# Examen de Teoría de la Programación

E. U. ING. TEC. EN INFORMÁTICA DE OVIEDO

Final Febrero – Curso 2007-2008 11 de febrero de 2008

DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

**3. (1,5 puntos) Dado un conjunto de enteros positivos *a* y un número entero positivo *s*. El problema consiste en diseñar un algoritmo para encontrar todos los posibles subconjuntos de *a* que sumen *s*.**

**a) (1 punto) El problema lo hemos resuelto utilizando la técnica de Backtracking. Completar el código Java para dar solución a este problema (escribirlo en los recuadros preparados a tal efecto).**

```
  private int a[];              //vector de números
  private int s;                //suma dada
  private int solucion[];       //array del estado del problema
  private int sParcial = 0;     //suma del estado construido
  private int n;                //tamaño del problema

[…]  //Suponer un constructor que inicialice todo lo necesario

  void ensayar(_____ int posi _____) {
    for (___int i = 0; i <= 1; i++___) {
➔     if (sParcial+i*a[posi] <= s) {
        solucion[posi] = i;
        sParcial += i * a[posi];

        if (posi < n - 1) {
          ensayar(____posi + 1____);
        }
        else {
          if (____sParcial == s _____) {
            tratarSolucion();
          }
        }
        sParcial -= i * a[posi];
➔     } //fin if
    } //cierra el for

  }
```

**b) (0,5 puntos) Queremos optimizar el árbol de estados desarrollados, de tal forma que cuando la suma de los elementos contemplados supere la suma dada, realice una poda de todos sus hijos. Escribir el código Java necesario para realizar esto (usar el espacio de abajo), e indicar en el código del apartado a) la línea o líneas donde se insertaría este código.**

```
____ if (sParcial+i*a[posi] <= s) { _____________________
_____________________________________________________
____} //fin if ____________________________________________
```

Nota: A esta segunda parte hay múltiples alternativas válidas.

**4. (1,25 puntos) Dado un sistema monetario compuesto por monedas de distinto valor, el problema del cambio consiste en descomponer en monedas de curso legal cualquier cantidad dada *j*, utilizando el menor número posible de monedas de dicho sistema monetario.**

**Siendo *n* el número de tipos de monedas distintos, *j* la cantidad que queremos descomponer y *T(1..n)* un vector con el valor de cada tipo de moneda del sistema. Suponemos que tenemos una cantidad inagotable de monedas de cada tipo. Se debe calcular la cantidad mínima final de monedas de cada tipo para dicha cantidad *j* inicial.**

**La función *C(i,j)* es el número mínimo de monedas para obtener la cantidad *j* restringiéndose a los tipos *T(1), T(2), … T(i)*. La función que nos proporciona la solución al problema es la siguiente:**

$$C(i,j)=\begin{cases}\infty \text{ si } (i=0) \text{ ó } (j<0)\\ 0 \text{ si } j=0 \text{ y } i>0\\ Min(C(i\text{-}1,j), C(i,j\text{-}T_i)+1) \text{ en otro caso}\end{cases}$$

**Utilizaremos la técnica de Programación Dinámica para obtener la solución óptima al problema.**

**a) Representar la tabla necesaria para almacenar los valores intermedios (sin rellenar). Sabiendo que tenemos que devolver 7 céntimos con las cinco primeros tipos de monedas del sistema euro.**

- Para devolver 7 céntimos con los 5 primeros tipos de monedas. Cantidad → j, Tipos monedas → n.

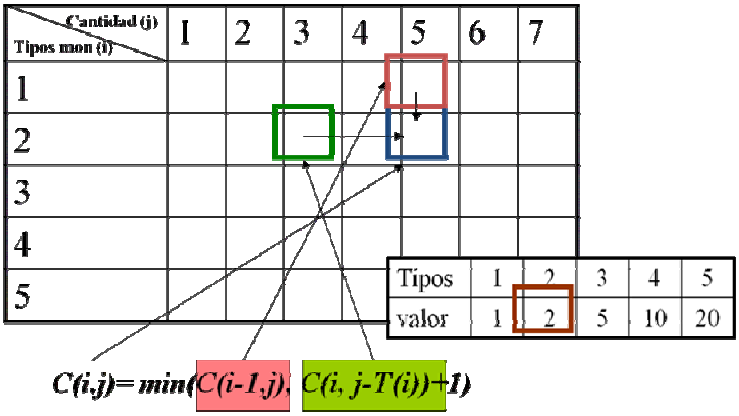
| Cantidad (j) / Tipos mon (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |
| 5 | | | | | | | |

C[i,j]

**b) Marcar en la tabla anterior que valores podemos rellenar de forma directa en la tabla.**

Hemos eliminado de la tabla los valores que se podían rellenar de forma directa, por tanto todas las celdas deben de ser calculadas.

**c) Buscar un patrón de dependencia de una celda cualquiera, es decir, marcar las celdas que son necesarias para calcular una celda dada.**




**d) ¿Qué complejidad tiene la implementación de esta solución? Razona la respuesta.**

Dos bucles anidados que van recorriendo cada una de las celdas del array.
O(i·j)→ O(n2)

**5. (1 punto) El problema de la mochila consiste en que disponemos de *n* objetos y una "mochila" para transportarlos. Cada objeto *i*= 1,2, ... *n* tiene un peso $w_i$ y un valor $v_i$. La mochila puede llevar un peso que no sobrepase *W*. En el caso de que un objeto no se pueda meter entero, se fraccionará, quedando la mochila totalmente llena. El objetivo del problema es maximizar valor de los objetos respetando la limitación de peso. Queremos resolver este problema mediante un algoritmo voraz.**

**a) Demostrar mediante un contraejemplo que elegir los objetos en orden decreciente de valor ($v_i$) no es necesariamente óptima.**

**b) Demostrar mediante un contraejemplo que elegir los objetos en orden creciente de peso ($w_i$) no es necesariamente óptima.**

n=5, W=100

| Objeto | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| w | 10 | 20 | 30 | 40 | 50 |
| v | 20 | 30 | 66 | 40 | 60 |

| Objeto | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| valor / peso | 2 | 1,5 | 2,2 | 1 | 1,2 |

| | 1 | 2 | 3 | 4 | 5 | Valor de la mochila |
|---|---|---|---|---|---|---|
| Orden decreciente de valor | 0 | 0 | 1 | 0,5 | 1 | 146 |
| Orden creciente de peso | 1 | 1 | 1 | 1 | 0 | 156 |
| Orden decreciente cociente valor / peso | 1 | 1 | 1 | 0 | 0,8 | 164 |

Vemos que para la misma mochila cogiendo los elementos en otro orden (3er caso) obtenemos un mejor valor en la mochila con lo que queda demostrado que los dos primeros heurísticos propuestos NO siempre proporcionan la solución óptima.

**c) Plantear un heurístico que proporcione la solución óptima en cualquier caso.**

Calcular el cociente de $v_i$ / $w_i$ de cada objeto y ordenarlos de mayor a menor (orden decreciente). Vamos metiendo en la mochila los objetos en este orden hasta que haya uno que supere la capacidad de la mochila, entonces este lo fraccionamos de tal forma que llene completamente la capacidad de la mochila.

**6. (2 puntos) El cuadriatlón es una variante de triatlón en la que se combinan Natación(N)-Piragua(P)-Ciclismo(C)-Atletismo(A). Ante la proximidad de los juegos olímpicos de Pekín 2008 (aunque esta no es una prueba olímpica) se ha propuesto, a modo de entrenamiento para los deportistas, hacer una competición. Se ha decidido que la prueba sea por relevos y cada participante realizará un deporte y le dará el relevo a su compañero. El seleccionador dispone de cuatro deportistas y de sus tiempos para cada una de las pruebas. Se quiere diseñar el algoritmo que mediante la técnica de ramificación y poda permita decidir al seleccionador que deportista realizará cada prueba.**
**Deportista 1 (18-39-24-15). Deportista 2 (23-28-25-18). Deportista 3 (22-29-26-17). Deportista 4 (15-30-25-20). Los tiempos están en el mismo orden en el que se realizan los deportes.**

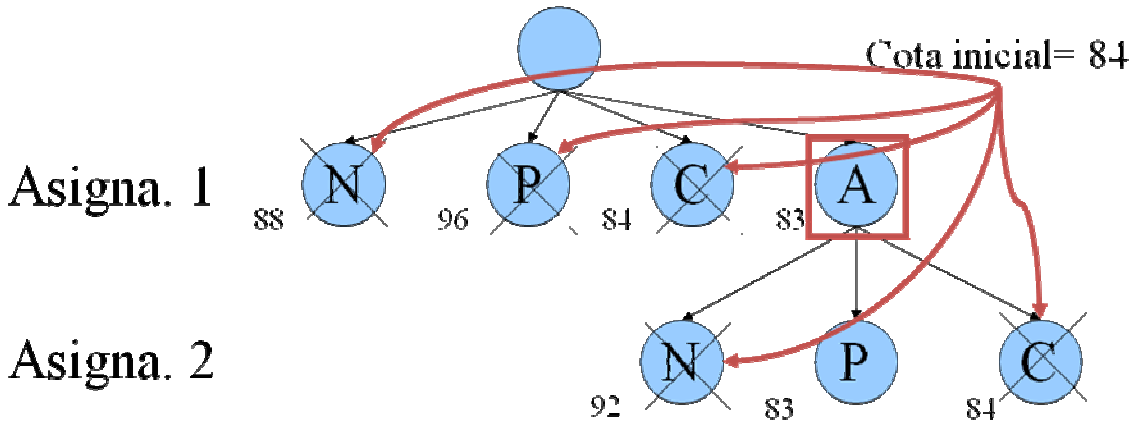
**a) (0,25 puntos) Explicar cómo se calcula el heurístico de ramificación. Y aplicarlo al estado en el que asignamos al deportista 1 a ciclismo y al deportista 2 a natación (y quedan dos deportistas por asignar).**

El cálculo del heurístico de ramificación consiste en: La suma de lo ya asignado en cada estado más el caso mejor (mínimo de columnas) de lo que queda por asignar.
Al estado 1→C, 2→N, le corresponde un coste de 24+23+29+17=93.

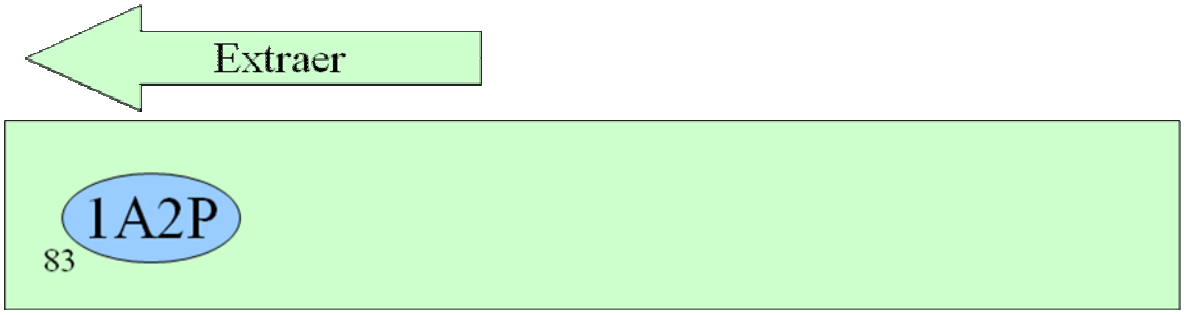
**b) (0,25 puntos) Explicar cómo se calcula la cota inicial de poda y razonar cuándo se produce el cambio de esta cota.**

Inicialmente, es la menor de la sumas de las dos diagonales de la matriz de costes.
Valor cota inicial: min(92,84)=84.
Cuando en el desarrollo de los estados del árbol, lleguemos a una solución válida cuyo valor sea menor que el de la cota actual, se cambiará la cota a este nuevo valor.

**c) (1 punto) Representar el árbol de estados después de haber expandido dos estados del árbol.**



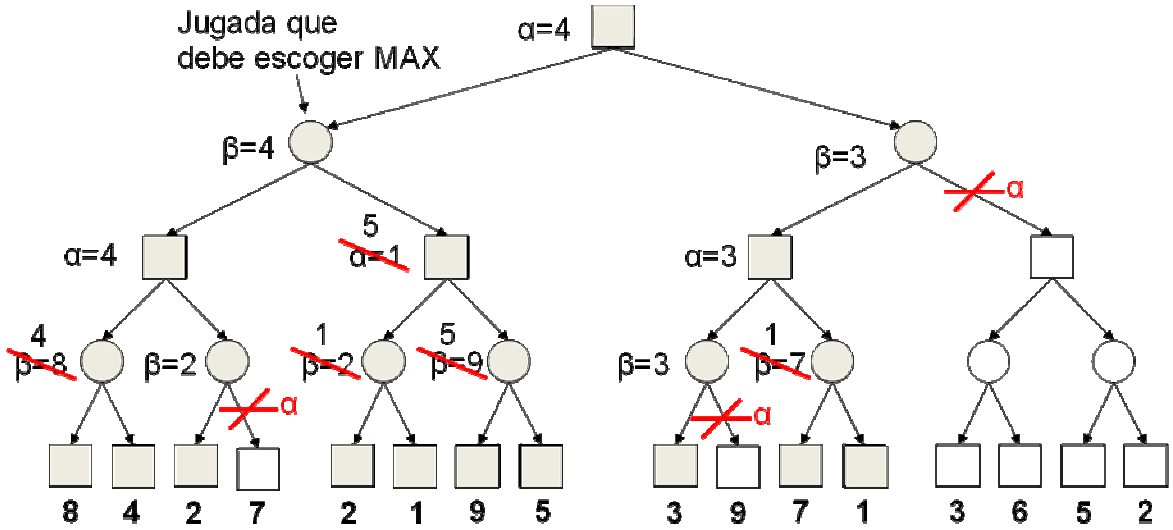

**d) (0,5 puntos) Representar de forma ordenada los estados que quedan en la cola de prioridad en la situación descrita en el punto c).**



DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

**7. (1,25 puntos) Desarrollar la poda α-β para conocer que jugada debe realizar el jugador MAX, sobre el siguiente árbol:**




a) **Sombrear los nodos que haya que desarrollar**
b) **Escribir las cotas α y β,**
c) **Marcar los cortes e indicar si son de tipo α o β,**
d) **Por último, indicar que jugada debe elegir MAX para situarse en la mejor posición posible.**

**Notas: El jugador que realiza el primer movimiento en el árbol es MAX. Los nodos del árbol se desarrollan de izquierda a derecha.**

**Ejemplo de indicaciones:**