Respuestas del examen. 8 de junio de 2.007

1. (3 puntos)

a)

Está claro que el tiempo de ejecución depende de **m**, pero no de **n**. Como máximo, el bucle **mientras** puede realizar 10 iteraciones, con un máximo de 3 llamadas recursivas. Luego el peor caso, $\mathbf{t_M(m)}$, será cuando se realicen esas 3 llamadas entre las 10 iteraciones. Por lo tanto, tenemos:

$$t_M(m) = \begin{cases} 5 + 3\cdot 10 = 35 & \text{Si } m <= 1 \\ 5 + 3\cdot 10 + 3\cdot(2+ t_M(m-1)) = 41 + 3\, t_M(m-1) & \text{Si } m > 1 \end{cases}$$

El mejor caso será cuando no se realice ninguna llamada recursiva. Para ello, la matriz `P` debe contener siempre valor FALSO, con lo cual el bucle **mientras** se ejecutará 10 veces. El número de instrucciones sería:

$$t_{mejor}(m) = 5 + 3\cdot 10 = 35 \in \Theta(1)$$

b)

Tomando el caso general del tiempo en el peor caso, $t_M(m) = 41 + 3\, t_M(m-1)$, se deduce rápidamente la ecuación característica:

$$(x-3)(x-1) = 0$$

De forma que el tiempo será de la forma:

$$t_M(m) = c_1 3^m + c_2$$

Tomando los casos base: $t_M(1) = 35$, y $t_M(2) = 41 + 3\cdot 35 = 146$, nos queda: $c_1 = 18{,}5$ y $c_2 = -20{,}5$. Por lo tanto, el tiempo de ejecución será:

$$t_M(m) = 18{,}5\cdot 3^m - 20{,}5 \in \Theta(3^m)$$

c)

Aplicando varias veces la recurrencia, tenemos:

$$t_M(m) = 41 + 3\, t_M(m-1) = 41 + 3\,(41 + 3\, t_M(m-2)) = 41(1+3) + 3^2\, t_M(m-2) =$$

$$41(1+3+3^2) + 3^3\, t_M(m-3) = \ldots = 41(1+3+3^2+\ldots+3^{m-2}) + 3^{m-1}\, t_M(1) = 41\textstyle\sum_{i=0..m-2} 3^i + 35\cdot 3^{m-1}$$

Resolviendo la serie geométrica y despejando, tenemos:

$$t_M(m) = 41(3^{m-1} - 1)/2 + 35\cdot 3^{m-1} = (41/2+35)\, 3^{m-1} - 41/2 = 55{,}5\cdot 3^{m-1} - 20{,}5 = 18{,}5\cdot 3^m - 20{,}5$$

Como cabía esperar, obtenemos el mismo resultado que en el apartado b).

d)

De manera trivial, $t_M(m) \in O(3^m)$ y $t_M(m) \in \Omega(3^m)$. Vamos a demostrar por inducción la segunda afirmación. Debemos comprobar que $t_M(m) > c\cdot 3^m$ para todo valor de **m** asintóticamente grande.

Caso Base. Para el caso base **m** = 1, la demostración es trivial:

$$t_M(1) = 35 > c\cdot 3^1$$

para algún **c**, por ejemplo 1.

Paso de inducción: Suponiendo que para cierto **m**-1 se cumple que $t_M(m-1) > c\cdot 3^{m-1}$, vamos a comprobarlo para **m**, usando la fórmula recursiva del apartado a):

$$t_M(m) = 41 + 3\, t_M(m-1) > 3\, t_M(m-1) > 3\cdot c\cdot 3^{m-1} = c\cdot 3^m \;\therefore$$

Luego, hemos demostrado por inducción que $t_M(m) \in \Omega(3^m)$. La otra demostración es un poco más compleja.

2. (2 puntos)

Se puede ver fácilmente que el ejercicio es una variante del problema de la mochila 0/1, con las siguientes diferencias: existen dos restricciones de coste (por precio y por tiempo); y los objetos se pueden seleccionar 0, 1 ó 2 veces.

Como en el caso de la mochila 0/1, podemos aplicar **avance rápido** para obtener una buena solución, aunque es difícil que se pueda garantizar la óptima. En esencia, el algoritmo consistirá en seleccionar los artistas con mejor ratio $\mathbf{c_i}/(\mathbf{d_i}\,\mathbf{p_i})$ (calidad por minuto y por euro) mientras no superen el presupuesto y el tiempo máximos. Siempre que quepa, se intentará añadir dos actuaciones del artista seleccionado.

La solución se representará mediante una tupla **s**= ($\mathbf{s_1}$, $\mathbf{s_2}$, …, $\mathbf{s_n}$), donde cada $\mathbf{s_i}$ puede valer 0, 1 ó 2, e indica el número de actuaciones del artista **i**-ésimo. El valor de inicialización será -1.
En definitiva, el algoritmo podría ser como el siguiente.

```
operación MurciaVoraz (c, d, p: array [1...n] de real; A, B: real; var s: array [1…n] de entero) : real
    s:= (-1, -1, …, -1)
    cacum:= 0              // Calidad total acumulada
    dacum:= 0              // Tiempo total acumulado
    pacum:= 0              // Precio total acumulado
    para i:= 1, …, n hacer
        j:= argmin_{∀a, con s[a]=-1} c[a]/(d[a]·p[a])
        si c[j]<=0 entonces
            s[j]:= 0
        sino si (dacum+2d[j] <= A) AND (pacum+1.5p[j] <= B) entonces
            s[j]:= 2
            dacum:= dacum + 2d[j]
            pacum:= pacum + 1.5p[j]
        sino si (dacum+d[j] <= A) AND (pacum+p[j] <= B) entonces
            s[j]:= 1
            dacum:= dacum + d[j]
            pacum:= pacum + p[j]
        sino
            s[j]:= 0
        finsi
    finpara
    devolver cacum
```

Tal y como está especificado, el tiempo del algoritmo sería un O($\mathbf{n}^2$). No obstante, ordenando los objetos a priori por el criterio **c/dp** se podría conseguir un O(**n** log **n**). La ejecución del ejemplo se deja como ejercicio. La solución es **s** = (0, 2, 2, 0, 2, 1, 2, 0, 0), con **cacum** = 70, **dacum** = 35, **pacum** = 23.

3. (2,5 puntos)
Planteando el problema como una serie de toma de decisiones, la decisión que surge para cada artista es si lo invitamos una, dos o ninguna vez. La forma de la ecuación recurrente será **PD(i, a, b)**, que consiste en resolver el problema con los **i** primeros artistas, tiempo **a** y presupuesto **b**. La ecuación será:

$$\mathbf{PD(i,a,b)} = \begin{cases} -\infty & \text{Si } i<0 \text{ OR } a<0 \text{ OR } b<0 \\ 0 & \text{En otro caso, si } i=0 \text{ OR } a=0 \text{ OR } b=0 \\ \max\{PD(i-1,a,b), & \text{En otro caso} \\ \quad PD(i-1,a-d[i],b-p[i])+c[i], & \\ \quad PD(i-1,a-2d[i],b-1.5p[i])+2c[i]\} & \end{cases}$$

La multiplicación por 1.5 puede producir valores no enteros, que se saldrían de la tabla. Para evitar este problema, multiplicaremos todos los precios por 2. En consecuencia, la tabla usada por el algoritmo será de tres dimensiones: **D**: array [0…n, 0…A, 0…2B] de entero, y el algoritmo para rellenarlas:

```
D:= 0
para i:= 1, …, n hacer
    para a:= 1, …, A hacer
        para b:= 1, …, 2B hacer
            D[i, a, b]:= max(D[i-1, a, b], D[i-1, a-d[i], b-2p[i]]+c[i], D[i-1, a-2d[i], b-3p[i]]+2c[i])
```

El beneficio óptimo se encuentra en **D[n, A, 2B]**. La solución se puede encontrar partiendo de esa posición y analizando las decisiones que se tomaron en cada caso (es decir, cuál de las tres opciones es la máxima).

4. (2,5 puntos)
Podemos usar la misma representación de la solución que en el ejercicio 2, es decir, una tupla **s**= ($\mathbf{s_1}$, $\mathbf{s_2}$, …, $\mathbf{s_n}$), donde cada $\mathbf{s_i}$ es el número de actuaciones de **i** (0, 1 ó 2). Los valores acumulados de calidad, tiempo y precio formarán ahora parte del tipo **Nodo**. Éste será de la forma:

```
tipo Nodo = registro
        tupla: array [1…n] de entero
        nivel: entero
        cacum, dacum, pacum: entero
        CI, CS, BE: real
finregistro
```

El nodo raíz es:

```
raiz.nivel:= 0
raiz.cacum:= 0
raiz.dacum:= 0
raiz.pacum:= 0
```

La forma del iterador abstracto **para todo nodo y hijo de x** será como la siguiente:

```
para i:= 0, …, 2 hacer
        y:= x
        y.nivel:= x.nivel + 1
        y.tupla[y.nivel]:= i
        y.cacum:= y.cacum + i*c[y.nivel]
        y.dacum:= y.dacum + i*d[y.nivel]
        y.pacum:= y.pacum + i^0,585*p[y.nivel]
        si y.dacum>A OR y.pacum>B entonces break
        …
```

Las funciones **Solución** y **Valor** son triviales:

```
operación Solución (n: Nodo) : booleano
        devolver n.nivel==n

operación Valor (n: Nodo) : real
        devolver n.cacum
```

En cuanto a las cotas de un nodo, para la cota inferior podemos usar el algoritmo voraz del ejercicio 2. Para la cota superior podemos suponer que el tiempo y el dinero restante se van a invertir en el artista con la mejor proporción de calidad por unidad de tiempo y coste. El beneficio estimado podría ser la media de ambas cotas. En definitiva, tenemos:

```
y.CI:= y.cacum + MurciaVoraz(c[y.nivel+1…n], d[y.nivel+1…n], p[y.nivel+1…n], A-y.dacum, B-y.pacum, s)
y.CS:= y.cacum + (A-y.dacum)(B-y.pacum)·max_{∀k=y.nivel+1…n} c[k]/(0,75·d[k]p[k])
y.BE:= (y.CI + y.BS)/2
```

El valor 0,75 es debido a que la segunda actuación de un artista cuesta la mitad, luego el coste medio en el mejor caso será (1+0,5)/2 = 0,75.

La estrategia de ramificación será una MB-FIFO: explorar primero los nodos de la LNV con mayor beneficio estimado y en caso de empate seguimos, por ejemplo, por el primero de los que se introdujeron. En cuanto a la estrategia de poda, la variable de poda **C** será la mayor de las cotas inferiores de los nodos generados, o de las soluciones finales obtenidas hasta ese momento. Podamos un nodo **j** cuando **CS(j)** <= **C**. Con esto quedaría completamente especificado el algoritmo. El esquema de ramificación y poda sería exactamente el mismo que el visto en clase.