UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Práctica 1: Análisis de eficiencia de algoritmos

Doble Grado Ingeniería Informática y Matemáticas

Autores:
Jose Alberto Hoces Castro
Javier Gómez López
Moya Martín Castaño

Marzo 2022

Índice

1. Introducción

Esta primera práctica, **Práctica 1**, consiste en el análisis de eficiencia de algoritmos, consiste en tres partes distintas:

- **Análisis de la eficiencia teórica:** estudio de la complejidad teórica del algoritmos (Mejor caso, peor caso y caso promedio).
- **Análisis de la eficiencia empírica:** ejecución y medición de tiempos de ejecución de los algoritmos estudiados.
- **Análisis de la eficiencia híbrida:** obtención de las constantes ocultas

A continuación, se explican en más profundidad dichas partes.

1.1. Análisis de la eficiencia teórica

El análisis de la **eficiencia teórica** consiste en analizar el tiempo de ejecución de los algoritmos dados para encontrar el peor de los casos, es decir, en qué clase de funciones en notación $\mathcal{O}$ grande se encuentran. Para ello, hemos utilizado las técnicas de análisis de algoritmos vistas en clase y en la asignatura *Estructura de Computadores.*

1.2. Análisis de la eficiencia empírica

Para el análisis de la **eficiencia empírica**, hemos ejecutado los algortimos en cada uno de nuestros equipos bajo las mismas normas y condiciones, hemos medido el tiempo de ejecución de dichos algoritmos con la biblioteca `<chrono>`, basándonos en la siguiente estructura del código:

```
#include <chrono>
...

high_resolution_clock::time_point tantes, tdespues;
duration <double> transcurrido;
..

tantes = high_resolution_clock::now();
//Sentencia o programa a medir
tdepues = high_resolution_clock::now();
transcurrido = duration_cast<duration<double>>(tdespues-tantes);
```

Además, para automatizar el proceso de ejecución de los algortimos, hemos usado la siguiente estrucutra para generar nuestros scripts:

```
i = #valor de la primera iteracion

while [ $i -le #valor ultima iteracion ]
do
./programa_a_ejecutar $i >> salida.dat
i=$[i+#salto entre valores para conseguir 26 puntos]
done
```

Hemos ejecutado cada algoritmo 15 veces en cada uno de los tamaños que han sido probados, y hemos hecho la media de ellos para reducir perturbaciones que puedan ocurrir de manera aleatoria y que nos lleven al mejor o peor caso, obteniendo de esta forma casos promedio.

Cabe destacar que para *seleccion* e *insercion* hemos además ejecutado dos programas adicionales para obtener el mejor y peor caso de estos, pero este hecho lo detallaremos más adelante.

1.3. Análisis de la eficiencia híbrida

Para el análisi de la eficiencia híbrida, hemos tomado los datos de cada uno de los alumnos del grupo y hemos hallado la K(constante oculta). Para ello, hemos usado gnuplot.

Lo primero que hacemos es definar la función a la que queremos ajustar los datos. Tenemos que tener en cuenta el análisis teórico que hemos realizado previamente para saber cuál va a ser la forma de esta función. Podemos definir esta función en gnuplot mediante el siguiente comando (ejemplo para $\mathcal{O}(n^2)$):

```
gnuplot> f(x) = a0*x*x+a1*x+a2
```

El siguiente paso es indicarle a gnuplot que haga la regresión:

```
gnuplot> fit f(x) 'salida.dat' via a0,a1,a2
```

donde `'salida.dat'` es nuestro dataset.

La parte que más nos interesa es la parte donde pone `Final set of parameters`, pues ahí están nuestros coeficientes.

2. Desarrollo

A continuación, realizaremos el estudio individual de cada algortimo, como se ha descrito anteriormente.

2.1. Inserción

2.2. Selección

2.3. Quicksort

2.4. Heapsort

2.5. Floyd

```
void Floyd(int **M, int dim)
{
        for (int k = 0; k < dim; k++) //O(n)
          for (int i = 0; i < dim;i++) //O(n)
            for (int j = 0; j < dim;j++) //O(n)
              {
                int sum = M[i][k] + M[k][j];
               M[i][j] = (M[i][j] > sum) ? sum : M[i][j]; //O(1)
              }
}               //Total O(n^3)
```

2.5.1. Eficiencia teórica

Como podemos observar en los comentarios del código que hemos hecho en la función `void Floyd`, estamos ante una función que pertenece a $\mathcal{O}(n^3)$. Son tres bucles `for` que están anidados, cada uno $\mathcal{O}(n)$, por tanto, multiplicando los órdenes obtenemos que la función es $\mathcal{O}(n^3)$, es decir,

$$T(n) \in \mathcal{O}(n^3)$$

donde $T(n)$ es la función que expresa el tiempo de ejecución del algoritmo.

2.5.2. Eficiencia empírica

Tras ejecutar el algoritmo en un rango de 176 a 2000 elementos, con saltos de 76 unidades por ejecución, obtenemos los siguientes resultados:

Intel Core i7-6700 3.40 GHz	
Elementos (n)	**Tiempo (s)**
176	0.0244106
252	0.0721776
328	0.155828
404	0.288165
480	0.465947
556	0.724968
632	1.09236
708	1.54374
784	2.13392
860	2.67022
936	3.52897
1012	4.4074
1088	5.42559
1164	6.6698
1240	8.06967
1316	9.55022
1392	11.4197
1468	13.3942
1544	15.5
1620	18.0399
1696	20.5893
1772	23.6714
1848	26.7337
1924	30.1601
2000	33.9673

Ordenador Jota	
Elementos (n)	**Tiempo (s)**
176	0.0274773
252	0.0995705
328	0.20657
404	0.307902
480	0.51806
556	0.799187
632	1.16729
708	1.65895
784	2.42549
860	3.00331
936	3.84788
1012	4.84029
1088	5.97643
1164	7.78043
1240	9.08228
1316	10.7251
1392	12.9933
1468	14.6689
1544	17.2185
1620	20.2626
1696	22.9733
1772	26.0557
1848	30.2843
1924	33.4252
2000	38.5217

Ordenador Moya	
Elementos (n)	**Tiempo (s)**
176	0.038495
252	0.111472
328	0.244523
404	0.45528
480	0.761621
556	1.17395
632	1.73408
708	2.4355
784	3.29426
860	4.35444
936	5.64407
1012	7.16827
1088	8.91362
1164	10.9311
1240	13.2386
1316	15.8513
1392	18.7744
1468	21.9844
1544	25.5768
1620	29.5543
1696	33.8275
1772	38.5849
1848	43.8038
1924	49.4368
2000	55.3965

Cuadro 1: Experiencia empírica de algoritmo de Floyd sin optimizar

Observamos pequeñas diferencias, pero en general nada fuera de lo común. Estas diferencias son debidas a los distintos agentes tecnológicos usados para la realización del análisis de la eficiencia empírica en esta práctica.

2.5.3. Eficiencia híbrida

A través de la eficiencia híbrida, comprobaremos que el ajuste teórico realizado es correcto. Para realizar este análisi, tomamos los datasets de todos los integrantes del grupo.

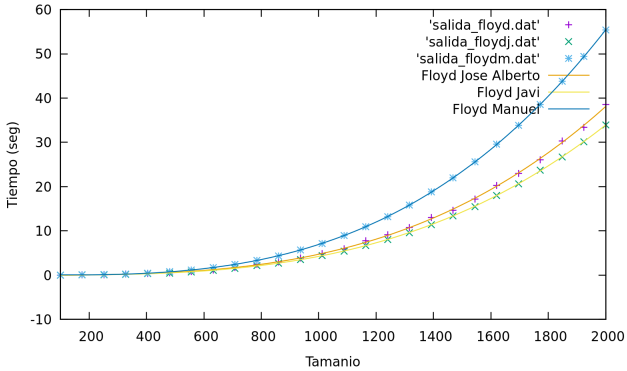

Figura 1: Gráfica con los tiempos de ejecución del algoritmo de Floyd

En esta gráfica están representados los 26 puntos obtenidos tras la ejecución del algoritmo de Floyd en los distintos equipos de los integrantes equipos. Tras una serie de cálculos con gnuplot, observamos que las constantes ocultas son:

- i7-6700 3.40Ghz $\rightarrow T_1(n) = 4{,}38237 \cdot 10^{-9}x^3 - 4{,}33753 \cdot 10^{-7}x^2 + 0{,}000337001x - 0{,}0504332$.
- Ordenador Jota $\rightarrow T_2(n) = 5{,}12922 \cdot 10^{-9}x^3 - 1{,}11315 \cdot 10^{-6}x^2 + 0{,}00083571x - 0{,}134397$.
- Ordenador Moya $\rightarrow T_3(n) = 6{,}77297 \cdot 10^{-9}x^3 + 5{,}13099 \cdot 10^{-7} - 0{,}000427834x + 0{,}0714028$.

A continuación, mostramos una gráfica que muestra otras posibilidades de ajuste para otros puntos, y se observa que el ajuste con una función cúbica es el mejor, confirmando así nuestro análisis teórico.

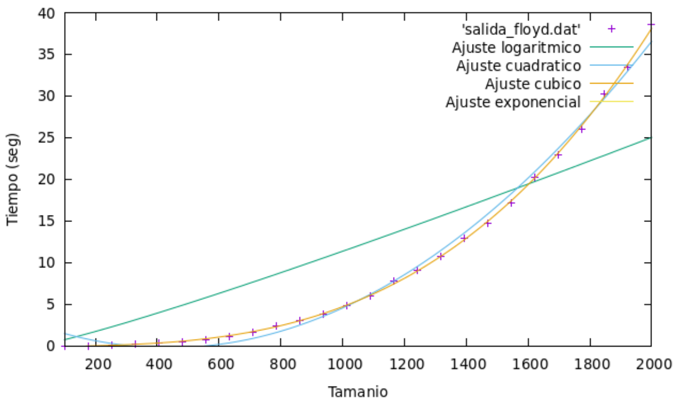

Figura 2: Bondad de ajuste cúbico Floyd

Podemos observar que nuestro análisis teórico es correcto. Además, podemos observarlo con el coeficiente de regresión para cada una de nuestra funciones de ajuste:

- $T_1(n) \longrightarrow R^2 = 0{,}00204522$
- $T_2(n) \longrightarrow R^2 = 0{,}044778$
- $T_3(n) \longrightarrow R^2 = 0{,}000855184$

Estos valores son muy cercanos a 0, y por tanto indican que el ajuste es muy bueno.

2.6. Hanoi