor);
            }
        }
    return matriz;
}
public Matriz obtenerTranspuesta(Matriz M1) {
    int [] xy = M1.length();
    matriz = new Matriz(xy[1], xy[0]);
    for( int y=0; y<xy[0]; y++ ) {
        for( int x=0; x<xy[1]; x++ ) {
            int valor = M1.getElemento(x, y);
            matriz.addElemento(x, y, valor);
            }
        }
    return matriz;
}
public Matriz multiplicar( Matriz M1, Matriz M2 ) {
    int [] xy1 = M1.length();
    int [] xy2 = M2.length();
    if( xy1[1]!=xy2[0] ) {
        System.out.println( "No se puede realizar la multiplicacion de " );
        System.out.println( M1 + " y " + M2 );
        return null;
        }
    matriz = new Matriz(xy1[0], xy2[1]);
    for( int y=0; y<xy1[0]; y++ ) {
        for( int x=0; x<xy2[1]; x++ ) {
            int valor = 0;
            for( int k=0; k<xy1[1]; k++ ) {
                valor += (M1.getElemento(k, x) * M2.getElemento(y, k));
                //System.out.println( M1.getElemento(k, x) + " " +
                //          M2.getElemento(y, k) + " " + valor );
                }
            //System.out.println( valor );
            matriz.addElemento(x, y, valor);
            }
```

```
            }
        return matriz;
    }
}
/**
 * @author sdelaot
 */
public class ProbadorDeMatriz {
    public static void main(String[] args) {
        Matriz M1 = new Matriz( 2, 3 );
        Matriz M2 = new Matriz( 3, 2 );
        //System.out.println( M1 );
        M1.addElemento(0, 0, 2);
        M1.addElemento(0, 1, 3);
        M1.addElemento(0, 2, 5);
        M1.addElemento(1, 0, 7);
        M1.addElemento(1, 1, 2);
        M1.addElemento(1, 2, 4);
        System.out.println( M1 );
        //System.out.println( M2 );
        M2.addElemento(0, 0, 1);
        M2.addElemento(0, 1, 6);
        M2.addElemento(1, 0, 7);
        M2.addElemento(1, 1, 2);
        M2.addElemento(2, 0, 0);
        M2.addElemento(2, 1, -5);
        System.out.println( M2 );
        AritmeticaMatricial am = new AritmeticaMatricial();
        Matriz M3 = am.sumar(M1, M2);
        System.out.println( M3 );
        M3 = am.multiplicarPorEscalar(M1, 4);
        System.out.println( M3 );
        M3 = am.obtenerTranspuesta(M1);
        System.out.println( M3 );
        M3 = am.multiplicar(M1, M2);
        System.out.println( M3 );
    }
}
```

Código 2.2.2.3 Multiplicación matricial por el algoritmo de Strassen

```
/**
 * @author sdelaot
 */
public class Strassen{
    private final int COL;
    private final int FIL;
    public Strassen( int fil, int col){
        FIL = fil;
        COL = col;
```

```
}
public void ejecutarAlgoritmo(int sel){
   int [][] A = new int[FIL][COL];
   int [][] B = new int[FIL][COL];

   for( int i=0; i<A.length; i++ ){
      for( int j=0; j<B.length; j++ ){
         switch(sel) {
            case 0: // asignacion determinada
               A[i][j] = i+1;
               B[i][j] = i+1;
               break;
            case 1: // asignacion aleatoria
               A[i][j] = (int)(Math.random()*100);
               B[i][j] = (int)(Math.random()*100);
               break;
            }
         }
      }
   System.out.println(
         "**********************************************************" );
   System.out.println( "Matriz A:" );
   imprimir(A);
   System.out.println(
         "**********************************************************");
   System.out.println(
         "**********************************************************" );
   System.out.println( "Matriz B:" );
   imprimir(B);
   System.out.println(
         "**********************************************************" );
   //sum(A,B);
   //rest(A,B);
   System.out.println(
         "**********************************************************" );
   System.out.println(
         "Algoritmo de Multiplicacion de matrices, orden n^3:" );
   int[][]multResult = multiplicar(A,B);
   imprimir(multResult);
   System.out.println(
         "**********************************************************" );
   System.out.println();
   ejecutarStrassen(A,B);
}
public int[][] sumar(int[][]A,int [][]B){
   int[][]C = new int[A.length][B.length];
   for(int i = 0; i <A.length;i++){
      for(int j = 0; j < B.length;j++){
         C[i][j] = A[i][j] + B[i][j];
         }
```

```
        }
    //imprime(C);
    return C;
}
public int[][] restar(int[][]A,int [][]B){
    int[][]C = new int[A.length][B.length];
    for(int i = 0; i <A.length;i++){
        for(int j = 0; j < B.length;j++){
            C[i][j] = A[i][j] - B[i][j];
            }
        }
        //imprime(C);
        return C;
}
public int[][] multiplicar(int[][]A,int [][]B){
    int[][]C = new int[A.length][A.length];
    for(int i = 0; i < C.length; i++){
        for(int j = 0; j < C.length;j++){
            for(int k = 0; k < C.length; k++){
                C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    //imprime(C);
    return C;
}
public void imprimir(int[][]arreglo2d){
    for(int i = 0; i < arreglo2d.length; i++){
        for(int j= 0; j < arreglo2d.length; j++){

            if( arreglo2d[i][j]<10 ) {
                System.out.print( "       " );
                }
            else
            if( arreglo2d[i][j]>=10 && arreglo2d[i][j]<100 ) {
                System.out.print( "      " );
                }
            else
            if( arreglo2d[i][j]>=100 && arreglo2d[i][j]<1000 ) {
                System.out.print( "     " );
                }
            else
            if( arreglo2d[i][j]>=1000 && arreglo2d[i][j]<10000 ) {
                System.out.print( "    " );
                }
            else
            if( arreglo2d[i][j]>=10000 && arreglo2d[i][j]<100000 ) {
                System.out.print( "   " );
                }
            else
```

```
            if( arreglo2d[i][j]>=100000 && arreglo2d[i][j]<1000000 ) {
               System.out.print( "  " );
               }
            else
            if( arreglo2d[i][j]>=1000000 && arreglo2d[i][j]<10000000 ) {
               System.out.print( " " );
               }
            System.out.print(arreglo2d[i][j]);
            }
            System.out.println();
         }
}
public void ejecutarStrassen(int[][]A, int[][]B){
   int mitad = A.length/2;
   int[][]A11 = new int[mitad][mitad];
   int[][]A12 = new int[mitad][mitad];
   int[][]A21 = new int[mitad][mitad];
   int[][]A22 = new int[mitad][mitad];
   int[][]B11 = new int[mitad][mitad];
   int[][]B12 = new int[mitad][mitad];
   int[][]B21 = new int[mitad][mitad];
   int[][]B22 = new int[mitad][mitad];
   for(int i = 0; i < mitad; i++){
      for(int j = 0; j < mitad; j++){
         A11[i][j] = A[i][j];
         B11[i][j] = B[i][j];
         A12[i][j] = A[i][mitad+j];
         B12[i][j] = B[i][mitad+j];
         A21[i][j] = A[mitad+i][j];
         B21[i][j] = B[mitad+i][j];
         A22[i][j] = A[mitad+i][mitad+j];
         B22[i][j] = B[mitad+i][mitad+j];
         }
      }
   int[][]M1 = multiplicar((sumar(A11,A22)),(sumar(B11,B22)));
   int[][]M2 = multiplicar((sumar(A21,A22)),(B11));
   int[][]M3 = multiplicar((A11),(restar(B12,B22)));
   int[][]M4 = multiplicar((A22),(restar(B21,B11)));
   int[][]M5 = multiplicar((sumar(A11,A12)),(B22));
   int[][]M6 = multiplicar((restar(A21,A11)),(sumar(B11,B12)));
   int[][]M7 = multiplicar((restar(A12,A22)),(sumar(B21,B22)));
   int[][]C11 = sumar((sumar(M1,restar(M4,M5))),M7);
   int[][]C12 = sumar(M3,M5);
   int[][]C21 = sumar(M2,M4);
   int[][]C22 = sumar((sumar(restar(M1,M2),M3)),M6);
   int[][]C = new int[A.length][A.length];
   for(int i = 0; i<mitad; i++){
      for(int j= 0; j<mitad;j++){
         C[i][j] = C11[i][j];
         C[i][mitad+j] = C12[i][j];
```

```
            C[mitad+i][j] = C21[i][j];
            C[mitad+i][mitad+j] = C22[i][j];
            }
         }
      System.out.println(
            "*******************************************************" );
      System.out.println( "Algoritmo de Strassen, orden n^2.8:" );
      imprimir(C);
      System.out.println(
            "*******************************************************" );
   }
}
/**
 * @author sdelaot
 */
public class ProbadorDeStrassen {
   public static void main(String[] args) {
      int filas = 20;
      int columnas = 20;
      Strassen strassen = new Strassen(filas, columnas);
      strassen.ejecutarAlgoritmo(0);
   }
}
```

TAREA

Aquí lo que hay que hacer es

1. Verificar con matrices de algún tamaño el programa del código 2.2.2.1, es decir validar que la forma de multiplicar las matrices es óptima (introduzca los valores de las matrices desde varios archivos)
2. Utilizar ambos algoritmos para comprobar el punto 1, es decir con el algoritmo normal y con el de Strassen.
3. Justificar el algoritmo del código 2.2.2.3 con la teoría correspondiente
4. Reportar todos los puntos, si hay errores, decir cómo se salvaron, si tiene que crear más código para ello también se deberá entregar