Tarea 2

Algoritmos y Complejidad

"Tiqui, tiqui, ti..."

Algorithm Knights

13 de septiembre de 2016

1. Se tienen dos secuencias de números reales $\langle a_1, a_2, \ldots, a_n \rangle$ y $\langle b_1, b_2, \ldots, b_n \rangle$. Se pide demostrar que se obtiene la suma máxima de los productos $a_i b_j$ sin repetir el par $i\,j$ si se ordenan ambas secuencias de menor a mayor, y se multiplican en ese orden. O sea, se obtiene el máximo con $a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ si $a_1 \leq a_2 \leq \cdots \leq a_n$ y $b_1 \leq b_2 \leq \cdots \leq b_n$. Nótese que este es esencialmente un algoritmo voraz.

   (35 puntos)

2. En el remoto país de Nadira, por ley se exige entregar vuelto en el mínimo número de billetes. Los billetes de Nadira vienen en denominaciones de 1, 4, 7, 13, 28, 52, 91 y 365.

   a) Demuestre que el algoritmo voraz de entregar lo más que se puede de la máxima denominación sin usar aún no da siempre el número óptimo de billetes. Note que Kozen y Zaks [1] demuestran que si las monedas son $1 = c_1 < c_2 < \cdots < c_m$, si el algoritmo voraz no da el óptimo hay un contraejemplo en el rango $c_3 + 1 < x < c_n + c_{m-1}$. Pearson [2] da un algoritmo eficiente para hallar un contraejemplo.

   b) Describa un algoritmo recursivo que calcule el número mínimo de billetes para entregar la suma x. No se preocupe de hacerlo eficiente, asegúrse de que sea correcto.

   c) Escriba un programa que entregue el número mínimo de billetes y cuántos de cada denominación deben darse para la cantidad x. Los datos de entrada son el número de casos de prueba, y las cantidades para cada uno de ellos. La salida debe constar exactamente de el número total de billetes, seguido en una línea aparte del número de billetes de cada denominación en orden creciente.

   (65 puntos)

1. Condiciones de entrega

- La tarea se realizará *individualmente* (esto es grupos de una persona), sin excepciones.
- La entrega debe realizarse vía Moodle en un *tarball* en el área designada al efecto, bajo el formato `tarea-2-`*`rol`*`.tar.gz` (`rol` con dígito verificador y sin guión).
  Dicho *tarball* debe contener dos directorios:
  - Un directorio `tarea`, que contenga las fuentes en LaTeX (al menos `tarea.tex`) de la parte escrita de su entrega, además de un archivo `tarea-2.pdf`, correspondiente a la compilación de esas fuentes.
  - Un directorio `programa`, que contenga su código *Python* con al menos un archivo `main.py` como punto de entrada y un `README` indicando la versión de *Python* utilizada y, opcionalmente, información adicional que deba saber el evaluador.
- Además de esto, la parte escrita de la tarea debe en hojas de tamaño carta en Secretaría Docente de Informática (Piso 1, edificio F3).
- Tanto el *tarball* como la entrega física deben realizarse el día indicado en Moodle.
  Por cada día de atraso se descontarán 20 puntos y a partir del tercer día de atraso no se reciben más tareas y la nota es automáticamente cero.

2. Revisión del código

Para evaluar el código se considerarán los siguientes aspectos:

Programa

- Uso adecuado de funciones 10 %
- Uso de estructuras de control 15 %
- Uso de estructuras de datos 10 %
- Código claro y simple 15 %
  - Nombres adecuados.
  - Indentación correcta.
  - Comentarios suficientes.
  - Ausencia de código comentado.
- Ejecución correcta 50 %
  - Ausencia de errores.
  - Resultados correctos en casos de prueba.

Referencias

[1] Dexter Kozen and Shmuel Zaks: *Optimal bounds for the change-making problem.* Theoretical Computer Science, 123(2):377–388, January 1994.

[2] David Pearson: *A polynomial-time algorithm for the change-making problem.* Operations Research Letters, 33(4):231–234, May 2005.