Teoría de la Recursividad
Capítulo 4: Presentación de un programa recursivo

El ejercicio 1.7 del capítulo 1, es un caso particular de un problema más general que podría enunciarse así:

Dados 2N números cualesquiera (N > 0), distribuirlos en dos grupos de N elementos cada uno, de modo que la diferencia entre las sumas de ambos grupos sea mínima.

La solución del ejercicio 1.7, se basa en probar con todas las combinaciones posibles de cuatro números de la lista, para encontrar una que minimice la diferencia de sumas.

Esto se logra mediante cuatro ciclos anidados. Esquemáticamente:

```
for(int i=0; i<5; i++)
    for(int j=i+1; j<6; j++)
        for(int k=j+1; k<7; k++)
            for(int l=k+1; l<8; l++)
            {
                // trabajar con la combinación de
                // elementos a[i],a[j],a[k],a[l]
            }
```

Observe que la forma de definir los ciclos garantiza que no se repitan combinaciones.

Se obtiene así todas las variantes posibles para el primer grupo. El otro, quedará constituido, en cada caso, por los restantes elementos.

Pero, como ya se ha dicho, la solución del problema general no puede plantearse en forma similar, pues solo conoceremos la cantidad de números de la lista (y de ahí, la cantidad de ciclos "`for`" necesarios), en el momento de la ejecución.

A continuación analizaremos una solución recursiva al problema general.

El efecto de anidamiento de ciclos para obtener todas las combinaciones posibles para el grupo 1 se logra mediante un procedimiento recursivo:

```
public static void Selecciona(int niv,int posic,bool[] b,bool[] sol) // {a}
{
     if (niv<=n)                                        // {b}
           for(int i=posic; i<n+niv; i++)               // {c}
           {
                 b[i]=true;
                 Selecciona(niv+1,i+1,b,sol);           // {d}
                 // {  -------  }
                 b[i]=false;
           }
     else
     {                                                  // {e}
           int s_act=0;
           for(int i=0; i<a.Length; i++)
                 if (b[i]) s_act+=a[i];
           if (Math.Abs(semisuma-s_act)<min)
           {
                 min=Math.Abs(semisuma-s_act);
                 s1=s_act;
                 s2=s-s1;
                 for(int i=0; i<b.Length; i++)          // {1}
                       sol[i]=b[i];
           }
     }
}
```

Análisis del procedimiento:

{a} Parámetros.

En nuestra solución, cada “nivel” de ejecución del procedimiento se encargará de agregar un nuevo elemento al primer grupo. Es usual, y generalmente muy cómodo, transmitir como parámetro el “nivel” de ejecución (variable `niv`).
Es muy recomendable limitar los restantes parámetros al mínimo imprescindible. En este caso, no obstante, necesitamos otro parámetro (`posic`) y dos arreglos `bool` `b` y `sol`, según se verá enseguida.

{b} Subdivisión del procedimiento.

Es típico de un procedimiento recursivo como el que analizamos, el constar de dos partes: Una, que se encarga de la selección de la combinación, lo que se realiza mediante sucesivos llamados recursivos; y otra, que corresponde a las acciones necesarias una vez que se ha completado la selección actual.

En este caso, la opción de una u otra alternativa se hace mediante la cláusula:

```
if (niv<=n) { ... }  // (construir nueva selección)
else { ... }         // (operar con nueva selección)
```

donde n es la cantidad de elementos de cada grupo a formar.

{c} Selección.

```
    for(int i=posic; i<n+niv; i++)        // {c}
    {
        b[i]=true;
        Selecciona(niv+1,i+1,b,sol);      // {d}
        // {  -------  }
        b[i]=false;
    }
```

Es un proceso muy sencillo que consiste en “marcar” la posición adecuada (dada por el ciclo `for` ...) y pasar al nivel siguiente en forma recursiva.
Para marcar que un elemento ha sido incluido en la actual selección, se utiliza un arreglo `bool` en el que los valores `true` indican que la posición corresponde a un elemento del primer grupo.
Como cada nuevo llamado recursivo (`{d}`) es equivalente a “anidar” un nuevo ciclo, se hace necesario que cada nuevo ciclo comience una posición más a la derecha que la del elemento que se acaba de seleccionar. (Ej. en la solución para 2N=8: `for (int j=i+1; j<6; j++) {...etc...}`)
Por ello hace falta transmitir como parámetro la posición inicial para el nuevo ciclo (`posic`).
El valor final del ciclo, esta dado por la expresión n+niv, que generaliza los valores finales necesarios en cada caso.

Como comprobación, verifiquemos para el caso de 8 elementos (n = 4) los valores que se obtiene con esta formula:

niv	n+niv	valor final ciclo
1	4+1	5
2	4+2	6
3	4+3	7
4	4+4	8

Como puede apreciarse, obtenemos exactamente los mismos valores que utilizamos en la solución no recursiva (ejercicio 1.7).

{d} Como “funciona” la recursividad.

Para concretar ideas, imaginemos el proceso funcionando para una lista de 6 elementos (n = 3).

Es necesario que el procedimiento sea “activado” desde el programa principal con los parámetros (1,0), es decir niv = 1 y posic = 0.

Como `niv<n`, se ejecuta la parte “then”, y el ciclo comienza en el valor i = 0. Se “marca” `b[0]=true`, y se llama nuevamente al procedimiento con los argumentos (2,1). La ejecución en el nivel 1 se interrumpe, y las correspondientes variables LOCALES niv e i (también posic, aunque esta ya no interesa pues no necesitaremos ese valor al retornar a este nivel), permanecen en memoria con sus valores respectivos, pero “en espera”, momentáneamente inactivas, hasta que se retorne a este nivel.

Por su parte, se crean nuevamente las variables, ahora correspondientes al nivel 2, en el que, tal como se hizo el llamado, niv = 2 y posic = 1.

Nuevamente es `niv<n`, y se ejecuta otra vez la parte “then”. El ciclo comienza ahora con i = 1, se “marca” `b[1]=true` y se pasa al tercer nivel. Las variables i, niv (del nivel 2) quedan también momentáneamente inactivas, y se crean las variables niv = 3, posic = 2 e i = 2 del tercer nivel.

Resulta ahora `niv==n`, pero igualmente se ejecuta la parte “then”. Por tanto, se marca `b[2]=true` y se pasa al nivel 4.

Ahora, resulta `niv>n`, por lo que se ejecuta la parte `else` con la primera selección realizada (números de las posiciones 0, 1 y 2). Esta parte la veremos después. Por ahora digamos que la selección de lugares 0, 1, 2 se almacena en otro arreglo como la mejor hasta el momento, y el procedimiento retorna.

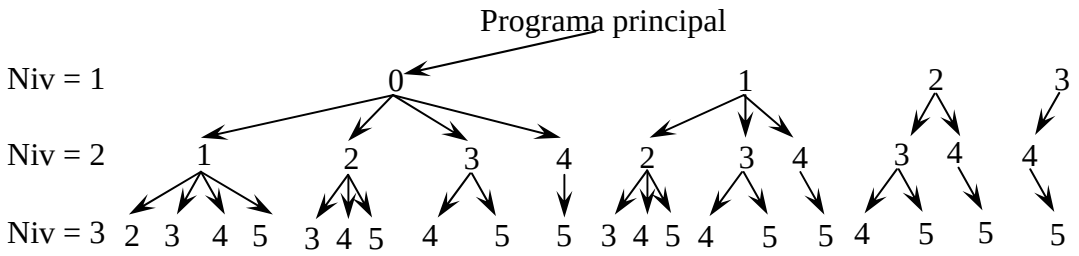
El retorno, como sabemos, siempre se produce al módulo que realizó el llamado, a la instrucción siguiente al mismo. Por tanto, la ejecución vuelve al nivel 3, parte indicada con `// {-------}` donde se reactivan las variables de este nivel, mientras se destruyen las del nivel 4. Se “borra” la marca realizada (`b[2]=false`) y continúa la ejecución del ciclo (el mismo que se comenzó la primera vez que se “llamó” al nivel 3), donde ahora i toma valor 3 y por tanto se marca `b[3]=true`, pasando nuevamente al nivel 4, ahora con la selección de posiciones 0, 1 y 3.

Por un proceso similar al descrito, luego se obtiene las selecciones 0, 1, 4 y 0, 1, 5. Pero al retornar al nivel 3 después de procesar esta última, con el borrado de la marca de `b[5]`, concluye el ciclo de este nivel y se llega al fin de la ejecución del procedimiento (en el nivel 3). Por ello, se produce otro retorno, ahora al nivel 2.

Se desmarca `b[1]` y, prosiguiendo el ciclo de este nivel, se marca `b[2]`. En forma similar a lo detallado, a partir de ahora se generarán las combinaciones 0, 2, 3; 0, 2, 4 y 0, 2, 5. Y así sucesivamente, hasta terminar el proceso después de obtener la ultima combinación 3, 4, 5 y, por retornos sucesivos, volver al programa principal.

En el proceso se utiliza continuamente el recurso de marcar y luego desmarcar el elemento seleccionado en el nivel. Evita muchas dificultades y garantiza un correcto funcionamiento del programa, realizar la marca inmediatamente antes de pasar al siguiente nivel, y desmarcar inmediatamente después de retornar al mismo.

Representación esquemática del proceso. (Para 2N=6)



El “árbol” de combinaciones para 6 elementos

{e} Acciones al completar una selección.

```
else
{                                                   // {e}
      int s_act=0;
      for(int i=0; i<a.Length; i++)
            if (b[i]) s_act+=a[i];
      if (Math.Abs(semisuma-s_act)<min)
      {
            min=Math.Abs(semisuma-s_act);
            s1=s_act;
            s2=s-s1;
            for(int i=0; i<b.Length; i++)     // {1}
                  sol[i]=b[i];
      }
}
```

Aquí, la descripción será mucho más breve:

En esta parte, se utiliza las variables:

Nombre de variable	Descripción	Tipo (local o global)
semisuma	Mitad de la suma de todos los elementos.	Local
dif	Menor diferencia hasta ahora entre una suma actual y la semisuma.	Local
s	Suma total de todos los elementos de la lista.	Local
s1, s2	Suma de los nuevos grupos formados (si con la actual selección se logró disminuir la diferencia anterior).	Local
s_act	Suma de elementos de la actual combinación	Global

El proceso consiste en obtener la nueva suma actual, y si resulta mejor que las obtenidas anteriormente (más próxima a “semisuma”), almacenar las sumas de cada nuevo grupo y la combinación actual. Para esto, se transfiere a otro arreglo `bool sol` el arreglo `b` tal como está en ese momento.

Observe (`{1}`) que para ello se emplea un ciclo `for` que recorre ambos arreglos paralelamente asignando `sol[i]=b[i]`.

Acciones en el programa principal (o en un método público previo).

Para no hacerlo excesivamente largo, se ha programado la lectura de datos y la impresión de resultados en métodos aparte:

```
1. public static int[] LeerLista(int cantidad)
2. public static void Imprimir()
```

Entonces, el programa principal se limita a calcular la suma y luego la semisuma de los elementos de la lista; poner inicialmente en `false` todos los elementos del arreglo de marcas auxiliar y el de solución; inicializar `dif` con un valor positivo suficientemente grande; y activar el procedimiento recursivo mediante el llamado:

```
Selecciona(1,0,aux,sol); // en este caso le llamamos
                         // aux al auxiliar y sol al de solución
```

Veamos el programa completo:

```
using System;
namespace Sumas_Similares
{
      class ClasePrincipal
      {
            static int[] a;
            static bool[] aux;
            static bool[] sol;
            static int s1=0;
            static int s2=0;
            static int s=0;
            static double semisuma=0;
            static double min= 0;
            static int n=0;

            public static int[] LeerLista(int cantidad)
            {
                  int[] a=new int[cantidad];
                  Console.WriteLine("Deme elementos de la lista.");
                  for(int i=1; i<=cantidad; i++)
                  {
                        Console.Write(i+" - ");
                        a[i-1]=int.Parse(Console.ReadLine());
                  }
                  return a;
            }
```

```
public static void Selecciona(int niv,int posic,bool[] b,bool[] sol)
{
      if (niv<=n)
            for(int i=posic; i<n+niv; i++)
            {
                  b[i]=true;
                  Selecciona(niv+1,i+1,b,sol);
                  // {  -------  }
                  b[i]=false;
            }
      else
      {
            int s_act=0;
            for(int i=0; i<a.Length; i++)
                  if (b[i]) s_act+=a[i];
            if (Math.Abs(semisuma-s_act)<min)
            {
                  min=Math.Abs(semisuma-s_act);
                  s1=s_act;
                  s2=s-s1;
                  for(int i=0; i<b.Length; i++)
                        sol[i]=b[i];
            }
      }
}

public static void Imprimir()
{
      int[] grupo1= new int[a.Length/2];
      int[] grupo2= new int[a.Length/2];
      Console.WriteLine("Los dos grupos son:");
      int j=0;
      int k=0;
      for(int i=0; i<sol.Length; i++)
            if (sol[i])
            {
                  grupo1[j]=a[i];
                  j++;
            }
            else
            {
                  grupo2[k]=a[i];
                  k++;
            }

      Console.Write(grupo1[0]);
      for(int i=1; i<grupo1.Length; i++)
          Console.Write(" + "+grupo1[i]);
      Console.WriteLine(" = "+s1);

      Console.Write(grupo2[0]);
      for(int i=1; i<grupo2.Length; i++)
          Console.Write(" + "+grupo2[i]);
      Console.WriteLine(" = "+s2);

      Console.WriteLine("Diferencia: " + Math.Abs(s2-s1));
}
```

```
            static void Main(string[] arg)
            {
                  int m;
                  do
                  {
                        Console.Write("Cantidad de elementos (debe ser PAR): ");
                        m=int.Parse(Console.ReadLine());
                        n=m/2;
                  }while (m%2!=0);

                  a=LeerLista(n*2);
                  aux=new bool[n*2];
                  sol=new bool[n*2];
                  for(int i=0; i<a.Length; i++)
                        s+=a[i];
                  min=Math.Abs(s);  // <- valor inicial suficientemente "grande"
                  semisuma=s/2;
                  Selecciona(1,0,aux,sol);   // <- desencadena la recursividad
                  Imprimir();
                  Console.ReadLine();
            }
      }
}
```

Sugerencia al lector:
Antes de pasar al cap¡tulo siguiente, aseg£rese de haber comprendido
bien el funcionamiento del programa que se acaba de explicar.