PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

PED2

Miroslav Vladimirov

email:miro.kv89@gmail.com

1.- ENUNCIADO DE LA PRÁCTICA: Los coeficientes binomiales Los coeficientes binomiales, números combinatorios o combinaciones se utilizan en combinatoria para calcular el número de formas en que se pueden extraer subconjuntos a partir de un conjunto dado. El número de formas de escoger k elementos a partir de un conjunto de n, se denota como$C(n,k)$ o $\binom{n}{k}$, y se define conforme a la siguiente expresión:

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

Supongamos, por ejemplo, que se tiene un conjunto con 6 objetos A,B,C,D,E,F, y se desea conocer el número de subconjuntos posibles de dos elementos (sin importar el orden de elección). Equivaldría a calcular:

$$\binom{6}{2} = \frac{6!}{2!(6-2)!} = \frac{6!}{2!*4!} = \frac{720}{2*24} = 15$$

Es decir, es posible construir 15 subconjuntos de dos elementos a partir del conjunto original. Dichos subconjuntos serían: $\{\{A,B\},\{A,C\},\{A,D\},\{A,E\},\{A,F\},\{B,C\},\{B,D\},\{B,E\},\{B,F\},\{C,D\},\{C,E\},\{C,F\},\{D,E\},\{D,F\},\{E,F\}\}$
La expresión (1) puede escribirse de forma recursiva como:

$$\binom{n}{k} = \begin{cases} 1, & si\ k = 0\ ó\ k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k}, & x \geq 0 \end{cases}$$

El algoritmo que implementa esta ecuación recursiva resulta ser de complejidad exponencial por la repetición de los cálculos que realiza. Sin embargo, es posible diseñar un algoritmo más eficiente aplicando la idea del Triángulo de Pascal, descrita en la página 131 del libro de la asignatura. En base a esta idea, se pide diseñar e implementar un algoritmo con el esquema de programación dinámica para el cálculo de coeficientes binomiales.

•**Respuestas a los siguientes apartados:**

1. Describa el esquema algorítmico utilizado y como se aplica al problema, incluyendo las ecuaciones que representan el problema, y la tabla de resultados parciales.

Para la realización del programa se ha utilizado el algoritmo de la página 132 con el que vamos a construir el triángulo de Pascal. El coste total que vamos a conseguir es aproximadamente de $2n + k + nk$ ó $O(nk)$. Para generar los resultados se utilizará una matriz de enteros. El algoritmo opera de la siguiente manera:

-Primero recorre toda la columna de la izquierda del triangulo y la pone a 1
-Después recorre la segunda columna, desde el segundo elemento hasta el final, introduciendo valores que empiezan en 1 y acaban en n
-Recorre todas las posiciones del vector desde 2 hasta k y pone el valor 1 en los casos en los que n es igual a k (Posiciones 1,1; 2,2; 3,3...) -Por ultimo nos encontramos con el bucle de coste O(nk)". Se recorre la matriz a lo largo del eje n, desde 3 hasta n; y por cada valor de n se recorre la matriz a lo largo del vector k desde 2 hasta n-1. En el programa realizado se ha simplificado un poco este ultimo caso, para no contar valores a partir de cuando n y k tienen el mismo valor. Quedando el codigo del programa de la siguiente manera:

```
public int calcular(int n, int k){
    this.n=n;
    this.k=k;
    this.matriz=new int[n+1][k+1];
    if((n-k)==1){matriz[n][k]=n;return matriz[n][k];}
    if(k<1 || k==n){matriz[n][k]=1;return matriz[n][k];}
    else{
        for (int i=0; i<=n; i++) matriz[i][0]=1;
        for (int i=1; i<=n; i++) matriz[i][1]=i;
        for (int i=2; i<=k; i++) matriz[i][i]=1;
        for (int i=3; i<=n; i++){
            for(int j=2; j<n; j++){
                if(j<=k && j<i){
                    matriz[i][j] = matriz[i-1][j-1]+matriz[i-1][j];
                }
            }
        }
    }
    return matriz[n][k];
}
```

Observamos que dentro del condicional "if" se ha incluido el criterio de verificación "j¡i" y con ello nos evitamos rellenar la mitad de la tabla quedando el coste computacional en aproximadamente O($nk/2$). Por otro lado también hemos conseguido mejorar el coste de almacenamiento, reduciéndolo prácticamente a la mitad: $O_{almacenamiento} = (n+1)k/2$. A pesar de que quizás el coste de almacenamiento es alto por estar almacenando toda la matriz, esto se compensa con la velocidad y claridad de codigo que adquiere el algoritmo con diferencia a hacerlo recursivo.

2. Analice el coste computacional y espacial del algoritmo.

Como hemos mencionado en el apartado anterior, conseguimos un algoritmo con coste computacional de $O(nk/2)$ y coste espacial de $O_e = (n+1)k/2$. Hay que destacar que los costes que obtenemos al dibujar la traza es exactamente igual al coste del algoritmo.

3. Exponga alternativas al esquema utilizado si las hay, y compare su coste con el de la solución realizada.

Como alternativa a la resolución propuesta, se ha generado otra opción que se activara en el programa mediante la opción "-c". Con el siguiente algoritmo se ha tratado de reducir al máximo los cálculos realizados desenrollando el coeficiente binomial y simplificándolo lo máximo posible.

```
public int calcular(int n, int k){
    boolean optimo=((2*k)>n);
    if(optimo){
        for(int i=n; i>k; i--){
            String miI=(i+"");
            totaln=totaln.multiply(new BigInteger(miI));
        }
        for (int j=1; j<=n-k; j++){
            String miJ=(j+"");
            totalk=totalk.multiply(new BigInteger(miJ));
        }
    }
    else{
        for(int i=n; i>(n-k); i--){
            String miI=(i+"");
            totaln=totaln.multiply(new BigInteger(miI));
        }
        for (int j=1; j<=k; j++){
            String miJ=(j+"");
            totalk=totalk.multiply(new BigInteger(miJ));
        }
        resultado=totaln.divide(totalk);
    }
    return resultado.intValue();
}
```

Dicho algoritmo funciona comprobando los valores de n y k, y eliminando cálculos innecesarios. Lo primero se comprueba que $2 * k > n$ y en caso de serlo, tomamos el primer camino que es el mas optimo:

$$\binom{20}{3} = \frac{20!}{3!(20-3)!} = \frac{20*19*18*\cancel{17!}}{3*2*1*\cancel{17!}} = \frac{20*19*18}{3*2*1} = \frac{6840}{6} = 1140$$

Hemos conseguido simplificar básicamente todo el cálculo del coeficiente quitándonos 17 operaciones que ademas había que hacer 2 veces, por lo que el aumento en velocidad es significante. Observamos que este código en vez de empezar a multiplicar desde 1, empieza desde el numero mayor, que es por lo que conseguimos ahorrarnos los cálculos.

$$\binom{20}{14} = \frac{20!}{14!(20-14)!} = \frac{20*19*18*17*16*15*\cancel{14!}}{\cancel{14!} * 6!} = \frac{20*19*18*17*16*15}{1*2*3*4*5*6} =$$

$$= \frac{27907200}{720} = 38760$$

En este segundo caso se observa la otra posibilidad, si $2 * k < n$. Ahora la parte que habría que simplificar es la de la izquierda, puesto que es mayor y nos quitaríamos mas cálculos al hacerlo.Es un caso menos favorable que el anterior, pero se observa que aun así no hay demasiados operando. En el caso peor, con esta implementación habría n elementos (20 en este caso), n/2 en el dividendo y n/2 en el divisor. Pero ese es un caso particular y se cumple solo cuando k=n/2. Aun en el caso peor vemos que conseguimos n elementos, y tendremos un coste lineal O(n). El caso medio sin embargo esta en O(n/2). Acerca del coste espacial, en realidad no se necesita memoria para almacenar una operación a excepción del resultado.
Pero también podemos señalar desventajas a este algoritmo. Si, tiene un coste lineal en el caso peor, pero por mas operaciones que se realicen, esto no mejorara por no usar memoria. Ademas esta opción esta desprovista de una traza por el hecho de ser una mera operación aritmética, con lo que no es válida para la realización del programa. Por otro lado, el coste de la primera implementación requiere de una sola ejecución para generar una gran cantidad de valores, que al guardarlos en memoria podrán ser consultados con un coste constante en un futuro, sin necesidad de volver a calcular nada, ni de hacer el triangulo de Pascal. Eso no se cumple si los coeficientes son mayores que los que ya hay guardados claro. Por ejemplo, una vez calculado 20-10, el coste para accedes a

16-8 podría ser constante, para 30-15 sin embargo habría que hacer cálculos.

4. Describa los datos de prueba utilizados y los resultados obtenidos con ellos.

Se han utilizado los datos propuestos en la practica, añadiendo otro mas para probar con numeros mayores. Estos son los resultados obtenidos:

Entrada	Salida
2 1	2
2 2	1
6 2	15
20 10	184756
30 15	155117520
50 40	1682343578

Puesto que la capacidad de "int" no es muy grande, se considera que si n es superior a 34 es probable que haya que actualizar el programa con una matriz con números mas grandes, Long o BigInteger.
Imagen de la traza 20-10. El valor buscado se encuentra abajo a la izquierda:

A	B	C	D	E	F	G	H	I	J	K
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	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
1	13	78	286	715	1287	1716	1716	1287	715	286
1	14	91	364	1001	2002	3003	3432	3003	2002	1001
1	15	105	455	1365	3003	5005	6435	6435	5005	3003
1	16	120	560	1820	4368	8008	11440	12870	11440	8008
1	17	136	680	2380	6188	12376	19448	24310	24310	19448
1	18	153	816	3060	8568	18564	31824	43758	48620	43758
1	19	171	969	3876	11628	27132	50388	75582	92378	92378
1	20	190	1140	4845	15504	38760	77520	125970	167960	184756

Se ha diseñado la traza para que tenga tiempos de espera por lo que puede apreciarse el recorrido que hace el algoritmo.

5. Listado COMPLETO del codigo fuente:

```java
/*
 * Nombre: Miroslav Krasimirov Vladimirov
 * email: mkrasimir4@alumno.uned.es / miro.kv89@gmail.com
 * NIE: X4780953N
 * tlfn: 676867565
 */
public interface Coef {
    //Realiza los calculos
    public int calcular(int n, int k);
    //Devuelve el resultado de las operaciones
    public int getResultado();
}
```

```java
public class CoefB implements Coef {
    int[][] matriz= null;
    int n,k;
    public CoefB(CoefB coef){
        new CoefC(coef.getN(),coef.getK());
    }
    public int getN(){return n;}
    public int getK(){return k;}
    public CoefB(int n, int k){
        this.n=n;
        this.k=k;
        this.matriz=new int[n][k];
    }
    public int calcular(int n, int k){
        this.n=n;
        this.k=k;
        this.matriz=new int[n+1][k+1];
        if((n-k)==1){matriz[n][k]=n;return matriz[n][k];}
        if(k<1 || k==n){matriz[n][k]=1;return matriz[n][k];}
        else{
            for (int i=0; i<=n; i++) matriz[i][0]=1;
            for (int i=1; i<=n; i++) matriz[i][1]=i;
            for (int i=2; i<=k; i++) matriz[i][i]=1;
            for (int i=3; i<=n; i++){
                for(int j=2; j<n; j++){
                    if(j<=k && j<i){
                        matriz[i][j] = matriz[i-1][j-1]+matriz[i-1][j];
                    }
                }
            }
        }
        return matriz[n][k];
    }
    public int[][] getMatriz(){
        return matriz;
    }
    public int getResultado(){
        calcular(n,k);
        return matriz[n][k];
    }
    public void mostrarTraza(){
        new Traza(matriz,n,k);
    }
}
```

```java
import java.math.BigInteger;
public class CoefC implements Coef {
    //Implementacion con menor coste, pero sin traza
    BigInteger totaln= new BigInteger("1");
    BigInteger totalk= new BigInteger("1");
    BigInteger resultado= new BigInteger("0");
    int n,k;
    public CoefC(CoefC coef){
        new CoefC(coef.getN(),coef.getK());
       }
    public int getN(){return n;}
    public int getK(){return k;}
    public CoefC(int n, int k){
        this.n=n;
        this.k=k;
        calcular(n,k);
    }
    public int calcular(int n, int k){
        boolean optimo=((2*k)>n);
        if(optimo){
            for(int i=n; i>k; i--){
                String miI=(i+"");
                totaln=totaln.multiply(new BigInteger(miI));
            }
            for (int j=1; j<=n-k; j++){
                String miJ=(j+"");
                totalk=totalk.multiply(new BigInteger(miJ));
            }
        }
        else{
            for(int i=n; i>(n-k); i--){
                String miI=(i+"");
                totaln=totaln.multiply(new BigInteger(miI));
            }
            for (int j=1; j<=k; j++){
                String miJ=(j+"");
                totalk=totalk.multiply(new BigInteger(miJ));
            }
            resultado=totaln.divide(totalk);
        }
        return resultado.intValue();
    }
    public int getResultado(){
        return resultado.intValue();
    }
}
```

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;
import java.math.BigInteger;
public class coefbin{
    String cadena=""; // Variable utilizada para guardar el contenido del vector e imprimirlo
    CoefB cbb;
    public coefbin(String select, String f_entrada, String f_salida){
        boolean traza=(select.trim().equals("-t") || select.trim().equals("-T"));
        boolean help=(select.trim().equals("-h") || select.trim().equals("-H"));
        //Booleano Coeficiente C, para calcular con el segundo algoritmo creado.
        boolean CC=(select.trim().equals("-c") || select.trim().equals("-C"));
        cbb = new CoefB(0,0);
        if(!traza && !help && !CC){
            f_salida=f_entrada;
            f_entrada=select;
        }
        if(f_entrada.isEmpty())System.out.println("Seleccione archivo de entrada");
        //Solo se permitiran archivos con formato *.txt, en caso de que no se ponga expresament
        if(!f_entrada.endsWith(".txt"))f_entrada=f_entrada+".txt";
        if(!f_salida.isEmpty() && !f_salida.endsWith(".txt"))f_salida=f_salida+".txt";

        //(-h)OPCION HELP o AYUDA
        if (help){
            System.out.println("SINTAXIS:");
            System.out.println("servicio [-t] [-h] [fichero_entrada] [fichero_salida]");
            System.out.println("-t\t\t\t Traza la construcción del triángulo de Pascal");
            System.out.println("-h\t\t\t Muestra esta ayuda");
            System.out.println("fichero_entrada\t\t Nombre del fiechero de entrada");
            System.out.println("fichero_salida\t\t Nombre del fichero de salida");
        }
        //Caso con archivo de entrada
        else if(!f_entrada.isEmpty()){
            //Se utiliza 1 ArrayListde Coeficientes binomiales
            ArrayList<CoefB> listaB = new ArrayList<CoefB>();
            ArrayList<BigInteger> listaV= new ArrayList<BigInteger>();
            try {
                //Lectura del fichero de entrada
                BufferedReader lector = new BufferedReader(new FileReader(f_entrada));
                String linea = lector.readLine();
                while(linea != null){
                    String[] cadena = linea.split(" ");
                    cbb=new CoefB(Integer.parseInt(cadena[0]),Integer.parseInt(cadena[1]));
                    String numero = cbb.getResultado()+"";
                    if(CC){
                        BigInteger valor = new BigInteger(numero);
                        listaV.add(valor);
                    }
                    //Se rellena el ArrayList con los coeficientes existentes
                    listaB.add(cbb);
                    linea = lector.readLine();
                }
                lector.close();
```

```
                }
                lector.close();
            }
            catch (Exception e){System.out.println("Ha ocurrido un error no previsto");}
            //Se calcula el resultado del coeficiente binomial: n
            //Se rellena la cadena de salida utilizando el orden de llegada con coste: nlog(n)
            if(CC){
                for(BigInteger i: listaV){cadena+=(i.intValue()+"\n");}
            }
            else{
                for(CoefB cbb: listaB){
                    cadena+=(cbb.getResultado()+"\n");
                    //(-t)OPCION TRAZA solo esta disponible con la implementacion del triangulo
                    // y SOLO podra ser comprobada por pantalla
                    if (traza){
                        cbb.mostrarTraza();
                    }
                }
            }
            //Casos de salida de fichero
            try {
                //Sin archivo de salida, imprimimos por pantalla
                if(f_salida.isEmpty()){
                    System.out.println(cadena);
                }
                //Guardamos el contenido en el archivo de salida
                else if(f_salida!=null || !f_salida.isEmpty()) {
                    File nuevo=new File(f_salida);
                    String ruta=nuevo.getAbsolutePath();
                    File archivo=new File(ruta);
                    if(archivo.exists()){
                        System.out.println("Error, no se permite sobreescribir.");
                    }
                    else if(!archivo.exists()){
                        FileWriter escribir=new FileWriter(archivo,true);
                        escribir.write("\r\n");
                        escribir.write(cadena);
                        escribir.close();
                    }
                }
            }
            catch(Exception e){System.out.println("Ha ocurrido un error no previsto");}
        }
        else{System.out.println("Comando incorrecto");}
    }
    public static void main(String[] args) {
        //Main m = new Main(args[0],args[1],args[2]);
        new coefbin("-t","doc_e","");
    }
}
```

```java
import javax.swing.JTable;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;
public class Traza extends JFrame {
    public Traza(int[][] matriz,int n,int k) {
        final JTable tabla = new JTable(n+1, k+1);
        //Creamos un JscrollPane y le agregamos la JTable
        JScrollPane scrollPane = new JScrollPane(tabla);
        //Agregamos el JScrollPane al contenedor
        getContentPane().add(scrollPane, BorderLayout.CENTER);
        //manejamos la salida
        addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {System.exit(0);}
        });
        this.setExtendedState(MAXIMIZED_BOTH);
        tabla.setEnabled(false);
        setVisible(true);
           for (int i=0; i<=n; i++) {
                matriz[i][0]=1;
                   try {   Thread.sleep (100);
                           } catch (Exception e) {
                           System.out.println("Error indeterminado");
                           }
                tabla.setValueAt(matriz[i][0],i,0);
           }
           for (int i=1; i<=n; i++) {
               matriz[i][1]=i;
                   try {   Thread.sleep (100);
                           } catch (Exception e) {
                           System.out.println("Error indeterminado");
                           }
               tabla.setValueAt(matriz[i][1],i,1);
           }
           for (int i=2; i<=k; i++) {
               matriz[i][i]=1;
                   try {   Thread.sleep (100);
                           } catch (Exception e) {
                           System.out.println("Error indeterminado");
                           }
               tabla.setValueAt(matriz[i][i],i,i);
           }
           for (int i=3; i<=n; i++){
               for(int j=2; j<n; j++){
                   if(j<=k && j<i){
                       try {Thread.sleep (100);
                           } catch (Exception e) {
                           System.out.println("Error indeterminado");
                           }
                       tabla.setValueAt(matriz[i][j],i,j);
                   }
               }
           }
       }
}
```