Examen de Programación 3

4 de febrero de 2017

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Parte obligatoria

Esta parte es eliminatoria. Para la aprobación del examen debe obtenerse un mínimo del **50 % de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas.

Pregunta 1 (15 puntos)

Sean $f, g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ funciones de los naturales en los reales no negativos. Demuestre o muestre un contraejemplo de cada una de las siguientes proposiciones.

(a) $f + g \in O(\text{máx}(f, g))$.

(b) $f + g \in \Omega(\text{máx}(f, g))$.

(c) $\text{máx}(f, g) \in \Theta(f + g)$.

Solución:

Las tres proposiciones son verdaderas.

(a) Por definición de orden O se debe demostrar que existen $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $f(n) + g(n) \leq c \cdot \text{máx}(f(n), g(n))$ para $n \geq n_0$.

Como

$$f(n) + g(n) \leq 2 \cdot \text{máx}(f(n), g(n)) \quad \forall n \geq 0.$$

tomando $c = 2$ y $n_0 = 0$ se cumple la proposición.

(b) Por definición de Ω se debe demostrar que existen $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $f(n) + g(n) \geq c \cdot \text{máx}(f(n), g(n))$ para $n \geq n_0$.

Como

$$f(n) + g(n) \geq \text{máx}(f(n), g(n)) \quad \forall n \geq 0.$$

tomando $c = 1$ y $n_0 = 0$ se cumple la proposición.

(c) De la parte a y la parte b se obtiene que $f + g \in O(\text{máx}(f, g)) \cap \Omega(\text{máx}(f, g))$, lo que por definición de orden exacto significa que $f + g \in \Theta(\text{máx}(f, g))$. Por la propiedad $h \in \Theta(k) \iff k \in \Theta(h)$ vista en el curso se llega a $\text{máx}(f, g) \in \Theta(f + g)$.

Pregunta 2 (12 puntos)

Sea $G = (V, E)$ un grafo conexo no dirigido con una función de costo definida sobre E. Sean U un subconjunto de los vértices de V y A un subconjunto de las aristas de E tal que toda arista de A tiene uno de sus extremos en U y el otro en $V - U$.

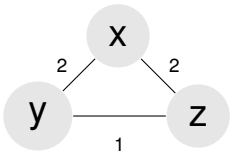
Se propone el siguiente enunciado como la propiedad `MST` vista en el curso:

> Sea $a \in A$ una arista tal que ninguna arista de A tiene costo menor estricto que a. Entonces si T es un árbol de cubrimiento de costo mínimo de G, a es una arista de T.

Muestre con un contraejemplo que el enunciado no es correcto y escríbalo correctamente.

Solución:

El enunciado no es correcto porque afirma que a debe pertenecer a todo árbol de cubrimiento de costo mínimo de G pero no se asegura que a sea la única arista de costo mínimo en A.



Si en el ejemplo que se muestra a la izquierda se considera que $U = \{x\}$, entonces $A = \{(x, y), (x, z)\}$ y por lo tanto la arista (x, y) cumple con las condiciones para ser la arista a. Sin embargo el árbol formado por las aristas (x, z) y (y, z) es un árbol de cubrimiento de costo mínimo de G.

Un enunciado correcto es:

> Sea $a \in A$ una arista tal que ninguna arista de A tiene costo menor o igual a. Entonces si T es un árbol de cubrimiento de costo mínimo de G, a es una arista de T.

Más en general, permitiendo que haya aristas de igual peso:

> Sea $a \in A$ una arista tal que ninguna arista de A tiene costo menor que a. Entonces existe un árbol de cubrimiento de costo mínimo de G que tiene a a como una de sus aristas.

Pregunta 3 (13 puntos)

(a) Sea `FibDC` el algoritmo que calcula los números de Fibonacci mediante la técnica *Dividir y Conquistar*. Llamamos $S(n)$ a la cantidad de sumas que se realizan al invocar `FibDC(n)`. Plantee la recurrencia que calcula $S(n)$.

(b) Escriba el algoritmo `FibPD` que calcula los números de Fibonacci mediante *Programación Dinámica*. El algoritmo debe usar $\Theta(1)$ espacio.

(c) Construya una tabla con la cantidad de sumas que realizan `FibDC` y `FibPD` para los valores de n desde 0 hasta 10. Explique en una oración a qué se debe la diferencia del crecimiento de la versión `FibDC` con respecto al de la versión `FibPD`.

Solución:

(a)
```
FibDC(n)
    if (n = 0) OR (n = 1)
        Fn = 1
    else
        Fn = FibDC(n-1) + FibDC(n-2)
    return Fn
```

$$S(n) = \begin{cases} 0 & \text{si } n \in \{0,1\} \\ 1 + S(n-1) + S(n-2) & \text{en otro caso.} \end{cases}$$

(b)
```
FibPD(n)
    F0 = 0
    F1 = 1
    Fn = 1
    for i = 2 to n
        F0 = F1
        F1 = Fn
        Fn = F0 + F1
    return Fn
```

(c)

n	0	1	2	3	4	5	6	7	8	9	10
FibDC	0	0	1	2	4	7	12	20	33	54	88
FibPD	0	0	1	2	3	4	5	6	7	8	9

El crecimiento de la versión `FibDC` es mayor debido a que la superposición de subproblemas hace que cada subproblema se resuelva varias veces.

La cantidad de sumas de la versión `FibPD(n)` es $n-1$ para $n > 0$. La de la versión `FibDC(n)` es $S(n)$ que se puede demostrar que es igual a $Fibonacci(n) - 1$.

Problemas

Problema A (30 puntos)

Una organización va a comenzar un nuevo proyecto para el cual convoca a un conjunto de sus funcionarios, a los que identifica con números del 1 al n. Los funcionarios serán divididos en equipos, cada uno de los cuales debe tener un coordinador que lo identifica, a quien se le pagará un aumento. Ese aumento, que varía según el funcionario, se mantiene en el arreglo `Costo` de tamaño n. El coordinador debe ser uno de los integrantes del equipo. Se pretende invertir el mínimo posible en los aumentos para los coordinadores. Cada funcionario debe ser asignado a un equipo, y sólo a uno. Algunos de los funcionarios están vinculados a otros en proyectos anteriores. Se exige que en ningún grupo queden funcionarios que ya estaban vinculados a otros funcionarios del mismo grupo. El conjuntos de vínculos de cada funcionario se mantiene en el arreglo `Vinculos`. Se asume que si j pertenece a `Vinculos`[i] entonces i pertenece a `Vinculos`[j]. Ningún funcionario pertenece a su propio conjunto de vínculos.

Exprese, con lenguaje natural y formal, en términos de `Backtracking` el problema de organizar los grupos.

(I) Defina la forma de la tupla.

(II) Formalice las siguientes expresiones, indicando a que categoría corresponden: restricción explícita, restricción implícita, función objetivo, predicado de poda.

a. Si un funcionario es coordinador de un equipo, debe ser el coordinador del equipo que integra.

b. No se asigna como coordinador del equipo de un funcionario i alguien que es más costoso que el propio i.

c. El coordinador del equipo de cada funcionario no pertenece a la lista de vínculos del funcionario.

(III) Complete la formalización.

Solución:

(I) **Forma de la tupla** Tupla $t = \langle t_1, \cdots, t_n \rangle$, de largo fijo n. Cada componente identifica el coordinador del equipo al que queda integrado el funcionario.

(II) a. Es una restricción implícita.

$$t_{t_i} = t_i, 1 \leq i \leq n.$$

b. Es un predicado de poda. Si no se cumple

$$Costo[t_i] \leq Costo[i], 1 \leq i \leq n$$

se debe detener la construción de la tupla porque asignar a i como coordinador del grupo generaría un costo menor.

c. Es una restricción explícita.

$$t_i \notin Vinculos[i], 1 \leq i \leq n.$$

(III) **Restricciones explícitas**

1. El coordinador pertenece al conjunto de funcionarios:

$$t_i \in \{1, \cdots, n\}, 1 \leq i \leq n.$$

2. El coordinador no pertenece a la lista de vínculos del funcionario:

$$t_i \notin Vinculos[i], 1 \leq i \leq n.$$

Esto es lo pedido en II c.

Restricciones implícitas

1. No se puede asignar un funcionario a un equipo al que ya se ha asignado otro con quien tiene vínculo:

$$t_i = t_j, j < i, \implies j \notin Vinculos[i], 1 \leq i \leq n.$$

2. No se puede asignar como coordinador de equipo un funcionario al que se había asignado otro coordinador:
$$t_i < i \implies t_{t_i} = t_i, 1 \leq i \leq n.$$
3. Si un funcionario ha sido asignado como coordinador de equipo, debe ser su propio coordinador de equipo:
$$\exists\, j < i \text{ tal que } t_j = i \implies t_i = i, 1 \leq i \leq n.$$

Las últimas dos restricciones pueden unificarse:

$$t_{t_i} = t_i, 1 \leq i \leq n.$$

Corresponden a lo que se pide en II a. La razón de separarlos es porque mediante la restriccion implicita 2 se logra podar tempranamente.

Función objetivo Definimos la función `esCoordinador` que indica si un funcionario es coordinador de su grupo:

$$\texttt{esCoordinador}(i) = \begin{cases} 1 & \text{si } t_i = i \\ 0 & \text{en otro caso.} \end{cases}$$

La función objetivo es

$$f = \min_{t \in T} \sum_{i=1}^{n} \texttt{esCoordinador}(i) \cdot Costo[i],$$

siendo T el conjunto de tuplas solución.

Predicados de poda No se sigue construyendo una tupla si se pretende asignar como coordinador del equipo de un funcionario a alguien que es más costoso que el propio funcionario. Entonces, se poda si

$$Costo[t_i] > Costo[i], 1 \leq i \leq n.$$

Esto es lo pedido en II b.

Problema B (30 puntos)

Se pretende ordenar de manera decreciente un arreglo de enteros no negativos con índices entre 0 y $n-1$. El arreglo está formado por 2^h segmentos cada uno de los cuales tiene longitud k y está ordenado de manera decreciente.

En lo que sigue se considera que la operación básica es la comparación entre elementos del arreglo.

(I) Implemente el algoritmo `ordenar`. El tiempo de ejecución en el peor caso debe ser $\Theta(n \cdot h)$. Fundamente por qué se cumple la cota de tiempo requerida.

```
void ordenar(int A[], int n, int k, int h);
```

Nota: Puede usar, asumiendo implementada, la funcion `merge` de la parte II, que ordena de manera decreciente un segmento formado por dos segmentos consecutivos que están ordenados de manera decreciente.

(II) Implemente la funcion `merge`. Muestre que el peor caso del tiempo de ejecución está en $\Theta(\text{largo})$.

```
/*
   Ordena de manera decreciente el segmento de A desde pos hasta pos + largo - 1.
   Precondiciones:
      - Los segmentos desde pos hasta pos + largo/2 - 1 y
      desde pos + largo/2 hasta pos + largo - 1
      estan ordenados de manera decreciente.
      - 0 <= pos <= n - largo - 2.
      - largo es multiplo de 2.
 */
void merge(int A[], int pos, int largo);
```

El siguiente es un ejemplo con $n = 12$, $k = 3$ y $h = 2$ del formato de A:

8 5 0 7 3 3 9 3 2 12 8 1

El resultado de una llamada a `merge(A, 6, 6)` debe ser:

8 5 0 7 3 3 12 9 8 3 2 1

Solución:

(I)
```
void ordenar(int A[], int n, int k, int h) {
  while (h >= 1) {
    for (int pos = 0; pos < n; pos += k*2)
      merge(A, pos, k*2);
    h = h-1;
    k = k*2;
    // n = 2^h * k = 2*2^(h-1) * k = 2^(h-1) * (k*2)
  }
}
```

El ciclo externo se ejecuta h veces. En el ciclo interno se hacen $n/(k \cdot 2)$ llamadas a `merge`, cada una de las cuales tiene costo $\Theta(k \cdot 2)$. Por lo tanto el costo es $\Theta(h \cdot n/(k \cdot 2) \cdot (k \cdot 2)) = \Theta(n \cdot h)$.

(II)
```
void merge(int A[], int pos, int largo) {
  int m = largo/2;
  // arreglos auxiliares en los que se copian los segmentos.
  int A1[m + 1], A2[m + 1];
  for (int j = 0; j < m; j++) {
    A1[j] = A[pos + j];
    A2[j] = A[pos + m + j];
  }
  // centinelas: A1[m] < A2[j] y A2[m] < A1[j], 0 <= j < m
```

```
  A1[m] = -1;
  A2[m] = -1;

  int i1 = 0, i2 = 0; // indices para recorrer A1 y A2
  for (int iA = pos; iA < pos + largo; iA++)
    if (A1[i1] >= A2[i2]) {
      A[iA] = A1[i1];
      i1++;
    } else {
      A[iA] = A1[i2];
      i2++;
    }
}
```

Un ejemplo en el que se da el peor caso es cuando los segmentos están ordenados de manera alternada: $A2[j] > A1[j] > A2[j+1]$, $0 \leq j < m-1$. Los primeros $m-1$ elementos de $A1$ participan en dos comparaciones y $A1[m-1]$ en 1. El total de comparaciones es

$$2(m-1)+1 = 2m-1 = largo-1 = \Theta(largo).$$