ANÁLISIS Y DISEÑO DE ALGORITMOS

Algoritmos voraces

Práctica 7 de laboratorio

Entrega: hasta el 12 de abril, 23:55h.
A través de Moodle

Suma máxima limitada (II)

Un comercio ha agraciado a uno de sus clientes con un premio. Consiste en el obsequio de todos los artículos que elija de su local comercial siempre que no repita ninguno y que el precio de venta de la selección no supere una cierta cantidad $T \in \mathbb{N}$. Si no cumple estas dos condiciones, pierde el premio. El cliente dispondrá de una relación con los n productos que podría seleccionar junto con sus precios de venta ($p_i \in \mathbb{N} - \{0\}$, $\forall\ i$). Se desea conocer cuál es la selección que obtendría un algoritmo voraz teniendo en cuenta que lo único que interesa es su valor total.

Por ejemplo, si se trata de decidir sobre $n = 8$ artículos cuyos precios son $\{11, 13, 19, 23, 29, 31, 41, 43\}$ y el umbral es $T = 99$, la mejor selección posible arroja un valor acumulado igual a 98 (corresponde, por ejemplo, a la selección de los cuatro objetos de valores 13, 19, 23 y 43. Sin embargo, esta solución ha sido obtenida mediante un algoritmo de programación dinámica. La estrategia voraz no es capaz de garantizar siempre la solución óptima ante este problema; obtiene en cambio, una aproximación cuya precisión depende del criterio escogido para la selección de objetos. En este ejemplo, si se seleccionaran los objetos siguiendo el orden en el que se han visto, la solución voraz escogería los cinco primeros, con precios 11, 13, 19, 23 y 29; y valor acumulado 95[1].

Por otra parte, aunque programación dinámica resuelve este problema, lo hace con complejidad temporal y espacial, respectivamente, $O(n \cdot T)$ y $O(T)$. Estas complejidades, aunque contenidas, podrían resultar impracticables con valores muy elevados de T.

En esta práctica se pide aplicar una estrategia voraz para encontrar una aproximación a la mejor selección posible de objetos. Se deja al criterio del alumno la elección del criterio de selección.

[1] En los ejemplos publicados se muestra otra solución voraz de esta instancia del problema cuyo valor acumulado se aproxima más al óptimo.

1. **Nombre del programa, opciones y sintaxis de la orden.**

   El programa a realizar se debe llamar `maxsum-greedy`. La orden tendrá la siguiente sintaxis:

   `maxsum-greedy -f` *fichero_entrada*

   El fichero con el problema a a resolver se suministra a través de la opción `-f`. En el caso de que no se suministre el fichero o se suministre uno inexistente se advertirá con un mensaje de error. No es necesario controlar posibles errores en el contenido del fichero de entrada ya que siempre se ajustará fielmente al formato establecido (véase siguiente apartado).

   Si se hace uso de la orden con una sintaxis distinta a la descrita se emitirá un mensaje de error advirtiéndolo y a continuación se mostrará la sintaxis correcta.

2. **Entrada de datos al programa. Formato del fichero de entrada.**

   De manera idéntica a la práctica anterior, el problema a resolver se suministrará codificado en un fichero de texto cuyo nombre se recogerá a través de la opción -f. Su formato y contenido será:

   - Línea 1 del fichero: Valores n y T, en este orden.
   - Línea 2: Valor de los n objetos: Una lista de n números naturales.

   Por tanto, el fichero contendrá 2 líneas que finalizarán con un salto de línea, salvo en todo caso, la última línea. El carácter separador entre números siempre será el espacio en blanco.

3. **Salida del programa. Ejemplo de ejecución.**

   El programa mostrará tres resultados. En primer lugar, la suma acumulada de los objetos seleccionados; en segundo lugar, la relación de sus valores separados (únicamente) mediante un espacio en blanco; en tercer lugar, el valor total calculado a partir de estos objetos.[2]. Cada resultado debe estar en una línea distinta que comenzará, respectivamente, con la etiqueta ‘‘`Greedy: `’’, ‘‘`Selection: `’’ y ‘‘`Selection value: `’’.

   A través de *Moodle* se puede descargar un archivo comprimido con varios ejemplos junto con sus soluciones.

   Téngase en cuenta que, dadas las características de los algoritmos voraces, la salida publicada no tiene por qué coincidir con la obtenida por el alumno en su implementación particular. Sin embargo, es imprescindible que sea una solución voraz, es decir, que el criterio de selección que se siga corresponda al de un algoritmo voraz.

```
$maxsum-greedy -f 1.problem
Greedy: 97
Selection: 13 41 43
Selection value: 97
```

[2]Este valor coincidirá con el obtenido con el algoritmo voraz pero servirá como comprobación de que la selección es compatible con la solución mostrada en dicho algoritmo.

Normas para la entrega.

ATENCIÓN: Estas normas son de obligado cumplimiento para que esta práctica sea evaluada.

a) Se debe entregar el código fuente y un *makefile* para obtener el ejecutable. No hay que entregar nada más, en ningún caso se entregarán ficheros de test.

b) Es imprescindible que no presente errores ni de compilación ni de interpretación (según corresponda). Para su corrección, la práctica se compilará con el estándar 11 de *g++* y con sistema operativo *GNU/Linux*. Se tratará de evitar también cualquier tipo de aviso (*warning*).

c) Todos los ficheros que se entregan deben contener el nombre del autor y su DNI (o NIE) en su primera línea (entre comentarios para que no afecte a la compilación).

d) Se comprimirán en un archivo `.tar.gz` cuyo nombre será el DNI del alumno, compuesto de 8 dígitos y una letra (o NIE, compuesto de una letra seguida de 7 dígitos y otra letra). Por ejemplo: `12345678A.tar.gz` o `X1234567A.tar.gz`. **Solo se admite este formato de compresión y solo es válida esta forma de nombrar el archivo.**

e) En el archivo comprimido **no deben existir subcarpetas**, es decir, al extraer sus archivos estos deben quedar guardados en la misma carpeta donde está el archivo que los contiene.

f) La práctica hay que subirla a *Moodle* respetando las fechas expuestas en el encabezado de este enunciado.