Algoritmos y Estructuras de Datos II - 22 de abril de 2015

Primer Parcial

Alumno: .....

Siempre se debe explicar la solución, una respuesta correcta no es suficiente sino viene acompañada de una justificación que demuestre que la misma ha sido comprendida. Las explicaciones deben ser completas. En el ejercicio de implementación, debe utilizarse pseudo-código. **La utilización de código c influirá negativamente.**
Por favor, desarrollá cada solución en **hojas diferentes** y escribí claramente **tu nombre** en todas las hojas, ya que las mismas se separarán para agilizar la corrección.

1. Calculá el orden de cada uno de los siguientes algoritmos:

   (a)
   ```
   proc P(in/out a: array[1..n] of nat)
       for i:= 1 to n do
           isort(a)
           swap(a,1,n)
       od
   end proc
   ```
   donde isort es el algoritmo de ordenación por inserción.

   (b)
   ```
   proc Q(in/out a: array[1..n] of nat, in izq, der: nat)
       var med : nat
       if izq < der then
          ssort(a,izq,der)
          med:= (izq + der) div 2
          Q(a,izq,med)
          Q(a,med+1,der)
       fi
   end proc

   proc main(in/out a: array[1..n] of nat)
       Q(a,1,n)
   end proc
   ```
   donde ssort(a,izq,der) ordena el del arreglo a entre las posiciones izq y der utilizando el algoritmo de ordenación por selección.

2. (a) Dado el siguiente algoritmo, planteá la recurrencia que indica la cantidad de asignaciones realizadas en función de la entrada n:

   ```
   fun f (n: nat) ret m: nat
       if n ≤ 1 then
           m := 1
       else
           m := f(n-1) + 2*f(n-2)
       fi
   end
   ```

   (b) Resolvé la siguiente recurrencia: $t(n) = \begin{cases} n & \text{si } n \in \{0,1\} \\ t(n-1) + 2t(n-2) & \text{si } n > 1 \end{cases}$

3. Ordená las siguientes funciones según el orden creciente de sus $\mathcal{O}$, estableciendo claramente en cuáles casos vale el $=$ y en cuáles el $\subset$ estricto. Justificá utilizando la jerarquía y las propiedades demostradas en la teoría.

   (a) $4^{\log_2 n}$ (b) $\sqrt{2n} * n^{1.5} + n * \sqrt[3]{n^2}$ (c) $n * \log_2 3^n$ (d) n^n (e) n^{n+1} (f) $(n+1)^n$

4. A continuación se especifica el TAD Contador:

   module TADContador **where**

   data Contador = Inicial
   | Incrementar Contador

   es_inicial :: Contador → Bool
   decrementar :: Contador → Contador - - se aplica solo a un Contador que no sea Inicial

   es_inicial Inicial = True
   es_inicial (Incrementar c) = False

   decrementar (Incrementar c) = c

   Asumimos que max_nat es el máximo número natural representable en nuestro lenguaje de programación, y que resulta insuficiente para la aplicación que estamos construyendo, pues el contador podría incrementarse más de max_nat veces. Por eso, te pedimos que implementes el TAD Contador de las dos siguientes maneras:

   (a) utilizando la representación

   type counter = **tuple**
   left: nat
   right: nat
   end

   con la intención de permitir más de max_nat × max_nat incrementos. Intuitivamente el natural de la izquierda (left) sería el "dígito" más significativo y el de la derecha (right), el "dígito" menos significativo. Implementá las cuatro operaciones del TAD, y una función es_max_value que devuelva verdadero siempre y cuando su argumento sea el máximo contador representable. No olvides las pre- y poscondiciones de cada operación.

   (b) utilizando la representación

   type counter = [nat]

   con la intención de permitir una cantidad ilimitada de incrementos, asumiendo que el lenguaje de programación viene equipado con listas. Implementá las cuatro operaciones del TAD nuevamente sin omitir las pre- y poscondiciones. La idea de esta implementación es representar valores elevados del contador utilizando el número de "dígitos" que sea necesario.

5. La empresa en la que trabajás le encarga a tu equipo de programadores el desarrollo de un video juego que, entre otras componentes, tendrá un robot que se desplaza por una grilla cuadrada de 10 por 10 realizando movimientos de un paso horizontal o verticalmente. Luego de pensar y discutir el problema con tus compañeros, descubrís la conveniencia de definir el TAD Recorrido para modelar el recorrido que el robot realiza desde el comienzo del juego. Te pedimos que especifiques el TAD Recorrido. Para ello, considerá el constructor **Ini**, que modela el recorrido trivial en que aún no se ha realizado ningún desplazamiento; y cuatro constructores más: **Izq**, **D**er, **Arr** y **Aba**. Cada uno de ellos, a partir de un recorrido construye otro que incluye un desplazamiento posterior hacia la izquierda, derecha, arriba o abajo respectivamente. Además de estos constructores, el TAD debe contar con las operaciones **es_ini**, que determina si un recorrido es el construido por Ini o no, **es_izq**, que determina si el último desplazamiento del recorrido es hacia la izquierda, y similarmente, **es_der**, **es_arr** y **es_aba**. El TAD debe incluir una operación **deshacer**, que deshace el último movimiento de un recorrido, y dos operaciones **fila** y **columna** que devuelven el número de fila y el número de columna en que el robot se encontrará al finalizar un recorrido dado. Para su definición, tené en cuenta que el robot comienza en la fila 0 y columna 0, que cada movimiento es un solo paso, salvo cuando el robot se encuentra en los bordes: por ejemplo, si está en el borde izquierdo y se desplaza hacia la izquierda debe aparecer en la misma fila en que se encontraba pero en el borde derecho. Y de manera similar en cada uno de los otros tres bordes de la grilla.