PRACTICA I: EFICIENCIA DE LOS ALGORITMOS

Estructuras de datos, Grupo A

11 de octubre de 2012

1. Objetivo

El objetivo de estas sesiones es que el/la alumno/a comprenda la importancia de analizar la eficiencia de los algoritmos y se familiarice con las formas de llevarlo a cabo. Para ello se mostrará como realizar un estudio teórico y empírico de un algoritmo.

2. Cálculo del tiempo teórico

A partir de la expresión del algoritmo, se aplicarán las reglas conocidas para contar el número de operaciones que realiza un algoritmo. Este valor será expresado como una función $T(n)$ que dará el número de operaciones requeridas para un caso concreto del problema caracterizado por tener un tamaño n.

El análisis que nos interesa será el del peor caso. Así, tras obtener la expresión analítica de $T(n)$, calcularemos el orden de eficiencia del algoritmo empleando la notación $O(\cdot)$.

2.1. Ejemplo 1: Algoritmo de Ordenación Burbuja

Vamos a obtener la eficiencia teórica del algoritmo de ordenación burbuja. Para ello vamos a considerar el siguiente código que implementa la ordenación de

un vector, desde la posición inicial a final de éste, de enteros mediante el método burbuja.

```
1.  void burbuja(int T[], int inicial, int final)
2.  {
3.  int i, j;
4.  int aux;
5.  for (i = inicial; i <final - 1; i++)
6.   for (j = final - 1; j >i; j–)
7.    if (T[j] <T[j-1])
8.    {
9.     aux = T[j];
10.    T[j] = T[j-1];
11.    T[j-1] = aux;
12.   }
13.  }
```

La mayor parte del tiempo de ejecución se emplea en el cuerpo del bucle interno. Esta porción de código lo podemos acotar por una constante a. Por lo tanto las íineas de 7-12 se ejecutan exactamente un número de veces de (final-1)-(i+1)+1, es decir, final-i-1. A su vez el bucle interno se ejecuta una serie de veces indicado por el bucle externo. En definitiva tendríamos una fórmula como la siguiente:

$$\sum_{i=inicial}^{final-2} \sum_{j=i+1}^{final-1} a \tag{1}$$

Renombrando en la ecuación (1) $final$ como n e $inicial$ como 1, pasemos a resolver la siguiente ecuación:

$$\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} a \tag{2}$$

Realizando la sumatoria interior en (2) obtenemos:

$$\sum_{i=1}^{n-2} a(n-i-1) \tag{3}$$

Y finalmente tenemos:

$$\frac{a}{2}n^2 - \frac{3a}{2}n + a \tag{4}$$

Claramente $\frac{a}{2}n^2 - \frac{3a}{2}n + a \in O(n^2)$

Diremos por tanto que el método de ordenación es de orden $O(n^2)$ o cuadrático.

3. Cálculo de la eficiencia empírica

Se trata de llevar a cabo un estudio puramente empírico. Es decir, estudiar experimentalmente el comportamiento del algoritmo. Para ello mediremos los recursos empleados (tiempo) para cada tamaño dado de las entradas.

En el caso de los algoritmos de ordenación el tamaño viene dado por el número de componentes del vector a ordenar. En otro tipo de problemas, como es el caso del algoritmo para obtener el factorial de un número o la sucesión de fibonacci, la eficiencia dependerá del valor del entero.

Para obtener el tiempo empírico de una ejecución de un algoritmo lo que vamos a hacer es definir en el código dos variables como las siguientes :

clock_t tantes;

clock_t tdespues;

De esta forma, en la variable ***tantes*** capturamos el valor del reloj antes de la ejecución del algoritmo al que queremos medir el tiempo. La variable $tdespues$ contendrá el valor del reloj después de la ejecución del algoritmo en cuestión.

Así, si deseamos obtener el tiempo del algoritmo de ordenación burbuja tendremos que poner algo parecido a lo siguiente:

$tantes = clock()$; *//Captura el valor del reloj antes de la llamada a burbuja*
$burbuja(T, 0, n)$;*// Llama al algoritmo de ordenación burbuja*
$tdespues = clock()$;*//Captura el valor del reloj después de la ejecución de burbuja*

Para obtener el número de segundos simplemente escribiremos un mensaje como éste:
$cout << (double)(tdespues - tantes)/CLOCKS\_PER\_SEC << endl;$

Con esta instrucción lo que hacemos es obtener la diferencia entre los dos instantes y pasarlo a segundos mediantes la constante $CLOCKS\_PER\_SEC$. Para hacer uso de estas sentencias tenéis que usar el include $ctime$.

OBSERVACIóN: Para casos muy pequeños, el tiempo medido es muy pequeño, por lo que el resultado será 0 segundos. Estos tiempos tan pequeños se pueden medir de forma indirecta ejecutando la sentencia que nos interesa muchas veces y después dividiendo el tiempo total por el número de veces que se ha ejecutado. Por ejemplo:

```
#include <ctime>
...


clock_t tantes,tdespues;
double tiempo_transcurrido;
const int NUM_VECES=10000;
int i;
tantes=clock();
for (i=0; i<NUM_VECES;i++)
//Sentencia cuyo tiempo se pretende medir
tdespues = clock();
```

```
tiempo_transcurrido=((double)(tdespues-tantes)/(CLOCKS_PER_SEC*
(double)NUM_VECES));
```

Para obtener la eficiencia empírica deberemos de ejecutar el mismo algoritmo para diferentes ejemplos. Así para un algoritmo de ordenación lo ejecutaremos para diferentes tamaños del vector a ordenar y obtendremos el tiempo. Estos tiempos los almacenaremos en un fichero.

Podéis hacer uso de una macro, como la que se muestra a continuación, para obtener el tiempo empírico de un algoritmo.

```
#!/bin/csh -vx
@ i = 10
echo """>burbuja.dat
while ( $i <10000 )
echo " $i ‘burbuja $i‘" >>burbuja.dat
@ i += 100
end
```

Así en esta macro se va a escribir en el fichero ***burbuja.dat*** el tiempo en segundos que tarda el algoritmo de ordenación en ordenar vectores de 10 a 10000 elementos. Las muestras se han tomado de 100 en 100. Para poder ejecutar esta macro debeis de darle a ósta permisos de ejecución. Por ejemplo, mediante la siguiente sentencia:

```
chmod +x macro
```

y a continuación ejecutar la macro como

```
./macro
```

NOTA: Cuando redactóes un informe sobre la eficiencia empírica de un algoritmo deberán aparecer todos los detalles relevantes al proceso de medida: tipo de ordenador utilizado, compilador empleado, opciones de compilación, etc.

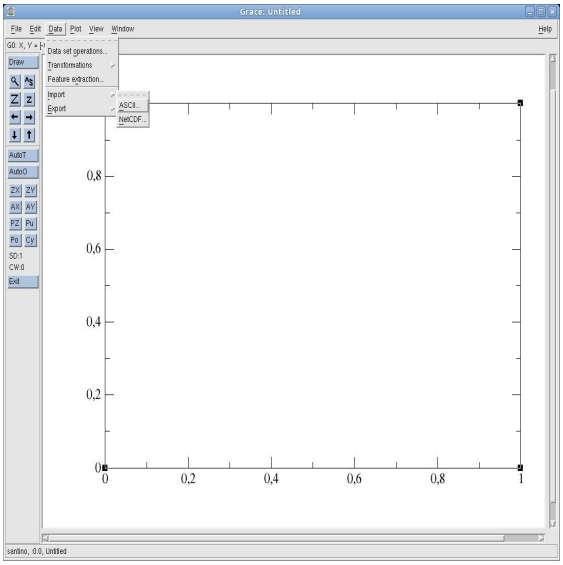

Figura 1: Inicio de xmgrace

3.1. Cómo mostrar los datos.

Para mostrar la eficiencia empóírica haremos uso de tablas que recojan el tiempo invertido para cada caso o bien podemos hacer uso de gráficas.

Para mostrar los datos en gráfica pondremos en el eje X el tamaño de los casos y en el eje Y el tiempo, medido en segundos, requerido por la implementación del algoritmo.

Para hacer esta representación de los datos podemos hacer uso del software grace (xmgrace). Este software trabaja sobre LINUX aunque ha sido exportado a plataformas como Windows 2000/NT/XP. Podéis encontrarlo en http://plasma-gate.weizmann.ac.il/Grace/.

XMGRACE.-

El programa se inicia mediante xmgrace. Al ejecutar el programa os debe aparecer una ventana como la que se muestra en la figura 1. (Todas las imógenes que se muestran a continuación fueron obtenidas con la versión de xmgrace 5.1).

A continuación debemos introducir el fichero donde tenemos almacenados los datos empíricos. Esto se puede ver como se realiza en las figuras 1 y 2.

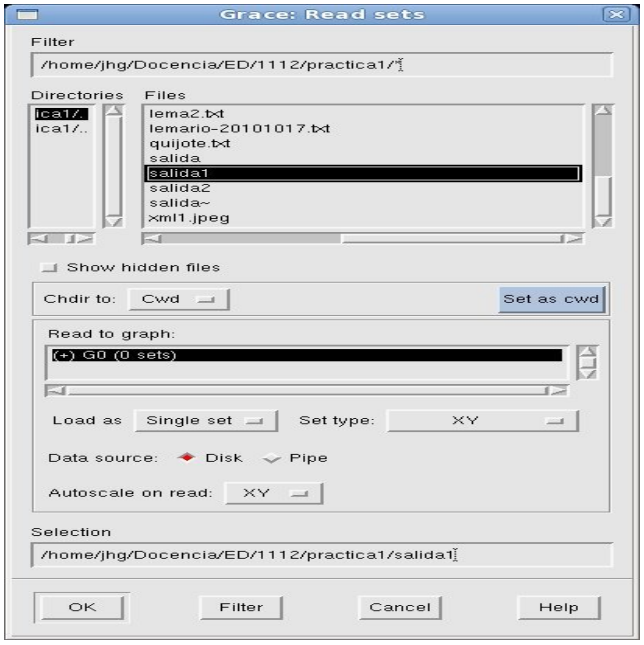

Figura 2: Seleccionar los datos

La gráfica se podrá representar como se muestra en la figura 3.

Para salvar el plot debemos irnos a File>Print Setup. A continuación le damos el nombre al fichero de salida y el formato de imagen en que queremos salvarlo. Finalmente debemos de pulsar FILE>Print

4. Cálculo de la eficiencia híbrida.

El cálculo teórico del tiempo de ejecución de un algoritmo nos da mucha información. Es suficiente para comparar dos algoritmos cuando los suponemos aplicados a casos de tamaño arbitrariamente grande. Sin embargo, cuando se va a aplicar el algoritmo en una situación concreta, es decir, especificadas la implementación, el compilador utilizado, el ordenador sobre el que se ejecuta, etc., nos interesa conocer de la forma más exacta posible la ecuación del tiempo.

Así, el cálculo teórico nos da la expresión general, pero asociada a cada término de esta expresión aparece una constante de valor desconocido. Para describir completamente la ecuación del tiempo, necesitamos conocer el valor de esas cons-

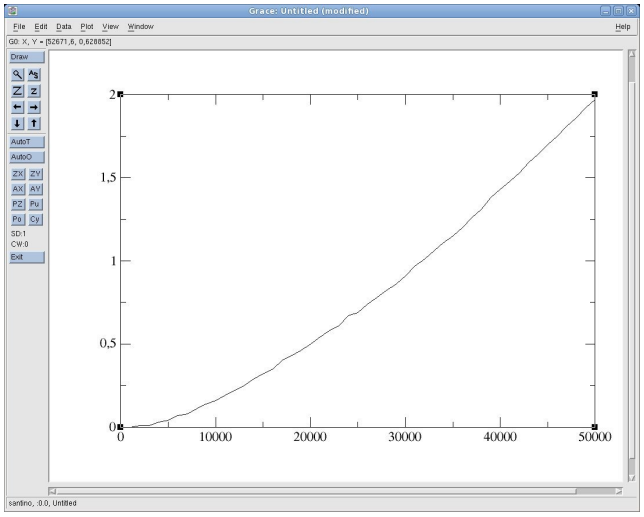

Figura 3: Grófica con los tiempo del algoritmo Burbuja

tantes. La forma de averiguar estos valores es ajustar la función a un conjunto de puntos.

En nuestro caso, la función es la que resulta del cólculo teórico, el conjunto de puntos lo forman los resultados del anólisis empóórico y para el ajuste emplearemos regresión por móónimos cuadrados. Por ejemplo en el algoritmo de ordenación la función que vamos a ajustar a los puntos obtenidos en el cálculo de la eficiencia empírica será:

$$T(n) = a_o * x^2 + a_1 * x + a_2$$

Al ajustar a los puntos obtenidos por mínimos cuadrados obtendremos los valores de a_0, a_1y a_2 es decir las constantes ocultas.

4.1. Cómo obtener las constantes ocultas

Para facilitar la labor del ajuste, emplearemos el programa xmgrace. Para ello nos iremos al menú que se muestra en la figura 4.

A continuación introduciremos la forma funcional de $T(n)$ (utilizaremos directamente la regresión cuadrática como indica la figura 5

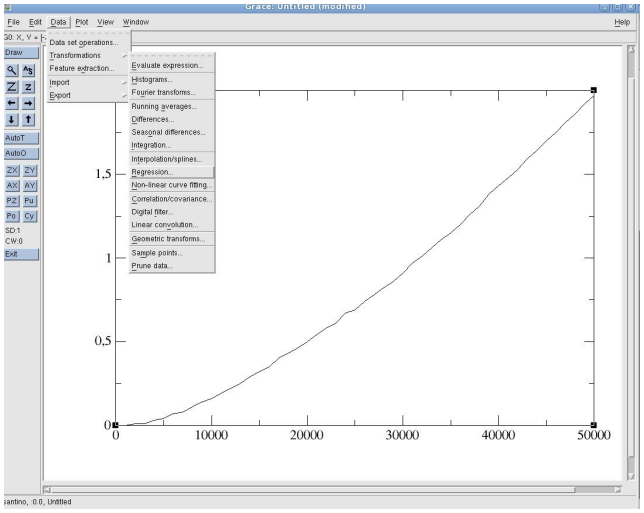

Figura 4: Cómo realizar el ajuste de las constantes ocultas

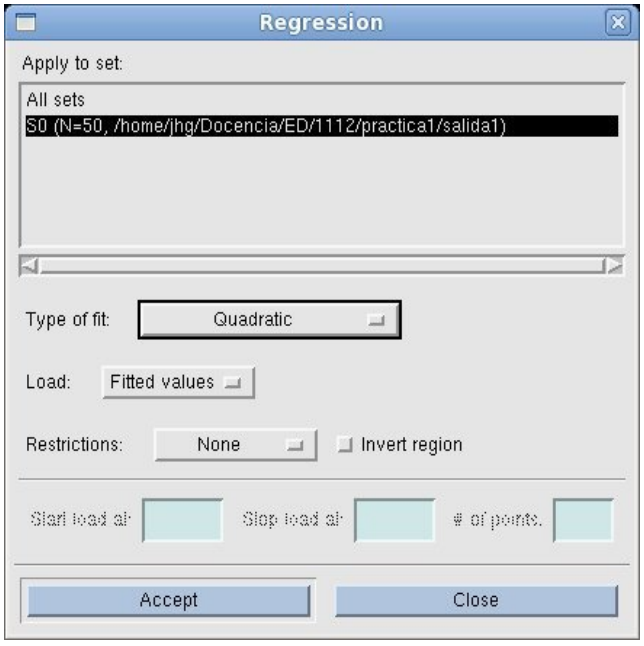

Figura 5: Introducir regresión.

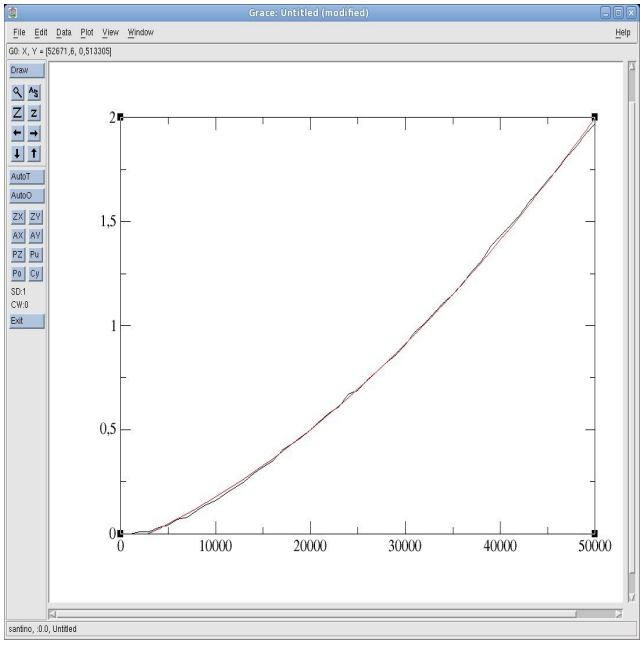

Figura 6: Grófica tiempo teórico vs. empórico.

A continuación le damos al botón *Accept y* apareceró una ventana que muestra que valores han adoptado las constantes ocultas además de obtener información de estadísticos como Chi-cuadrado, el coeficiente de correlación, el error cuadrático medio, entre otros. De entre estos estadísticos podemos quedarnos con el coeficiente de correlación. De tal forma que el coeficiente de correlación cuanto más se aproxime a 1 mejor ajuste representa.

Simultáneamente se representa la gráfica que se obtiene con $T(n)$ ya con los valores que hemos ajustado para las constantes ocultas como indica la figura 6.

5. Tareas a realizar.

En la página de la asignatura se encuentran los siguientes ficheros:

- lema.txt: Fichero con distintas palabras del castellano.
- quijote.txt: Un fichero de texto con el contenido del Quijote.

- xxxx.cpp: Ficheros de código con distintos algoritmos y ejemplos de cómo se puede calcular la eficiencia de los mismos.
  - Contar ocurrecias de una palabra
  - Algoritmo de ordenación burbuja
  - Contar frecuencias de aparición de una palabra en un documento. Para este último caso, se disponen de distintas versiones. La idea es que el alumno pueda comprender las ventajas que aporta el uso de distintas estructuras de datos (a nivel de eficiencia) para la resolución de un problema.

El alumno deberó ejecutar los distintos programas, generando como salida un fichero con los tiempos de ejecución de cada algoritmo, en función del tamaño de las entradas. Una vez que tengamos los tiempos se debe proceder al análisis híbrido de los mismos como se indica en la práctica.

5.1. Ejemplo 1: Algoritmo de ordenación por burbuja.

Pasos:

1. Compilar el fichero de código.

   ```
   g++ -o ordenacion ordenacion.cpp
   ```

2. Ejecutar el algoritmo de ordenación, recogiendo la salida del programa en un fichero llamado ordenacionB.datos. En cada iteración se ordena un subconjunto del diccionario de palabras, variando el tamaóo del subconjunto a ordenar.

   ```
   ./ordenacion > ordenacionB.dat
   ```

   Como resultado obtendremos un fichero con una salida parecida a

   ```
   100 0
   5100 0.53
   10100 2.12
   15100 4.72
   ```

```
20100 8.39
25100 13.11
30100 18.73
35100 25.4
....
```

donde la primera columna representa el tamaño del conjunto de palabras y la segunda el tiempo necesario para ordenarlas.

3. Realizar el análisis híbrido. En esta caso, podemos realizar una regresión cuadrática, que utilizando los datos del fichero `ordenB.eje.dat` nos da los siguientes valores.

   $T(n) = 2{,}081e^{-8}n^2 - 1{,}4924e^{-5}n + 0{,}15918$

5.2. Ejemplo 2: Algoritmo de ordenación sort de la STL.

La biblioteca STL proporciona una función de ordenación `sort` que tiene un orden de eficiencia `O(n log(n))`

Repite el mismo proceso que en el ejemplo 1 usando esta función y compara ambas soluciones.