# Diseño de algoritmos

*Jesús Bermúdez de Andrés. UPV-EHU*
*Guías para la solución de ejercicios: Vuelta atrás*

*Curso 2008-09*

1. Escriba un algoritmo de *vuelta atrás* para determinar el mínimo número de monedas necesarias para sumar una cantidad exacta $L$, cuando se dispone de $n$ clases diferentes de monedas, con valores $v_1 \dots v_n$ respectivamente, y es posible tomar tantas monedas de cada clase como se desee.

   **Solución:**
   Supongamos que $v_1 \leq \ldots \leq v_n$.

   Decidimos que un ensayo quede representado por una secuencia SEC=$[s_1, \dots s_{long}]$, de valores de monedas tal que $s_i \leq s_{i+1}$ y $\sum_{i=1}^{long} s_i \leq L$. Sea *long* y *suma* la longitud y suma de valores de SEC, respectivamente. SEC puede estar implementada con un array.

   Por tanto, si $s_{long} = v_k$, $k$ es la clase de moneda con la que comenzaríamos a extender el ensayo SEC.

   Denominamos OPT y longOPT al mejor ensayo calculado hasta el momento y su longitud, respectivamente.

```
proc ENSAYAR (L, SEC, k, long, suma, OPT, longOPT)
  if suma = L then
    if long < longOPT then
      OPT ← SEC
      longOPT ← long
    end if
  else
    while long + 1 < longOPT ∧ suma + v_k ≤ L ∧ k ≤ n loop
      SEC(1 + long) ← v_k
      ENSAYAR (L, SEC, k, 1 + long, suma + v_k, OPT, longOPT)
      k ← k + 1
    end loop
```

   La llamada inicial pertinente sería ENSAYAR (L, [ ], 1, 0, 0, OPT, longOPT)

2. Escriba un algoritmo de *vuelta atrás* que, dados dos números enteros positivos $X$ y $r$, calcule todas las formas posibles de descomponer $X$ como suma de $r$ cuadrados. Es decir, calcule todos los conjuntos de $r$ números enteros positivos $\{x_1, \dots, x_r\}$ tales que $x_1^2 + \dots + x_r^2 = X$.

**Solución:**

Un ensayo será una secuencia de valores $[x_1, \ldots, x_k]$ de manera que $x_1 \leq \ldots \leq x_k$. Obsérvese que, debido a la conmutatividad de la suma, si estamos estudiando $1^2 + 1^2 + 2^2$ y previamente hemos estudiado $1^2 + 1^2 + 1^2 + 2^2$, no interesa repetir el trabajo con $1^2 + 1^2 + 2^2 + 1^2$.

En el algoritmo siguiente, SEC representa el ensayo actual, de longitud $K <= r$, S es la suma de los cuadrados de los números de SEC, y $j$ es el primer número entero con el que extenderíamos SEC. SEC está implementada con un array.

**proc** ENSAYAR ($X$, $r$, SEC, S, $K$, $j$)
  **if** S $= X \wedge K = r$ **then**
    'Añadir SEC a la respuesta'
  **else** $\{\text{S} < X \vee K < r\ \}$
    **while** $j^2$+ S $\leq X(*) \wedge K < r$ **loop**
      SEC$(K+1) \leftarrow j$
      ENSAYAR ($X$, $r$, SEC, S+$j^2$, $K+1$, $j$)
      $j \leftarrow j + 1$
    **end loop**

(*) Otra expresión booleana, para restringir más el abanico de extensiones, podría ser la siguiente. Obsérvese que el máximo $j$ sería un $i$ tal que $(r-K)i^2 = X-$ S (al principio, cuando $K = 0$ y S= 0, sería $i = \sqrt{\frac{X}{r}}$). Entonces una condición de terminación del blucle, más restrictiva, sería $j^2(r-K) \leq X-$ S. La llamada inicial sería ENSAYAR ($X$, $r$, [ ], 0, 0, 1).

3. Una agencia matrimonial dispone de las tablas de información $S(1 \ldots n, 1 \ldots n)$ y $C(1 \ldots n, 1 \ldots n)$ sobre las preferencias de $n$ señoras y $n$ caballeros, respectivamente. Para cada señora $i$, $S(i, j)$ es un número que representa el orden de preferencia de la señora $i$ por el caballero $j$. Análogamente, para cada caballero $x$, $C(x, y)$ es un número que representa el orden de preferencia del caballero $x$ por la señora $y$.

   Escriba un algoritmo de *vuelta atrás* que indique a la agencia matrimonial una forma de emparejar biyectivamente a las señoras con los caballeros de manera que se minimice la suma del producto de las respectivas preferencias de cada pareja; es decir,

$$\text{minimizar} \sum_{(a,b) \in \text{Parejas}} S(a,b) \times C(b,a)$$

   **Solución:**

Una posibilidad es considerar como *ensayo* una secuencia de números de $1 \ldots n$, de manera que SEC=$[3, 1, 5]$ representa que la señora número 1 está emparejada con el caballero número 3 y las señoras 2 y 3 emparejadas, respectivamente, con 1 y 5. Un ensayo puede extenderse con cualquier número de $1 \ldots n$ que no esté en la secuencia SEC.

Las posibilidades de extender un ensayo de longitud $i-1$ consisten en emparejar a la señora número $i$ con cada uno de los caballeros que todavía no han sido emparejados.

La condición de poda más simple consiste en no continuar con un ensayo SEC de longitud $p$, cuya última extensión se ha realizado con el valor $k$, si resulta que $Beneficio$(SEC') + $S(p,k)C(k,p)$ "no tiene esperanza de mejorar el óptimo temporal calculado hasta ahora" (SEC' representa el ensayo SEC antes de añadirle el elemento $k$ en la posición número $p$).

Además de la secuencia SEC, podemos representar la presencia de un número en SEC con un array, denominado hombres, de $n$ valores booleanos. De este modo, comprobar si un número está en SEC se realiza en tiempo constante.

Denominamos OPT y sumaOPT al mejor ensayo calculado hasta el momento y su suma, respectivamente.

```
proc Ensayar (S, C, hombres, i, SEC, suma, OPT, sumaOPT)
{i representa la señora en curso}
  if i > n then
    if suma < sumaOPT then
      OPT(1..n) ← SEC(1..n)
      sumaOPT ← suma
    end if
  else
    for j in 1...n loop
      if ¬hombres(j) then
        hombres(j) ← true
        if suma + (S(i,j) × C(j,i)) < sumaOPT then
          SEC(i) ← j
          Ensayar (S, C, hombres, i + 1, SEC, suma+(S(i,j) × C(j,i)),
                   OPT, sumaOPT)
        end if
        hombres(j) ← false
      end if
    end loop
```

La llamada inicial sería: Ensayar ($S$, $C$, hombres, 1, [ ], 0, OPT, $\infty$).

4. Tenemos $n$ programas para grabar en un disco, pero el espacio de memoria que necesitan excede la capacidad del disco. Cada programa $P_i$ requiere $m_i$ kilobytes de memoria, la capacidad del disco es de $C$ kilobytes y $C < \sum_{i=1}^{n} m_i$.

Diseñe un algoritmo, utilizando la técnica de *vuelta atrás*, que calcule una colección de esos programas para grabar en el disco de manera que se minimice el espacio que queda libre en el disco después de la grabación.

**Solución:**
Una posibilidad es la siguiente: Un ensayo es una secuencia de números tomados de $1 \dots n$ que representan a los programas correspondientes que grabaríamos en el disco. Por ejemplo $[2, 5, 7]$ representa la posibilidad de grabar los programas número $2, 5$ y $7$. Para no estudiar ensayos repetidos decidimos que un ensayo sólo puede extenderse con un número mayor que el último de la secuencia. Así, el ensayo ejemplo puede extenderse con $[2, 5, 7][8]$, $[2, 5, 7][9]$ y aumentando el último número hasta $[2, 5, 7][n]$. Un ensayo es aceptable si la suma de sus kilobytes de memoria no supera la capacidad del disco $C$. Con este diseño, todo ensayo aceptable es una solución. Además, tendremos soluciones que son prefijo de otras soluciones (habrá que optar por el esquema de *vuelta atrás* que tiene en cuenta esa posibilidad). Según vamos encontrando soluciones nos vamos quedando con la que maximiza la cantidad de memoria ocupada.

Podemos añadir una condición más para restringir el número de posibilidades de extensión de un ensayo. Si ordenamos previamente los programas según su ocupación de memoria (de menor a mayor, $m_k \leq m_{k+1}$) entonces cuando una extensión $[2, 5, 7][k]$ no es aceptable, tampoco lo serán las siguientes y por tanto podemos terminar el bucle de extensiones de ensayos.

Otra posibilidad es considerar un ensayo como una secuencia de unos y ceros que representan la inclusión o no, en la solución, del programa que ocupa esa posición en la numeración. Por ejemplo, $[0, 1, 0, 0, 1, 0, 1]$ representa al ejemplo de antes, que incluye los programas número $2, 5$ y $7$. Las posibilidades de extensión son dos 1 y 0. Un ensayo es aceptable si la suma de sus correspondientes kilobytes de memoria no supera la capacidad del disco $C$. Un ensayo es solución cuando tiene longitud $n$. Según vamos encontrando soluciones nos vamos quedando con la que maximiza la cantidad de memoria ocupada.

También aquí, si ordenamos los programas según necesidad creciente de memoria ($m_k \leq m_{k+1}$), podemos podar algunas ramas siguiendo el mismo criterio que vimos en la solución del problema de la suma de subconjuntos.

5. Diseñe un algoritmo de *vuelta atrás* que, dado un número entero positivo $C$, calcule la lista de números enteros positivos dispuestos en orden creciente (y por lo tanto, distintos) que sumen $C$ con la mayor cantidad de impares posible. Por ejemplo, si $C = 6$ entonces la salida puede ser: $[1, 2, 3]$ o bien $[1, 5]$ ya que ambas tienen 2 impares.

   **Solución:**
   A continuación se presenta una solución inmediata, sin aplicación de posibles optimizaciones.

   Un ensayo será una secuencia de números enteros positivos en orden creciente. Las posibilidades de extender un ensayo que termina con el número $k$ consisten

en añadir a la secuencia un número. Para garantizar el orden creciente, la primera extensión se realiza con el número $k+1$ y posteriormente con sus siguientes hasta llegar al máximo que sea aceptable. Es decir, hasta llegar a un número que sumado a los demás del ensayo supere la cantidad $C$.

El parámetro S representa el ensayo, el parámetro Impares representa la cantidad de números impares que hay en el ensayo S, y el parámetro Suma representa la suma de los números que hay en S. Análogamente, SOPT e ImparesOPT representan la mejor solución encontrada hasta el momento y el número de impares que contiene. El parámetro $k$ representa el último número de la secuencia S.

Describimos el algoritmo suponiendo que representamos S con una estructura de lista, sobre la que usamos las operaciones AÑADIR_AL_FINAL(S, $x$) (que añade el elemento $x$ al final de la lista S) y RETIRAR_ULTIMO(S) (que retira el último de la lista S).

```
proc ENSAYAR (C, S, Impares, Suma, k, SOPT, ImparesOPT)
  if Suma = C then
    if Impares > ImparesOPT then
      SOPT ← S
      ImparesOPT ← Impares
    end if
  else
    e ← k + 1
    while Suma + e ≤ C loop
      AÑADIR_AL_FINAL(S, e)
      Suma ← Suma + e
      if e mod 2 = 1 then Impares ← Impares + 1 end if
      ENSAYAR(C, S, Impares, Suma, e, SOPT, ImparesOPT)
      RETIRAR_ULTIMO(S)
      Suma ← Suma - e
      if e mod 2 = 1 then Impares ← Impares − 1 end if
      e ← e + 1
    end loop
```

La llamada inicial será ENSAYAR($C$, [ ], 0, 0, 0, [ ], 0).

6. Considere un tablero escaqueado (como el del ajedrez) de tamaño $n \times n$, con un caballo (pieza del juego de ajedrez) situado en un escaque arbitrario de coordenadas $(x, y)$. El problema consiste en determinar $n^2 - 1$ movimientos del caballo tal que cada escaque del tablero se visite exactamente una vez, si tal secuencia de movimientos existe.

   Para abreviar notación y cálculo de coordenadas puede disponer de la función que comentamos a continuación: El número entero $m$ representa una de las ocho alternativas posibles de movimiento de un caballo. Comenzando por el norte, y en el sentido de las agujas del reloj, las numeramos $1 \ldots 8$. Partiendo

de las coordenadas $(i, j)$ y realizando la alternativa $m$, las coordenadas de la posición siguiente vienen dadas por la expresión $f(i, j, m)$.

**Solución:**
Un ensayo puede representarse mediante una pila de las posiciones alcanzadas hasta el momento. Inicialmente *ensayo*=$[(x, y)]$, que son las coordenadas de partida. Cuando la longitud de *ensayo* sea $n^2$ tendremos una solución. No son posiciones aceptables para incluir en *ensayo* aquellas cuyas coordenadas queden fuera del tablero o bien sean coordenadas presentes en *ensayo*. Gestionaremos una matriz de marcas $V(1..n, 1..n)$ que registre las coordenadas visitadas: Inicialmente $V(x, y) = true$ y $\forall (i, j) \neq (x, y)$. $V(i, j) = false$.

```
proc CABALLO (ensayo, éxito, V(1..n, 1..n))
  if ensayo.LONG() = n^2 then
    éxito ← true
    IMPRIMIR(ensayo)
  else
    for m ∈ 1...8 loop
      i ← ensayo.CIMA().pri {primera coordenada de la cima}
      j ← ensayo.CIMA().seg {segunda coordenada de la cima}
      if DENTRO_DE_TABLERO(f(i, j, m)) ∧ V(i, j) = false then
        V(i, j) ← true
        ensayo.PUSH(i, j)
        CABALLO (ensayo, éxito, V(1..n, 1..n))
        if éxito then
          return
        else
          (i, j) ← ensayo.POP()
          V(i, j) ← false
```

La llamada inicial será CABALLO ($[(x, y)]$, $false$, $V$), con la inicialización comentada de $V$.