PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

Septiembre 2015

Normas de valoración del examen:

- La nota del examen representa el 80% de la valoración final de la asignatura (el 20% restante corresponde a las prácticas).
- Cada cuestión contestada correctamente vale 1 punto.
- Cada cuestión contestada incorrectamente baja la nota en 0.3 puntos.
- Debe obtenerse un mínimo de 3 puntos en las cuestiones para que el problema sea valorado (con 3 cuestiones correctas y alguna incorrecta el examen está suspenso).
- La nota total del examen debe ser al menos de 4.5 para aprobar.
- Las cuestiones se responden en una hoja de lectura óptica.

SOLUCIONES:

Test:

Tipo A: 1D 2A 3D 4A 5A 6C
Tipo B: 1C 2D 3D 4A 5A 6A

Problema (4 puntos).

Una pareja decide divorciarse tras 10 años de matrimonio. Deciden repartir su patrimonio a partes iguales. Cada uno de los *n* activos (indivisibles) que hay que repartir tiene un valor entero positivo. Los cónyuges quieren repartir dichos activos a medias y, para ello, primero quieren comprobar si el conjunto de activos se puede dividir en dos subconjuntos disjuntos, de forma que cada uno de ellos tenga el mismo valor.

La resolución del problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos)
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto)
3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto)

Solución:

1. No se puede encontrar una función de selección que garantice, sin tener que reconsiderar decisiones, una elección de los activos que cumpla con la restricción del enunciado, por ello no se puede aplicar el esquema voraz. Tampoco se puede dividir el problema en subproblemas que al combinarlos nos lleven a una solución. Al no ser un problema de optimización, el esquema de exploración de grafos más adecuado es el **esquema vuelta-atrás**. Vuelta atrás es un recorrido en profundidad de un grafo dirigido implícito. En él una

solución puede expresarse como una n-tupla [x_1, x_2, x_3,… x_n], donde cada x_i, representa una decisión tomada en la etapa i-ésima, de entre un conjunto finito de alternativas.

El esquema general de la técnica de vuelta-atrás se puede encontrar en el libro de texto de la asignatura, en la página 161.

En este caso, el espacio de búsqueda es un árbol de grado 2 y altura n+1. Cada nodo del i-ésimo nivel tiene dos hijos correspondientes a si el i-ésimo activo va a un cónyuge o al otro.

Para poder dividir el conjunto de activos en dos subconjuntos disjuntos, de forma que cada uno de ellos tenga el mismo valor, su valor debe ser par. Así, si el conjunto inicial de activos no tiene un valor par el problema no tiene solución.

2. Descripción de las estructuras de datos necesarias.

   - El conjunto de activos y sus valores se representa en un array de enteros v= [v_1, v_2, v_3,… v_n], donde cada v_i representa el valor del activo i-ésimo.
   - La solución se representa mediante una array de valores x= [x_1, x_2, x_3,… x_n], donde cada x_i, podrá tomar el valor 1 o 2 en función de que el activo i-ésimo se asigne al subconjunto de un cónyuge o del otro.
   - Un array de dos elementos, suma, que acumule la suma de los activos de cada subconjunto.

3. Algoritmo completo a partir del refinamiento del esquema general.

   Una posible condición de poda consistirá en que se dejarán de explorar aquellos nodos que verifiquen que alguno de los dos subconjuntos que se van construyendo tiene un valor mayor que la mitad del valor total.

```
función Solucion (k: entero; suma: array[1..2]) dev booleano
        si (k = n) AND (suma[1] = suma [2]) entonces
                dev Verdadero
        sino
                dev Falso
        fsi
ffuncion


función SeparacionConyugesMitad (v: array[1..n], k: entero, suma: array[1..2], sumaTotal: entero)
        dev x: array[1..n]; separacion: boolean

si Solucion (k, suma) entonces
        dev x, Verdadero
sino
        para i desde 1 hasta 2 hacer
                x[k] ← i
                suma[i] ← suma[i] + v[k]
                si suma[i] <= (sumaTotal DIV 2) entonces
                        si k < n entonces
                                x ← SeparacionConyugesMitad(v, k+1, suma, sumaTotal)
                        fsi
```

```
                sino
                        suma[i] ← suma[i] - v[k]
                fsi
        fpara
fsi
ffunción


función ProblemaSeparacionConyugesMitad (v: n-tupla) dev x: n-tupla; separacion: boolean

        para i desde 1 hasta n hacer
                sumaTotal ← sumaTotal + v[i]
        fpara

        si par(sumTotal) entonces
                dev SeparacionConyugesMitad (v, 1, suma[0,0], sumaTotal)
        sino
                dev 0
        fsi
ffuncion
```

Llamada inicial ProblemaSeparacionConyugesMitad (v);

4. Estudio del coste del algoritmo desarrollado. En este caso, el coste viene dado por el número máximo de nodos del espacio de búsqueda, esto es: $T(n) \in O(2^n)$