Segundo Parcial de Programación 3

28 de noviembre de 2016

- Este parcial dura **3** horas y consta de 3 carillas. El total de puntos es **60**.
- **NO** se puede utilizar ningún tipo de material de consulta. Salvo que se indique lo contrario podrá usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.
- Justifique sus respuestas.

Se requiere:

I) Numerar todas las hojas e incluir en cada una el **nombre, cédula de identidad, número de página y cantidad de hojas entregadas**.

II) Utilizar las hojas de **un solo lado** y escribir con lápiz.

III) Iniciar cada ejercicio en hoja nueva.

Ejercicio 1 (20 puntos)

Carla, una estudiante de Ingeniería, camina todos los días de su casa a la Facultad. En el trayecto pasa siempre primero por una panadería y luego por lo de su amigo Jorge. Además, no quiere pasar dos veces por la misma esquina. Para ello, quiere implementar un algoritmo mediante la técnica de *Backtracking* que le retorne el camino de menor distancia que cumpla con las condiciones requeridas.

Las esquinas del mapa se representan mediante enteros comprendidos en el intervalo $[1 \dots e]$, donde e es la cantidad de esquinas. Carla identifica a su casa mediante la constante C, a la Facultad mediante la constante F, a la panadería mediante la constante P y a la casa de Jorge mediante la constante J. Todas son de tipo entero y representan a una esquina válida.

Se dispone de una tabla de distancias D, tal que $D(i, j)$ representa el largo de la cuadra que va de la esquina i a la esquina j, o ∞ si no hay una cuadra que las conecte. Todas las distancias son positivas

(a) Defina la forma de la tupla solución para el problema, considerando que cada elemento representa a una esquina.

(b) Defina la función $distancia_j(t)$ que calcula la distancia recorrida por Carla en la tupla t, si se consideran las primeras j esquinas recorridas.

(c) Indique en lenguaje natural y formalmente: restricciones implícitas, restricciones explícitas, función objetivo y predicado de poda en caso de que corresponda.

Ejercicio 2 (20 puntos)

Sea $A = (A_1, \cdots, A_n)$ una secuencia de n enteros **diferentes**, mayores que 0. Una subsecuencia de A es una secuencia $(A_{h_1}, A_{h_2}, \cdots, A_{h_m})$ de elementos de A, con $1 \leq h_1 < h_2 < \cdots < h_m \leq n$. Una subsecuencia $(A_{h_1}, A_{h_2}, \cdots, A_{h_m})$ es creciente si $A_{h_1} < A_{h_2} < \cdots < A_{h_m}$. Se quiere calcular para cada i, $1 \leq i \leq n$, la longitud, denotada L_i, de la subsecuencia creciente más larga de A que termina en A_i.

Observación: Para cada i puede haber más de una subsecuencia creciente más larga de longitud L_i.

Ejemplo: Dado $A = (5, 2, 4, 7, 1, 6, 3)$, las longitudes de las subsecuencias crecientes más largas son: $1, 1, 2, 3, 1, 3, 2$. La subsecuencia más larga que termina en A_7 puede ser tanto $(2, 3)$ como $(1, 3)$. Las subsecuencias $(5, 6)$ y $(1, 6)$ son subsecuencias crecientes que terminan en A_6, pero $L_6 = 3$ porque también existe la subsecuencia creciente $(2, 4, 6)$, que se obtiene al incluir A_6 al final de la subsecuencia creciente más larga que termina en A_3 (lo cual es válido ya que $A_3 < A_6$).

(a) Llamamos P_i al problema de calcular L_i.

   I. ¿Qué condición se debe cumplir en A para que $L_i = 1$?

   II. Si $L_i \neq 1$, ¿cuáles son los subproblemas que se deben haber resuelto para resolver P_i?

(b) Escriba la recurrencia para calcular las longitudes L_i, para $1 \leq i \leq n$.

Nota: Para la notación se asume que el máximo de un conjunto vacío es 0.

(c) Escriba el seudocódigo que obtiene las longitudes $(L_1, \cdots, L_n)$ implementando la recurrencia de la parte b. La ejecución debe hacerse en tiempo $\Theta(n^2)$ y ocupar $\Theta(n)$ espacio.

Ejercicio 3 (20 puntos)

Un equipo de programación se propone diseñar un algoritmo que ordene un arreglo de enteros, A, de tamaño n. Uno de los programadores sugiere el siguiente algoritmo híbrido de ordenación, argumentando que puede resultar más eficiente para tamaños relativamente reducidos de n:

1. Dividir el vector de tamaño n en n/k subvectores de tamaño k cada uno, siendo k un parámetro a determinar. Se asume que n es múltiplo de k y que n/k es potencia de 2.
2. Ordenar cada uno de los n/k subvectores empleando el algoritmo *InsertionSort*.
3. Hacer, por etapas, una secuencia de mezclas (el *2-way-merge* del *mergesort*) de pares de subvectores ordenados del mismo largo. En la primera etapa se empieza con los n/k subvectores de longitud k. Luego, en cada etapa se hacen mezclas de subvectores del doble de tamaño de la etapa anterior hasta obtener un único vector ordenado de tamaño n.

Se quiere determinar el costo en el peor caso de este algoritmo. La operación básica con la que se establece el costo es la comparación entre elementos del arreglo. No se considera relevante el costo del paso 1. En cuanto al paso 3, se calcula que el costo (despreciando términos de menor orden) es $c_3 n \log(n/k) = c_3 n \log n - c_3 n \log k$ para alguna constante positiva c_3. Esto se argumenta en base a que hay $\log(n/k)$ etapas, cada una con un costo total n.

(a) Calcule el tiempo de ejecución del algoritmo híbrido completo (puede agregar alguna constante generada al calcular el costo del paso 2).

(b) ¿Para cuál de los siguientes valores de k el comportamiento asintótico de este algoritmo híbrido alcanza la cota inferior de los algoritmos de ordenación basados en comparaciones de pares de elementos?

I. $k = \log n$

II. $k = \sqrt{n}$

(c) Exhiba el árbol de decisión del *InsertionSort* (usado en el paso 2) para arreglos de tamaño $n = 3$.