ANÁLISIS Y DISEÑO DE ALGORITMOS

Vuelta atrás

Práctica 8 de laboratorio

Entrega: hasta el 10 de mayo, 23:55h.
A través de Moodle

Suma máxima limitada (III)

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 mejor selección 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[1]).

Ya vimos en la práctica 6 que programación dinámica resuelve este problema con costes temporales y espaciales contenidos; sin embargo, aquellas soluciones podrían resultar impracticables con valores elevados de T. Por otra parte, no se conoce ninguna estrategia voraz que garantice la solución óptima (práctica 7). **En esta nueva práctica se pide aplicar una estrategia de vuelta atrás** para obtener la mejor selección posible de objetos.

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

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

   `maxsum-bt -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

[1] Podría ocurrir, como de hecho sucede en este ejemplo, que la solución no sea única, ni en los objetos seleccionados ni en el número de estos. En tal caso bastará con mostrar una alternativa cualquiera de entre todas las posibles.

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 cinco resultados:

   1. La suma acumulada de los valores de los objetos seleccionados, precedida de la etiqueta "`Backtracking: `".
      Por ejemplo: `Backtracking: 98`
   2. La relación de los valores de los objetos seleccionados; separados (únicamente) mediante un espacio en blanco, y precedida de la etiqueta "`Selection: `". El orden en el que se muestren es indiferente.[2]
      Por ejemplo: `Selection: 13 19 23 43`
   3. El número de nodos que el algoritmo de vuelta atrás ha necesitado expandir para llegar a la solución, precedido de la etiqueta "`Expanded nodes: `".[3]
      Por ejemplo: `Expanded nodes: 511`.
   4. El número de nodos descartados como consecuencia de los mecanismos de poda; con la etiqueta con la etiqueta "`Discarded nodes: `".
      Por ejemplo: `Discarded nodes: 67`.
   5. Tiempo de CPU (en milisegundos) requerido para encontrar la solución; con de la etiqueta "`CPU time (ms): `".[4]
      Por ejemplo: `CPU time (ms): 0.131`.

   Debe seguirse ese orden y cada resultado debe ir en una línea distinta.

[2]Esta relación puede no ser única. En tal caso da igual cuál se muestre.

[3]El número de nodos expandidos y descartados depende de cada implementación particular. Ahora bien, por regla general cuanto menor es el primero y mayor es el segundo, más eficiente es el algoritmo.

[4]Para obtener el tiempo CPU debe utilizarse la función *clock()* -véase práctica 1 o 2-

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

```
$maxsum-bt -f 1.problem
Backtracking: 98
Selection: 13 19 23 43
Expanded nodes: 511
Discarded nodes: 67
CPU time (ms): 0.131
```

4. **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.