Universidad Rey Juan Carlos

Prueba de Análisis de Algoritmos – Curso 2015-2016 – 4 de mayo de 2016

Diseño y Análisis de Algoritmos – Grado en Ingeniería Informática

Valor: 35 % de la nota final. Duración: **2 horas**

Se compensa el examen con 4 puntos de los 10 posibles

Ejercicio 1 ¿Dígito impar? [2 puntos]

Se pide diseñar e implementar un algoritmo recursivo en java para determinar si un número entero no negativo contiene un dígito impar. Además, se debe especificar:

(a) El tamaño del problema

(b) El/los casos base

(c) El/los subproblemas de menor tamaño similares al original que se usarán para construir los casos recursivos

(d) El/los casos recursivos, junto con un diagrama que indique el razonamiento seguido para hallar dichos casos recursivos

---

Ejercicio 2 Punto de inflexión [3 puntos]

Sea $\mathbf{v}$ un vector de n números enteros, el cual está organizado de manera que los números pares aparecen antes que los impares. Es decir, el índice de cualquier número par será menor que el índice de cualquier número impar. Por ejemplo:

$$\mathbf{v} = [2, -4, 10, 8, 0, \mathbf{12}, 9, 3, -15, 3, 1].$$

Se pide implementar un algoritmo recursivo eficiente en java, que se ejecute en $\Theta(\log n)$, para determinar el mayor de los índices de números pares. Es decir, aquel índice i para el cual v_i es par, pero donde todos los elementos v_j serán impares para todo $j > i$. Se considera que el índice del primer elemento es 0. Por tanto, en el ejemplo el resultado sería $i = 5$, ya que $v_5 = 12$ (par), mientras que $v_6 = 9$ (impar). Si el vector $\mathbf{v}$ no contiene números pares entonces la función recursiva devolverá -1.

Ejercicio 3 Camino de longitud mínima por un laberinto [1 punto]

Dado un laberinto, se desea encontrar un camino que no solo sea una solución, sino que además sea de longitud más corta. Este problema se puede resolver con la técnica de *backtracking*, pero en este ejercicio se pide una solución mediante un algoritmo **voraz**. La solución consiste en aplicar una estrategia de "transforma y vencerás", para poder aplicar uno de los algoritmos voraces vistos en clase.

Se pide describir cómo se puede resolver el problema concreto (cómo se transformaría el problema, y qué algoritmo voraz se aplicaría). No hay que escribir código. La descripción será muy breve: 5 líneas como máximo.

Ejercicio 4 Permutación optima [4 puntos]

En este problema se trata de implementar una función recursiva en java, basada en la técnica de *backtracking*, para hallar una permutación óptima de n elementos distintos (por ejemplo, enteros desde 0 hasta $n-1$), de acuerdo a una determinada función objetivo, que hay que minimizar. Dicha función no solo es capaz de devolver un valor para una permutación completa de n elementos, sino que también puede retornar el valor asociado a una permutación (solución) parcial de $m < n$ elementos. En concreto, la función es $f(m, \mathbf{p}, \mathbf{w})$, donde $m \leq n$, y evalúa únicamente los primeros m elementos de la permutación $\mathbf{p}$. Además, usa un vector de n pesos $\mathbf{w}$, que está fijado de antemano, pero se pasa igualmente como parámetro a la función. En este problema $f(m, \mathbf{p}, \mathbf{w}) \leq f(k, \mathbf{p}, \mathbf{w})$ si $m < k$ (es decir, la función objetivo no puede disminuir al ampliar una solución parcial).

La búsqueda debe tener en cuenta el mejor valor hallado hasta el momento para no proseguir en caso de que no pueda mejorarse una solución parcial con $m \leq n$ elementos. Se asume que $f(m, \mathbf{p}, \mathbf{w})$ ya está implementada, y se puede invocar mediante la instrucción `funcionObjetivo(m,p,w)`. Aunque varias permutaciones diferentes podrían conseguir el menor valor para f, solo se pide hallar una (cualquiera) de ellas. El valor máximo de $f(m, \mathbf{p}, \mathbf{w})$ será m^3. Finalmente, la función recursiva deberá recibir como parámetros de entrada toda la información necesaria para hallar la mejor permutación (no podrán usarse variables globales o no locales a la función).
