UNIVERSIDAD DE GRANADA

Algorítmica

2º Grado en Ingeniería Informática

Práctica 1 – Cálculo empírico de eficiencia

Salvador Gutiérrez
Dpto. Ciencias de la Computación e I. A.
Universidad de Granada

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

Cálculo empírico de eficiencia de algoritmos

1. Objetivos

El objetivo de esta práctica es desarrollar las habilidades para analizar la eficiencia de algoritmos de forma práctica. Para ello, se presentarán diferentes algoritmos cuya implementación se proporciona en C/C++ y se obtendrá su tiempo de ejecución para diferentes casos. Además, se calculará el valor de la constante oculta para cada algoritmo y se comprobará gráficamente la relación entre el orden O(f(n)), calculado de forma teórica para cada algoritmo, y su tiempo de ejecución. Los siguientes apartados muestran cómo realizar cada paso que se solicita al alumno.

2. Análisis teórico del orden de algoritmos

Vamos a ver cómo llevar a cabo un estudio puramente empírico del comportamiento de los algoritmos analizados. 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 resolver el problema de las torres de Hanoi, el tamaño se corresponde con el valor del entero que representa el número de discos.

2.1. Cálculo del tiempo de ejecución

El siguiente código muestra cómo medir el tiempo de ejecución de un algoritmo de ordenamiento cualquiera para guardarlo a fichero:

1. Inclusión de bibliotecas. Se deben incluir las bibliotecas **iostream**, para gestionar ficheros, y **cstdlib** para poder utilizar el generador de números aleatorios y la función **clock** para medir el tiempo de ejecución del algoritmo:

```
#include <iostream>
#include <cstdlib> // Para usar srand y rand
#include <ctime> // Para usar clock
#include <fstream> // Para usar ficheros
using namespace std;
```

2. La función principal debe permitir que el programa tenga parámetros de entrada por línea de comandos, declarando **main** de la siguiente forma:

```
int main(int argc, char *argv[]) {
      return 0;
}
```

3. Las primeras líneas del código deben declarar las variables necesarias para poder representar el vector de a ordenar (en nuestro caso, supongamos que son números enteros), así como los tiempos inicial y final en los que comienza y termina el algoritmo, y el fichero de salida donde se guardarán los resultados:

```
  int *v;
  int n, i, argumento;
  unsigned long int tini, tfin; // Tiempos de ejecución inicial y final
  double tejecucion; // tiempo de ejecución del algoritmo en ms
  unsigned long int semilla; // Semilla para el generador de números aleatorios
  ofstream fsalida;
```

4. El programa tiene un número de parámetros de entrada variable. El primer parámetro (después del propio nombre del programa) debe ser el nombre del fichero de salida donde se guardarán los datos con el tamaño del caso del problema y el tiempo de ejecución del algoritmo para el mismo. Los parámetros restantes se corresponden con la semilla para el generador de números aleatorios y diferentes tamaños de casos que se deberán probar. Por tanto, las siguientes líneas del código deben comprobar que el número de parámetros del algoritmo es correcto, que el fichero de salida se abre correctamente y, por último, inicializar el generador de números aleatorios:

```
if (argc <= 3) {
      cerr<<"\nError: El programa se debe ejecutar de la siguiente forma.\n\n";
      cerr<<argv[0]<<" NombreFicheroSalida Semilla tamCaso1 tamCaso2 ... tamCasoN\n\n";
      return 0;
}

// Abrimos fichero de salida
fsalida.open(argv[1]);
if (!fsalida.is_open()) {
      cerr<<"Error: No se pudo abrir fichero para escritura "<<argv[1]<<"\n\n";
      return 0;
}

// Inicializamos generador de no. Aleatorios
semilla= atoi(argv[2]);
srand(semilla);
```

5. El siguiente fragmento de código debe recorrer todos los tamaños de caso de entrada al programa, generando un vector aleatorio de cada tamaño de caso y ejecutando el algoritmo con dicho vector de entrada. El tiempo actual del computador (en ticks de reloj) se puede obtener mediante la función `clock`. De este modo, si hacemos las variables `tini=clock();` y `tfin= clock();` antes y después de la ejecución del algoritmo, obtendremos los tiempos donde comienza a ejecutarse y termina la ejecución del algoritmo. Utilizando la diferencia `tfin-tini`, podemos calcular el tiempo en ms que tarda en ejecutarse el algoritmo, y que deberá ser guardada a fichero:

```
// Pasamos por cada tamaño de caso
for (argumento=3; argumento<argc; argumento++) {
    // Cogemos el tamaño del caso
    n = atoi(argv[argumento]);

    // Reservamos memoria para el vector
    v = new int[n];

    // Generamos vector aleatorio de prueba, con componentes entre 0 y n-1
    for (i= 0; i<n; i++)
        v[i]= rand()%n;

    tini = clock(); // Cogemos el tiempo en que comienza la ejecución del algoritmo
    AlgoritmoOrdenamiento(v, n); // Ejecutamos el algoritmo para tamaño de caso tam
    tfin = clock(); // Cogemos el tiempo en que finaliza la ejecución del algoritmo

    tejecucion= 1000*(tfin-tini)/(double)CLOCKS_PER_SEC;

    // Guardamos tam. de caso y t_ejecucion a fichero de salida
    fsalida << n << " " << tejecucion << "\n";

    // Lo mostramos tambien por pantalla
    cerr << "Tam. Caso: " << n << " T. Ejecucion: " << tejecucion << endl;

    // Liberamos memoria del vector
    delete [] v;
}
```

6. Por último, al finalizar el bucle anterior, debemos cerrar el fichero de salida de datos y salir del programa:

```
// Cerramos fichero de salida
 fsalida.close();

 return 0;
```

3. Cálculo de la constante oculta

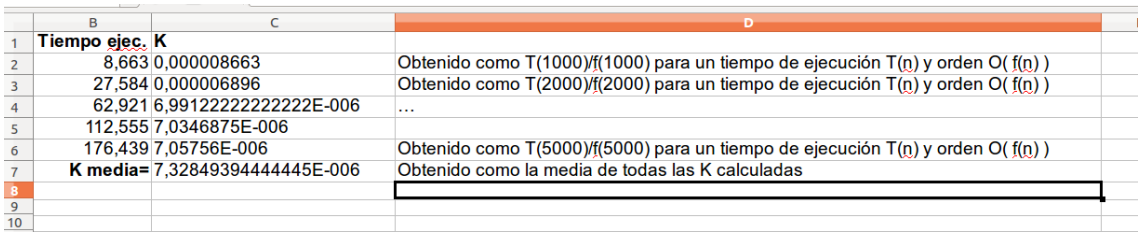
Teniendo calculado el orden O(f(n)) de cada algoritmo propuesto. Este orden O(f(n)) quiere decir que existe una constante **K**, para cada algoritmo, tal que el tiempo **T(n)** de ejecución del mismo para un tamaño de caso **n** es:

$$T(n) <= K*f(n)$$

El cálculo de la constante **K** se calcula despejando e igualando la fórmula anterior:

$$K= T(n)/f(n)$$

Este valor de K se calculará para todas las ejecuciones del mismo algoritmo, produciendo valores aproximados para K. Por tanto, calcularemos el valor final de K como la media de todos estos valores. La siguiente gráfica muestra un ejemplo del cálculo de este valor en LibreOffice Calc, para los resultados del algoritmo de ordenación por burbuja ejecutado para los casos 1000, 2000, 3000, 4000, 5000, utilizando una semilla de inicialización de números aleatorios igual a 123456.

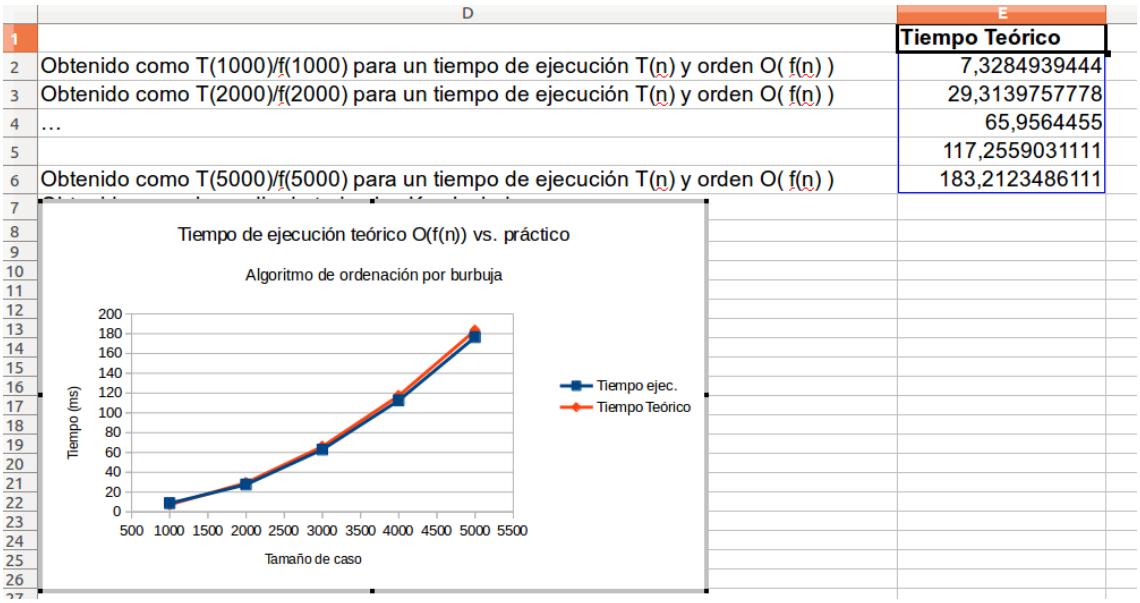


	B	C	D
1	**Tiempo ejec.**	**K**	
2	8,663	0,000008663	Obtenido como T(1000)/f(1000) para un tiempo de ejecución T(n) y orden O(f(n))
3	27,584	0,000006896	Obtenido como T(2000)/f(2000) para un tiempo de ejecución T(n) y orden O(f(n))
4	62,921	6,99122222222222E-006	…
5	112,555	7,0346875E-006	
6	176,439	7,05756E-006	Obtenido como T(5000)/f(5000) para un tiempo de ejecución T(n) y orden O(f(n))
7	**K media=**	7,32849394444445E-006	Obtenido como la media de todas las K calculadas
8			
9			
10			

NOTA: Para importar los datos de los ficheros de salida en LibreOffice Calc o Excel, es necesario reemplazar todos los puntos (.) por comas (,) en el fichero de datos. Depende de la configuración regional y de idioma de cada sistema.

4. Comparación gráfica de órdenes de eficiencia

Vamos a usar el algoritmo de ordenamento burbuja como ejemplo. Asumiendo que hemos calculado el orden de eficiencia del algoritmo como $O(n^2)$, y que el valor de la constante oculta en promedio vale K=7.33·10-6, a continuación, comprobaremos experimentalmente que el orden O(f(n)) calculado con una constante oculta superior siempre será mayor que el tiempo de ejecución real del algoritmo, por lo que hemos conseguido acotar el tiempo de ejecución del mismo y estimar, en el futuro, cuánto tardará en ejecutarse para otros tamaños de casos. Se muestra la siguiente gráfica generada por LibreOffice Calc para el tiempo de ejecución real y teórico con la constante oculta calculada:



5. Tareas a realizar

1. Calcula el tiempo de ejecución de cada uno de los algoritmos proporcionados y guárdelo a fichero para 10 tamaños diferentes de casos de cada problema. El tamaño de caso de cada alumno será diferente y dependerá de la potencia de cálculo de su PC, para evitar ejecuciones que requieran un excesivo tiempo de ejecución. No obstante, se recomienda comenzar con probar valores estándar como los siguientes: Para los problemas de ordenación: 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000. Para los problemas del cálculo de la sucesión de Fibonacci: 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40. Es importante modificar estos tamaños de casos si, para alguna ejecución, el alumno obtiene un tiempo de ejecución igual a 0.
2. Calcula el valor de las constantes ocultas de cada uno de los algoritmos.
3. Muestra una gráfica comparativa, para cada algoritmo, que compare la eficiencia teórica calculada con el tiempo de ejecución del mismo.

Todos los resultados se presentarán en una memoria sencilla pero bien escrita que se deberá entregar en formato PDF y debidamente formateada.