Apellido y Nombre:

nota

1	2	3	4	L

Algoritmos y Estructuras de Datos II **Examen Final** **24/7/2006**

1. (2pts) Se debe calcular el número de veces que el procedimiento p escribe la palabra "hola" en función de la entrada n. Para ello
   a) Definir la recurrencia correspondiente.
   b) Resolverla.
   Obs: recuerde que usted mismo puede comprobar que la recurrencia esté correctamente resuelta. El argumento de p puede ser cualquier número natural $n \geq 0$.

```
proc p(in n:nat)
    if n ≥ 1 then
        p(n-1)
        for i:= 1 to 2^n do
            for j:= 1 to n do escribir("hola") od
        od
        p(n-1)
    fi
end
```

2. (3pts) Se está organizando un campeonato de fútbol con n equipos. Cada equipo debe jugar exactamente una vez con cada uno de sus oponentes. Se desea diseñar un fixture, es decir una tabla donde se describe con quién tiene que jugar cada equipo en cada fecha. Por simplicidad asumimos que n es potencia de 2, es decir, $n = 2^k$ para algún número natural k. Como cada equipo tiene que jugar con cada uno de los restantes $n-1$ equipos habrá en total $n-1$ fechas.

   El siguiente es un ejemplo de una tabla con 8 equipos (7 fechas):

	Equipo							
Fecha	1	2	3	4	5	6	7	8
1	2	1	4	3	6	5	8	7
2	3	4	1	2	7	8	5	6
3	4	3	2	1	8	7	6	5
4	5	6	7	8	1	2	3	4
5	6	7	8	5	4	1	2	3
6	7	8	5	6	3	4	1	2
7	8	5	6	7	2	3	4	1

   Observe las tres primeras filas de la tabla. Se puede ver que está formada por 2 "sub-fixtures": sus primeras cuatro columnas contienen precisamente un fixture para los cuatro primeros equipos y sus últimas cuatro columnas contienen precisamente un fixture para los cuatro últimos equipos. Algo similar puede decirse de la primera fila de la tabla: está formada por 4 "sub-fixtures" muy pequeños: cada uno de ellos es para 2 equipos.
   a) Basándose en estas observaciones, dé un algoritmo que utilice la técnica "divide y vencerás" para armar el fixture con las $n-1$ fechas para n potencia de 2 arbitraria. Explique sus ideas, justifique detalladamente.
   b) ¿Cuál es la complejidad del algoritmo? Justifique.

3. (3pts) El tipo abstracto *multiconjunto* (conjuntos de enteros con repeticiones) tiene operaciones
   - *empty*: Multiconjunto sin elementos.
   - $ins(x, M)$: Inserta (una ocurrencia más de) el entero x en el multiconjunto M.
   - $del(x, M)$: Borra una ocurrencia del entero x en el multiconjunto M.
   - $mult(x, M)$: Devuelve el número de veces que ocurre el entero x en M.
   - $isEmpty(M)$: Devuelve *true* si y sólo si el multiconjunto M es vacío.

   Por ejemplo:

$$ins(5, \{1, 1, -4, 5\}) = \{1, 1, -4, 5, 5\}$$
$$ins(5, \{1, 1, -4\}) = \{1, 1, -4, 5\}$$
$$del(1, \{1, 1, -4, 5\}) = \{1, -4, 5\}$$
$$del(1, \{1, -4, 5\}) = \{-4, 5\}$$
$$mult(1, \{1, 1, 4, 5\}) = 2$$
$$mult(3, \{1, 1, 4, 5\}) = 0$$

   Implemente el tipo *multiconjunto* con sus cinco operaciones utilizando árboles binarios de búsqueda en los que cada nodo contiene un par de enteros (x, m) indicando que el elemento x ocurre m veces en el multiconjunto. No está permitido utilizar punteros ni ninguna otra implementación específica de los árboles binarios, utilice su notación abstracta ($< >$, $< l, e, r >$, etc.).

4. (2pts) Sea $G = (N, A)$ un grafo dirigido sin ciclos donde N es el conjunto de nodos y A el de aristas. Para cada arista $a \in A$ se tiene que $O(a) \in N$ y $D(a) \in N$ son respectivamente el nodo origen y destino de a. Un camino es una lista de aristas $C = a_1, \ldots, a_k$ tal que para todo $i \in \{1 \ldots k-1\}$, $D(a_i) = O(a_{i+1})$. Cuando un camino es no vacío (es decir, cuando tiene al menos una arista) la misma notación puede utilizarse: $O(C) = O(a_1)$ y $D(C) = D(a_k)$. Utilice la técnica de backtracking para escribir un algoritmo que determine la longitud del camino más largo de G. Explique sus ideas.

5. (Para alumnos libres). Se presenta la siguiente variante del problema de la moneda. Las monedas de algunas denominaciones son difíciles de conseguir mientras que otras son fáciles de conseguir. El problema ahora consiste en contar el menor número de monedas difíciles necesarias para pagar un monto determinado, sin importar cuántas monedas fáciles se utilizan. Sea $k_i = fácil$ o $k_i = difícil$ según sea fácil o difícil conseguir monedas de denominación d_i.
   a) Plantear una definición recursiva de $m(i, j)$ donde $m(i, j)$ es el menor número de monedas difíciles necesarias para pagar el monto j utilizando sólo monedas de denominación $d_1, \ldots, d_i$.
   b) Dar un algoritmo que utilice programación dinámica para resolver esta variante del problema de la moneda. El algoritmo tomará, además de los parámetros habituales, un arreglo k : **array**[1..n] **of** $\{fácil, difícil\}$.