



# PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

SEGUNDA PRUEBA  
DE  
EVALUACIÓN A DISTANCIA  
(PED2)

CURSO 2015/2016

---

David Pastor Sanz

---



## Índice

<b>1. Esquema algorítmico</b>	<b>2</b>
<b>2. Coste computacional y espacial del algoritmo</b>	<b>4</b>
2.1. Coste computacional . . . . .	4
2.2. Coste espacial . . . . .	4
<b>3. Alternativas al esquema utilizado</b>	<b>5</b>
<b>4. Datos de prueba</b>	<b>6</b>
<b>5. Código fuente</b>	<b>10</b>
5.1. Clase de lectura de archivo de entrada: Reader . . . . .	10
5.2. Clase servidor: Server . . . . .	11
5.3. Clase controladora, contiene el main: Controller . . . . .	12

# LOS COEFICIENTES BINOMIALES

## 1. Esquema algorítmico

Para la realización de esta práctica se ha utilizado un esquema algorítmico de **programación dinámica**. Dicho esquema se caracteriza por ir guardando resultados parciales, en una tabla, que se van obteniendo tras la resolución de subproblemas y que se repiten. Con esto se consigue reducir el coste computacional evitando la repetición de ciertos cálculos, como puede ocurrir en un esquema de *divide y vencerás*. Los primeros datos que se van almacenando en la tabla son las soluciones a los subproblemas más sencillos, a partir de ellos se van construyendo las soluciones a problemas mayores.

Un coeficiente binomial es el número de formas posibles en que se pueden obtener distintos subconjuntos a partir de un conjunto dado, sin que importe su orden, es decir, el número de combinaciones de elementos de dicho conjunto. La ecuación que representa el problema es:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Lo siguiente es identificar los resultados parciales. En este caso, se calculan sumando los dos resultados anteriores dados (se observará más fácilmente en la tabla). Por ejemplo, para un conjunto de tres elementos ( $\{A, B, C\}$ ) sólo hay una manera de combinar esos tres elementos en un grupo de tres. El número de combinaciones de dos elementos de ese conjunto serán de 3 ( $\{A, B\}$ ,  $\{A, C\}$  y  $\{B, C\}$ ), este se calcularía sumando el caso de número combinatorio de un conjunto de dos elementos y grupos de uno más el mismo grupo y su número combinatorio en subconjuntos de dos:

$$\binom{3}{2} = \binom{2}{1} + \binom{2}{2} = 3$$

A continuación y a partir del caso base y estableciendo el orden de llenado de la tabla, comenzando por los resultados parciales necesarios para los casos posteriores. De esta manera se podrán sustituir las llamadas recursivas, nombradas anteriormente para los esquemas de *divide y vencerás*, por consultas a la tabla. Esto evitará tener que repetir cálculos. La tabla quedaría:

	0	1	2	3	...	k-1	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
...	...		...	...	...		
n-1	...		...	...	...	$C(n-1, k-1)$	$C(n-1, k)$
n	...		...	...	...	...	$C(n-1, k-1) + C(n-1, k)$

Se puede observar en la tabla que cada número de esta se calcula sumando el que está una casilla “encima” más el de la izquierda de esa casilla de encima. Con lo cual a partir del número uno se pueden calcular todos los demás realizando su correspondiente operación, dicha suma. Esta tabla es la representación de el Triángulo de Pascal. Sólo será necesario acceder a la casilla para obtener el valor que se quiera.

El pseudocódigo de alto nivel de una posible función que realizaría esta tabla, o que devolvería el valor que se quiere calcular, es:

```

1. fun CoefBin( n, k : entero ): entero ∨ tabla
2.   var
3.     i, j: entero
4.     t: Tabla[0..n] de entero
5.   fvar
6.   si k ≤ 0 ∨ k = n entonces
7.     dev 1
8.   sino
9.     para i ← 0 hasta n hacer t[i,0] ← 1 fpara
10.    para i ← 1 hasta n hacer t[i,1] ← i fpara
11.    para i ← 2 hasta k hacer t[i,i] ← 1 fpara
12.    para i ← 2 hasta n hacer t[i,0] ← 1 fpara
13.      para j ← 2 hasta k-1 hacer
14.        si j ≤ k entonces
15.          t[i,j] ← t[i-1,j-1]+t[i-1,j]
16.        fsi
17.      fpara
18.    fpara
19.  fsi
20.  dev t[n,k] ∨ t
21. ffun

```

Observando la tabla y este código se puede ver que los casos en que el resultado sea el valor 1 se resuelve en la tabla directamente. Se marcan la fila y la columna 0 con unos, al igual que todos los valores en los que  $n = k$ . También se marcan los valores de cada fila con su índice para la primera columna, pues la suma de sus anteriores siempre sería uno más el anterior. Los demás valores se van calculando como se ha indicado más arriba,  $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ .

El algoritmo escrito en el lenguaje de programación *Java* escrito por el alumno quedaría:

```

1 public static BigInteger [][] pascal( int n, int k )
2 {
3     BigInteger [][] table = new BigInteger[n+1][k+1];
4     for( int i = 0; i <= n; i++ ) table[i][0] = new BigInteger( "1" );
5     for( int i = 1; i <= n; i++ ) table[i][1] = new BigInteger( Integer.toString( i ) );
6     for( int i = 2; i <= k; i++ ) table[i][i] = new BigInteger( "1" );
7     for( int i = 3; i <= n; i++ )
8         for( int j = 2; j <= k; j++ )
9             if( j <= k && table[i-1][j-1] != null && table[i-1][j] != null )
10                 table[i][j] = new BigInteger("0").add( table[i-1][j-1] ).add( table[i-1][j] );
11     return table;
12 }

```

Se ha decidido crear la tabla, no únicamente devolver el valor de un número combinatorio, así se podrán obtener distintos valores de distintos números combinatorios a partir de una única ejecución del algoritmo creando una única tabla, así ahorrando tiempo de ejecución. La dificultad que se tiene al hacer esto es que previamente hay que calcular cual son los mayores  $n$  y  $k$  de los números combinatorios que se quieran calcular, para así poder crear la tabla con estos dos valores mayores y poder acceder a los valores queridos. Pues no se podría obtener el valor si simplemente se crea la tabla para el número con mayor  $n$  y con un  $k$  menor que el mayor de todos los números.

## 2. Coste computacional y espacial del algoritmo

### 2.1. Coste computacional

El coste computacional de el algoritmo expuesto en el apartado anterior se puede calcular observando línea a línea el código. Primero se va a determinar el tamaño del problema. Este vendrá determinado por los números  $n$  y  $k$  que el método toma como parámetros, ambos juntos hacen un tupla con la cual se calcula el número combinatorio deseado.

- **Líneas 3 y 11:** La creación e inicialización de la tabla tiene un coste constante. La devolución mediante la sentencia *return* también tiene un coste constante.
- **Líneas 4 y 5:** Se componen de un bucle que se recorre  $n$  veces. En esas  $n$  veces que se recorre, se realizan operaciones de asignación con un coste constante. Cada una de esas líneas tienen un coste computacional de  $\mathcal{O}(n)$ .
- **Línea 6:** Es un caso parecido a las líneas 4 y 5, en este caso en vez de recorrerse y realizarse  $n$  asignaciones, se realizan  $k$  asignaciones. Con esto el coste computacional de esta línea será de  $\mathcal{O}(n)$ .
- **Líneas de 7 a 10:** El primer *for* de esta combinación de bucles se recorre  $n$  veces. En cada una de estas  $n$  veces que recorre el primer bucle, se recorre  $k$  veces el siguiente bucle (el de la línea 8). La línea 9 hará  $n \times k$  comprobaciones de que  $j$  sea menor o igual que  $k$ , todas las veces que se cumpla se realizará la asignación de la línea 10. Con todo esto en se realizan entonces en total  $n \times k$  operaciones. Esto hace que estas cuatro sentencias tengan un coste computacional de  $\mathcal{O}(nk)$ .

Realizando una suma de todos estos cálculos se tiene que  $T(nk) \in n + k + nk + 1$  (no se han contado las constantes) a lo que lleva, puesto que para el coste asintótico sólo cuentan las operaciones de mayor crecimiento, se puede decir que el coste computacional del algoritmo que realiza el cálculo de los coeficientes binomiales utilizando un esquema algorítmico de programación dinámica sería del orden de  $\mathcal{O}(nk)$ .

### 2.2. Coste espacial

Anteriormente se ha comentado la posibilidad de devolver, mediante el algoritmo, la tabla con todos los valores o sólo el valor de un coeficiente que se quiera obtener. Al haber decidido la opción de devolver la tabla y tener que crearla a partir de los mayores  $n$  y  $k$ , de todos los números combinatorios a calcular. Se “ahorrrará” en coste computacional pero esto requerirá mayor consumo de espacio de memoria. Al tener que crear una tabla con esos valores, con  $n$  filas y  $k$  columnas, el coste espacial será de  $\Theta(nk)$  (donde  $n$  y  $k$  son los valores mayores de todos los que se desean calcular).

Este coste espacial se podría mejorar, perdiendo en tiempo de ejecución. Esto se podría hacer creando un vector, en vez de una tabla, el cual iría calculando fila a fila hasta llegar al número buscado. Se calcularían primero la fila uno, luego la dos, así hasta la  $n$ . Cada fila iría calculando su “casilla” a partir del valor anterior de dicha casilla y la previa. Esa última fila  $n$  tendrá  $k$  elementos. Entonces el coste espacial de esta manera sería  $\Theta(k)$ . Se conseguiría un coste espacial mejor pero el coste computacional sería mayor, tendría una costante multiplicativa mayor, (dentro del mismo orden) ya que se tendría que ejecutar más veces el algoritmo, en vez de sólo acceder al valor de la tabla.

### 3. Alternativas al esquema utilizado

Una posible alternativa para solucionar el problema sería utilizar un esquema de *divide y vencerás*. Este divide en subproblemas hasta alcanzar con la solución querida. Se implementa de manera recursiva siguiendo la expresión:

$$\binom{n}{k} = \begin{cases} 1 & , \text{ si } k = 0 \text{ o } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & , \text{ si } 0 < k < n \end{cases}$$

Para demostrar que el número de cálculos sería mucho mayor que usando un esquema de *programación dinámica* se va a representar dicha expresión en pseudocódigo, en este se devolverá el valor del número combinatorio y no su tabla, ya que no se va almacenando para calcular los siguientes pasos:

```

1. fun CoefBinRec( n, k : entero ): entero
2.   si k = 0 ∨ k = n entonces
3.     dev 1
4.   sino
5.     dev CoefBinRec( n-1, k-1 ) + CoefBinRec( n-1, k )
6.   fsi
7. ffun

```

Las fórmulas para hallar el coste computacional de los algoritmos recursivos mediante reducción (que cada llamada recursiva se hace restando el tamaño del problema) son:

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases} \quad T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

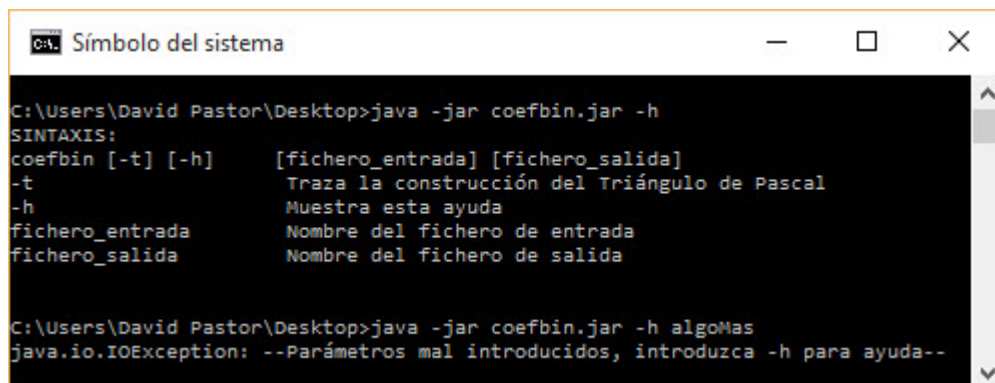
Donde  $a$  es el número de subproblemas en los que se descompone el problema inicial;  $b$  es el factor de reducción del problema; y  $k$  es el coste de las combinaciones parciales. En este caso tenemos que  $k = 0$ , ya que el coste del caso base es constante;  $a = 2$ , porque se realizan dos llamadas recursivas en cada ejecución;  $b = 1$ , el problema se va reduciendo en una unidad. Con estos datos y aplicando la fórmula anterior se tiene que  $a \geq 1$ , por tanto  $T(n) \in \Theta(2^n)$ . Este coste es mucho mayor que el que se obtiene al realizar mediante *programación dinámica*.

Otra opción podría ser calcular el resultado directamente usando la fórmula y calculando los factoriales. Estos se podrían calcular de manera recursiva con un coste computacional también exponencial usando un esquema de *divide y vencerás*. Calculándolo mediante *programación dinámica* el coste sería el calcular un vector de  $n$  factorial (ya que  $n$  siempre será más grande que  $k$ , y a partir de ese vector obtener el valor de factorial de  $k$  y de  $n-k$  y aplicar la fórmula. Su coste espacial sería  $\Theta(n)$ . El problema de hacerlo así sería a la hora de representar la traza.

## 4. Datos de prueba

Al igual que en la primera práctica, se van a mostrar diferentes datos introducidos para probar el programa, entre ellos se probarán con datos que hagan fallar el programa para probar su robustez.

- I*) Para el funcionamiento del comando `-h` se mostrará por pantalla la ayuda (). Si se introducen parámetros de más junto a dicho comando se mostrará un mensaje de error. Se puede observar en la *Figura 1*.

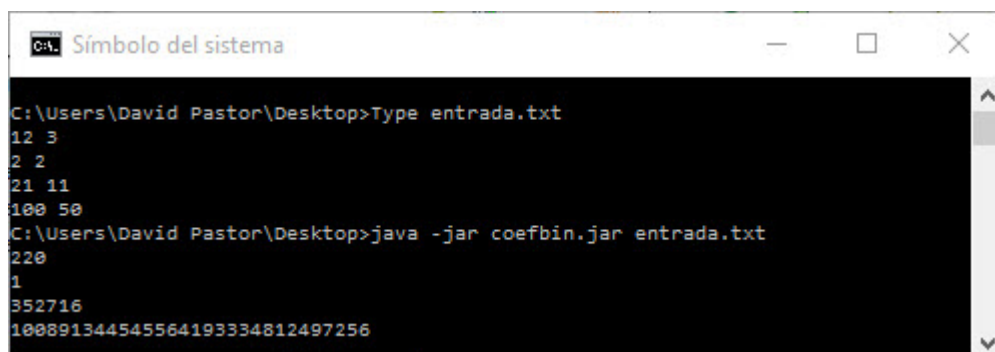


```
C:\Users\David Pastor\Desktop>java -jar coefbin.jar -h
SINTAXIS:
coefbin [-t] [-h]      [fichero_entrada] [fichero_salida]
-t                      Traza la construcción del Triángulo de Pascal
-h                      Muestra esta ayuda
fichero_entrada         Nombre del fichero de entrada
fichero_salida          Nombre del fichero de salida

C:\Users\David Pastor\Desktop>java -jar coefbin.jar -h algoMas
java.io.IOException: --Parámetros mal introducidos, introduzca -h para ayuda--
```

Figura 1: Muestra del comando `-h` y un error al meter más parámetros junto a él

- II*) A continuación se mostrará la salida dada metiendo el fichero de entrada. Para mostrar el archivo en *cmd* se utiliza el comando `Type archivo.txt`. En la *Figura 2* se puede observar el correcto funcionamiento para la entrada mostrada en la imagen.



```
C:\Users\David Pastor\Desktop>Type entrada.txt
12 3
2 2
21 11
100 50
C:\Users\David Pastor\Desktop>java -jar coefbin.jar entrada.txt
220
1
352716
100891344545564193334812497256
```

Figura 2: Fichero de entrada y su salida correspondiente por pantalla de comando



Si los números de la entrada no son enteros naturales (números menores que 0 u otros símbolos) el programa devolverá un mensaje de error. Lo mismo ocurrirá si el número  $k$  (segundo de la línea que se lee) sea mayor que el número  $n$ . En la *Figura 3* se muestra un ejemplo de este tipo de errores.

```

C:\Users\David Pastor\Desktop>Type entradaError.txt
15 13
1 8
6 3
C:\Users\David Pastor\Desktop>java -jar coefbin.jar entradaError.txt
java.lang.IllegalStateException: -El primer dato del binomio debe ser mayor que el segundo-
    
```

Figura 3: Fichero de entrada con errores y su salida

III ) Ahora se probará la salida de la traza por pantalla usando dos parámetros de entrada, el comando `-t` y el archivo de entrada. En la *Figura 4* se puede ver cual sería la ejecución del programa para un cierto archivo de entrada, mostrado también en esta, y su correcta salida impresa.

```

C:\Users\David Pastor\Desktop>Type entrada2.txt
2 1
2 2
20 10
C:\Users\David Pastor\Desktop>java -jar coefbin.jar -t entrada2.txt
2
1
184756
La tabla del coeficiente binomial de 2 1 es:
1
1 1
1 2
La tabla del coeficiente binomial de 2 2 es:
1
1 1
1 2 1
La tabla del coeficiente binomial de 20 10 es:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11
1 12 66 220 495 792 924 792 495 220 66 12
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14
1 15 105 455 1365 3003 5005 6435 5005 3003 1365 455 105 15
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19
1 20 190 1140 4845 15504 38760 77520 125970 167960 184756
    
```

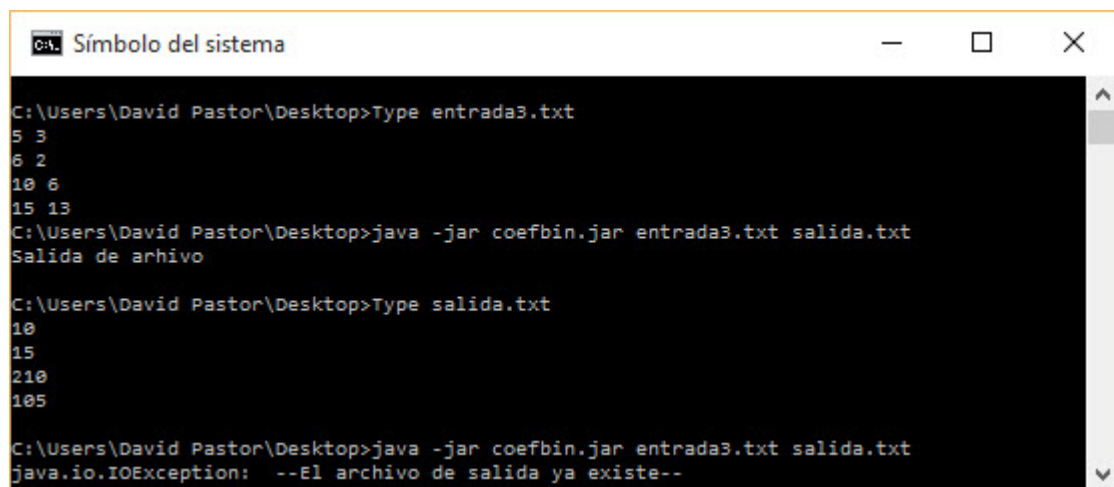
Figura 4: Fichero de entrada con su resultado y traza mostrada por pantalla

La traza que se muestra es el triángulo de pascal de cada número combinatorio que se quiere calcular. Tal como se puede observar en la *Figura 4*. El número buscado es el que se encuentra más abajo y más a la derecha, indica la posición  $(n,k)$  de la tabla.

Los posibles errores que puede haber en este caso son los mismos que para el anterior. Si el archivo de entrada no existe o contiene datos que no son válidos no se ejecutará el algoritmo y simplemente mostrará un error por pantalla.

IV) De nuevo se demostrará el funcionamiento del programa con dos parámetros de entrada, esta vez metiendo el archivo de entrada y el archivo de salida. Respecto al archivo de salida si ya existe, se mostrará por pantalla un mensaje de error indicando que dicho archivo ya existe y el algoritmo no se ejecutará. Si existe algún error en el algoritmo, por datos de entrada erróneos, se mostrará el mensaje de error correspondiente dentro del archivo de salida creado, siempre que no exista ya. Cuando se crea un archivo de salida se mostrará el mensaje de **salida de archivo** por la pantalla de comando.

En la siguiente imagen (*Figura 5*) se muestra una posible ejecución del programa, mostrando también el fichero de salida que genera. Además a continuación se muestra el error que generaría al ejecutar el programa con los mismos parámetros, una vez que ya se ha creado el archivo de salida.



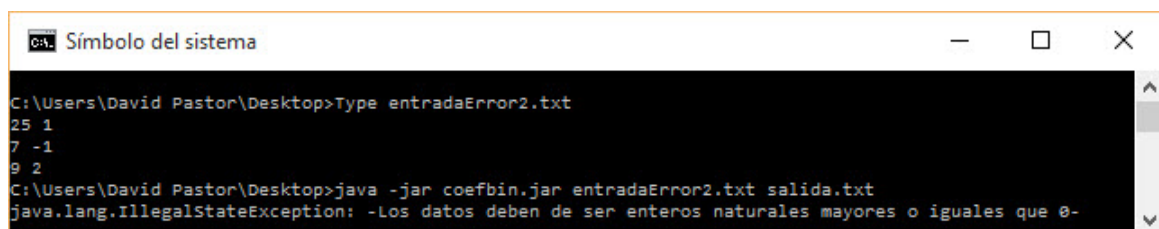
```
C:\Users\David Pastor\Desktop>Type entrada3.txt
5 3
6 2
10 6
15 13
C:\Users\David Pastor\Desktop>java -jar coefbin.jar entrada3.txt salida.txt
Salida de archivo

C:\Users\David Pastor\Desktop>Type salida.txt
10
15
210
105

C:\Users\David Pastor\Desktop>java -jar coefbin.jar entrada3.txt salida.txt
java.io.IOException: --El archivo de salida ya existe--
```

Figura 5: Fichero de entrada con su resultado mostrado en un archivo de salida

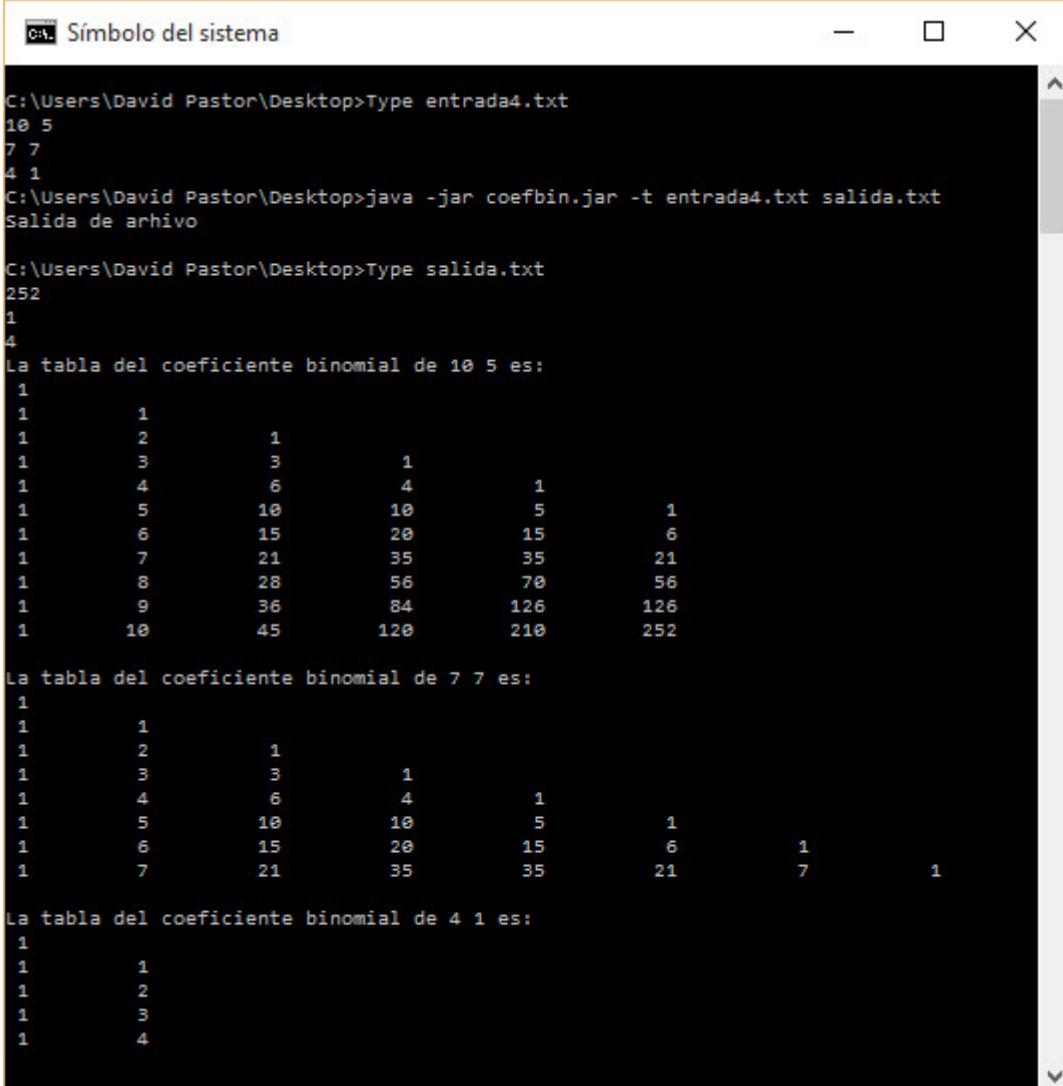
En la *Figura 6* se muestra la ejecución del programa con los mismos parámetros pero con un error en el archivo de entrada.



```
C:\Users\David Pastor\Desktop>Type entradaError2.txt
25 1
7 -1
9 2
C:\Users\David Pastor\Desktop>java -jar coefbin.jar entradaError2.txt salida.txt
java.lang.IllegalStateException: -Los datos deben de ser enteros naturales mayores o iguales que 0-
```

Figura 6: Fichero de entrada con error

- ∪) Por último se va a proceder a probar el programa introduciendo los tres argumentos posibles: `-t`, que indicará que se muestre la traza; `archivo_entrada`; y `archivo_salida`, que mostrará la solución al archivo de entrada junto a la traza que realiza, en caso de error y que el archivo de salida no exista, se mostrará el error en el archivo de salida. Se puede observar como se ejecuta en la *Figura 7* que se muestra a continuación.



```
C:\Users\David Pastor\Desktop>Type entrada4.txt
10 5
7 7
4 1
C:\Users\David Pastor\Desktop>java -jar coefbin.jar -t entrada4.txt salida.txt
Salida de archivo

C:\Users\David Pastor\Desktop>Type salida.txt
252
1
4
La tabla del coeficiente binomial de 10 5 es:
1
1      1
1      2      1
1      3      3      1
1      4      6      4      1
1      5      10     10     5      1
1      6      15     20     15     6
1      7      21     35     35     21
1      8      28     56     70     56
1      9      36     84     126    126
1     10     45     120    210    252

La tabla del coeficiente binomial de 7 7 es:
1
1      1
1      2      1
1      3      3      1
1      4      6      4      1
1      5      10     10     5      1
1      6      15     20     15     6      1
1      7      21     35     35     21     7      1

La tabla del coeficiente binomial de 4 1 es:
1
1      1
1      2
1      3
1      4
```

Figura 7: Fichero de entrada con su resultado mostrado en un archivo de salida junto a la traza

## 5. Código fuente

En esta sección se muestra el código fuente utilizado para la realización del programa.

### 5.1. Clase de lectura de archivo de entrada: Reader

```

1 package predaPED2;
2
3 import java.io.File;
4 import java.io.FileReader;
5 import java.io.BufferedReader;
6 import java.io.IOException;
7 import java.io.FileNotFoundException;
8 import java.lang.NumberFormatException;
9 import java.util.ArrayList;
10 /**
11  * Se encarga de leer un archivo y verificar que la entrada es correcta
12  * @author David Pastor Sanz
13  */
14 public abstract class Reader
15 {
16     /**
17      * Devuelve un array de números binomiales, si estos han sido
18      * correctamente leídos del archivo de entrada. Comprueba si n es mayor
19      * que k y que ambos sean números mayores que 0
20      * @return El array de números binomiales.
21      */
22     public static int [][] getBinomialList( String file ) throws FileNotFoundException,
        IOException, IllegalStateException, NumberFormatException
23     {
24         File rFile = new File( file );
25         int [][] table;
26         try
27         {
28             if( rFile.exists() )
29             {
30                 try
31                 {
32                     FileReader fileReader = new FileReader( rFile );
33                     BufferedReader bufferedReader = new BufferedReader( fileReader );
34                     ArrayList<String[]> numbers = new ArrayList<String[]>();
35                     String next;
36                     while( bufferedReader.ready() )
37                     {
38                         next = bufferedReader.readLine();
39                         numbers = checkLine( next, numbers );
40                     }
41                     bufferedReader.close();
42                     table = new int [numbers.size()][2];
43                     for( int i = 0; i < numbers.size(); i++ )
44                         table = addLine( numbers.get( i ), table, i );
45                 }
46                 catch( NumberFormatException e )
47                 {
48                     throw( e );
49                 }
50             }
51             else throw new FileNotFoundException();
52         }
53         catch( FileNotFoundException ef )
54         {

```

```

55         throw new FileNotFoundException( "—Fichero no encontrado—" );
56     }
57     return table;
58 }
59
60 private static ArrayList<String[]> checkLine( String line , ArrayList<String[]> table
61     ) throws IllegalStateException
62 {
63     String[] split = line.split( " " );
64     if( split.length != 2 ) throw new IllegalStateException( "—Cada número
65         combinatorio deben ser dos números enteros naturales—" );
66     table.add( split );
67     return table;
68 }
69
70 private static int[][] addLine( String[] next , int[][] table , int index ) throws
71     IOException , NumberFormatException
72 {
73     int n , k;
74     try
75     {
76         n = Integer.parseInt( next[0] );
77         k = Integer.parseInt( next[1] );
78         if( n < k )
79             throw new IllegalStateException( "—El primer dato del binomio debe ser mayor que
80                 el segundo—" );
81         if( n < 0 || k < 0 )
82             throw new IllegalStateException( "—Los datos deben de ser enteros naturales
83                 mayores o iguales que 0—" );
84         table[index][0] = n;
85         table[index][1] = k;
86     }
87     catch( NumberFormatException e )
88     {
89         throw new NumberFormatException( "—Los datos deben de ser enteros naturales
90             mayores o iguales que 0—" );
91     }
92     return table;
93 }

```

## 5.2. Clase servidor: Server

```

1 package predaPED2;
2
3 import java.math.BigInteger;
4
5 /**
6  * Este clase se encarga de realizar el algoritmo que calculará el
7  * valor del número combinatorio
8  * @author David Pastor
9  */
10 public abstract class Server
11 {
12     /**
13     * Devuelve la tabla que representa el triángulo de pascal del número
14     * cominatorio
15     * @param n Primer valor del número combinatorio
16     * @param k Segundo valor del número combinatorio
17     * @return Tabla que representa el triángulo de pascal del número combinatorio
18     */

```

```

19 public static BigInteger [][] pascal( int n, int k )
20 {
21     BigInteger [][] table = new BigInteger[n+1][k+1];
22     for( int i = 0; i <= n; i++ ) table[i][0] = new BigInteger( "1" );
23     for( int i = 1; i <= n; i++ ) table[i][1] = new BigInteger( Integer.toString( i ) );
24     for( int i = 2; i <= k; i++ ) table[i][i] = new BigInteger( "1" );
25     for( int i = 3; i <= n; i++ )
26         for( int j = 2; j <= k; j++ )
27             if( j <= k && table[i-1][j-1] != null && table[i-1][j] != null )
28                 table[i][j] = new BigInteger( "0" ).add( table[i-1][j-1] ).add( table[i-1][j] );
29     return table;
30 }
31
32 /**
33  * Devuelve el resultado, a partir de la tabla, de un
34  * número combinatorio
35  * @param n Primer valor del número combinatorio
36  * @param k Segundo valor del número combinatorio
37  * @param table La tabla de donde se calcula
38  * @return El valor del número combinatorio ( n, k )
39  */
40 public static int getResult( int n, int k, BigInteger [][] table )
41 {
42     return table[n][k].intValue();
43 }
44
45 /**
46  * Devuelve la traza de un número combinatorio,
47  * esta es el triángulo de Pascal
48  * @param n Primer valor del número combinatorio
49  * @param k Segundo valor del número combinatorio
50  * @param table La tabla donde se calcula
51  * @return El triángulo de Pascal del número ( n, k )
52  */
53 public static String trace( int n, int k, BigInteger [][] table )
54 {
55     String t = String.format( "La tabla del coeficiente binomial de %d %d es:\n\r 1\n\r",
56                               , n, k );
57     for( int i = 1; i <= n; i++ )
58     {
59         t += " 1";
60         for( int j = 1; j <= k; j++ )
61             if( table[i][j] != null )
62                 t += String.format( "%10d ", table[i][j] );
63         t += "\n\r";
64     }
65     return t;
66 }

```

### 5.3. Clase controladora, contiene el main: Controller

```

1 package predaPED2;
2
3 import java.math.BigInteger;
4 import java.io.IOException;
5 import java.io.FileNotFoundException;
6 import java.io.File;
7 import java.io.PrintStream;
8 import java.io.FileOutputStream;

```

```

9 import java.io.FileDescriptor;
10
11 /**
12  * Controla las demás clases y contiene el método main del programa
13  * @author David Pastor
14  */
15 public class Controller
16 {
17     public static void main( String[] args )
18     {
19         try
20         {
21             getArguments( args );
22         }
23         catch( IOException | IllegalStateException | NumberFormatException e )
24         {
25             System.out.println( e );
26         }
27     }
28
29     private static void getArguments( String[] args ) throws IOException
30     {
31         try
32         {
33             int[][] parameters;
34             BigInteger[][] table;
35             if( args.length == 0 )
36                 System.out.println( "—No se han detectado parámetros—" );
37             if( args.length == 1 && args[0].equals( "-h" ) ) System.out.println( help() );
38             else
39             {
40                 if( args.length == 1 )
41                 {
42                     if( args[0].equals( "-t" ) )
43                         throw new IOException( "—Parámetros mal introducidos, introduzca -h para ayuda—" );
44                     else
45                     {
46                         parameters = Reader.getBinomialList( args[0] );
47                         table = getTable( parameters );
48                         resultWithoutTrace( table, parameters );
49                     }
50                 }
51                 else if( args.length == 2 )
52                 {
53                     if( args[0].equals( "-h" ) ) throw new IOException( "—Parámetros mal introducidos, introduzca -h para ayuda—" );
54
55                     if( args[0].equals( "-t" ) )
56                     {
57                         parameters = Reader.getBinomialList( args[1] );
58                         table = getTable( parameters );
59                         resultWithTrace( table, parameters );
60                     }
61                     else
62                     {
63                         parameters = Reader.getBinomialList( args[0] );
64                         table = getTable( parameters );
65                         printInFile( args[1] );
66                         resultWithoutTrace( table, parameters );
67                         System.setOut( new PrintStream( new FileOutputStream( FileDescriptor.out ) ) );
68                         System.out.println( "Salida de archivo" );
69                     }

```

```

70     }
71     else if( args.length == 3 )
72     {
73         parameters = Reader.getBinomialList( args[1] );
74         table = getTable( parameters );
75         if( args[0].equals( "-t" ) )
76         {
77             printInFile( args[2] );
78             resultWithTrace( table, parameters );
79             System.setOut( new PrintStream( new FileOutputStream( FileDescriptor.out ) ) );
80             System.out.println( "Salida de archivo" );
81         }
82         else throw new IOException( "--Parámetros mal introducidos, introduzca -h para ayuda--" );
83     }
84 }
85 }
86 catch( IOException | IllegalStateException | NumberFormatException e )
87 {
88     System.out.println( e );
89 }
90 }
91
92 private static BigInteger [][] getTable( int [][] parameters )
93 {
94     int n = 0;
95     int k = 0;
96     for( int i = 0; i < parameters.length; i++ )
97     {
98         if( parameters[i][0] > n ) n = parameters[i][0];
99         if( parameters[i][1] > k ) k = parameters[i][1];
100     }
101     return Server.pascal( n, k );
102 }
103
104 private static void resultWithoutTrace( BigInteger [][] table, int [][] parameters )
105     throws IOException
106 {
107     int n, k;
108     for( int i = 0; i < parameters.length; i++ )
109     {
110         n = parameters[i][0];
111         k = parameters[i][1];
112         System.out.println( table[n][k] );
113     }
114 }
115 private static void resultWithTrace( BigInteger [][] table, int [][] parameters ) throws
116     IOException
117 {
118     resultWithoutTrace( table, parameters );
119     int n, k;
120     for( int i = 0; i < parameters.length; i++ )
121     {
122         n = parameters[i][0];
123         k = parameters[i][1];
124         System.out.println( Server.trace( n, k, table ) );
125     }
126 }
127 private static String help()
128 {
129     StringBuffer sb = new StringBuffer();

```



```

130
131     sb.append( "SINTAXIS:\n" );
132     sb.append( "coefbin [-t] [-h]      [fichero_entrada] [fichero_salida]\n" );
133     sb.append( "-t                      Traza la construcción del Triángulo de Pascal\n"
134               );
135     sb.append( "-h                      Muestra esta ayuda\n" );
136     sb.append( "fichero_entrada      Nombre del fichero de entrada\n" );
137     sb.append( "fichero_salida       Nombre del fichero de salida\n" );
138
139     return sb.toString();
140 }
141
142 private static void printInFile( String fileName ) throws IOException
143 {
144     File file = new File( fileName );
145     if( file.exists() && file.isFile() ) throw new IOException( " —El archivo de salida
146                               ya existe—" );
147     PrintStream out = null;
148     try
149     {
150         out = new PrintStream( new FileOutputStream( fileName ) );
151     }
152     catch ( FileNotFoundException e )
153     {
154         System.out.println( e + " —No se ha encontrado la ruta del archivo—" );
155     }
156     System.setOut( out );
157 }

```

---

Para la realización de esta práctica el alumno se ha basado en la bibliografía básica de la asignatura.

El programa ha sido escrito en el lenguaje de programación Java mediante el IDE *eclipse*.

Este documento ha sido realizado en  $\text{\LaTeX}$  con el editor de textos *TexMaker*.