Apellido y Nombre:
e-mail:

escrito	lab	nota

1	2	3	4	5	L

Algoritmos y Estructuras de Datos II **Examen Final** **2/7/2008**

Importante: Escribir nombre y apellido en todas las hojas, inclusive ésta que también debe ser entregada. Resolver cada ejercicio en hoja aparte.

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

2. (2,5pts) Especificar el TAD GruposUsuarios (en el sentido de grupos de usuarios). El tipo tendrá sólo dos constructores, uno que representa la situación inicial (en la que no hay ningún grupo y ningún usuario) y otra que permite justamente agregar un usuario a un grupo (asumiendo que si el grupo aún no existía, comienza a existir).

   TAD GruposUsuarios
   constructores
   nohaygrupos :: GruposUsuarios
   agregarusuario :: Usuario → Grupo → GruposUsuarios → GruposUsuarios

   Observar que con estos dos constructores todo grupo tendrá siempre al menos un usuario. Deberá dar la(s) ecuacion(es) que exprese(n) que al agregar un usuario a un grupo no tiene efecto si el usuario ya estaba en el grupo. Deberá también dar la(s) ecuacion(es) que exprese(n) que no importa el orden en que se agreguen los usuarios.

   Declare y dé las ecuaciones para las siguientes cinco operaciones: una que elimina un usuario de un grupo (y no tiene efecto si el usuario no estaba en el grupo), una que elimine un grupo (y no tiene efecto si el grupo no existía), una que permita decidir si un usuario dado está o no en un grupo dado, una que permita decidir si dos usuarios dados comparten o no algún grupo y una última que sirva para fusionar dos grupos, es decir, dados dos grupos diferentes elimine el segundo e incremente el primero con los usuarios del segundo. Esta última operación no debe producir ningún efecto en el caso en que los dos grupos dados sean en realidad el mismo.

3. (2pts) Dado un arreglo a de enteros positivos y un entero positivo k, se debe escribir un algoritmo que utilice la técnica divide y vencerás para hallar y devolver un elemento que aparece exactamente k veces en a. En caso de no existir tal elemento, el algoritmo debe devolver 0. En caso de existir más de un elemento, el algoritmo puede devolver cualquiera de ellos. Explique detalladamente su solución. Ayuda: utilizar como subalgoritmo una versión del algoritmo de pivote que coloca en forma contigua todos los elementos iguales al pivot. **Sólo para los alumnos libres:** dar además ese subalgoritmo.

4. (2pts) Tenemos que comprar n productos recorriendo los m supermercados de la ciudad. Sabiendo que los precios están dados por una matriz p : **array**[1..n,1..m] **of nat** de manera tal que $p[i,j]$ es el precio del producto i en el supermercado j, se pide dar un algoritmo que determina en qué supermercado debe comprarse cada producto. Para ello, el algoritmo devuelve un arreglo s : **array**[1..n] **of nat** tal que $s[i]$ indica en qué supermercado conviene comprar el producto i. Explicar detalladamente la solución. ¿Qué técnica de programación ha utilizado?

5. (1,5pts) Se tiene un grafo $G = (N, A)$ donde N es el conjunto de nodos y A el de aristas dirigidas. La función $o : A \to N$ devuelve el origen de cada arista. La función $d : A \to N$ devuelve el destino de cada arista. Finalmente, las aristas están coloreadas. La función $c : A \to$ Colores devuelve el color de cada arista. Observar que esta definición no excluye la posibilidad de que haya varias aristas que tengan igual origen e igual destino (pueden incluso tener igual color). Dados $x, y \in N$ el siguiente algoritmo utiliza backtracking para encuentrar el menor número de colores que hace falta utilizar para ir de x a y.

```
fun menos_colores(x : N,y : N) ret n: nat
                                   {calcula el menor número de colores para ir de x a y}
  n:= backtrack(x,y,{x},{})
end
fun backtrack(x' : N, y : N, V :cjto de N, C :cjto de Colores) ret n: nat
                        {x' es el último nodo visitado en el camino que se está construyendo}
                                {V es el conjunto de los nodos visitados en ese camino}
                              {C es el conjunto de los colores de las aristas de ese camino}
  if x' = y → n:= |C|
    x' ≠ y → n:= ∞
             for a ∈ A do
                 if o(a) = x' ∧ d(a) ∉ V →
                                  n:= min(n,backtrack(d(a),y,V ∪ {d(a)},C ∪ {c(a)})) fi
             od
  fi
end
```

Se pide explicar el algoritmo lo más detalladamente posible. Justificar claramente cada asignación a n y los valores de cada uno de los parámetros en cada llamada a la función backtrack.

Explicar si el algoritmo siempre termina o no y por qué.

Explicar qué ocurre con grafos no conexos. Y qué ocurre con grafos con ciclos.