ALGORITMIA

Grado en Ingeniería Informática del Software - Curso 2025-2026

GUION DE LA PRÁCTICA 1.1 (directorio p11)

0)INTRODUCCIÓN

En la sesión anterior se estableció que en Java había dos formas de medir los tiempos de ejecución: "SIN OPTIMIZACIÓN" o "CON_OPTIMIZACIÓN". Luego se nos plantea el dilema: ¿cómo medimos los tiempos (SIN o CON)?

Tomaremos una decisión salomónica, en las primeras sesiones de prácticas (hasta la práctica3) se propondrá de forma prioritaria la toma de tiempos primando que, aunque sean mayores, sintonicen con la complejidad temporal ("SIN OPTIMIZACIÓN); para posteriormente medir los tiempos habitualmente buscando la obtención de tiempos menores ("CON OPTIMIZACIÓN").

1)CÓDIGO DE PARTIDA PARA ESTA PRÁCTICA

Vamos a trabajar en ese apartado con la clase **Vector1.java,** que se le proporciona dentro del paquete **p11.** Esta forma de trabajar *empaquetando* las clases tiene la ventaja de poder estructurar y utilizar la información mucho mejor, por lo que **será nuestra forma de trabajo** en lo sucesivo.

Si utilizamos la plataforma en línea de comandos (JDK), según lo ya visto: contiene hacemos:

```
javac *.java      //(compila todos los fuentes .java)
```

A continuación, podemos comprobar que **Vector1.class** aparece en ese mismo directorio, haciendo:

```
dir
```

Para ejecutar, **se puede hacer desde cualquier directorio** tecleando:

```
java -Xint  p11.Vector1  n
```

```
(n es un entero que dimensiona al vector o tamaño del problema)
```

Si utilizamos el entorno Eclipse, simplemente creamos un proyecto que llamaremos **prac01_Tiempos<UOpropio>** y arrastramos el paquete p11 a la carpeta *src*. Para compilar y ejecutar utilizamos la opción "**Run as…**" como indicamos anteriormente. Para añadir los argumentos en la ejecución hay que configurarlos en "**Run configurations…**".

2)TOMA DE TIEMPOS DE EJECUCIÓN

Se trata de la medición empírica del tiempo de ejecución de un programa, pudiendo así comprobar si coincide o no con el comportamiento teórico obtenido a partir del estudio analítico de su complejidad temporal.

Se utiliza para ello el método ***currentTimeMillis()*** de la clase ***System*** del paquete de Java ***java.lang*** (este paquete está ya cargado, sin necesidad de importarlo explícitamente mediante la sentencia *import*). Ese método ***currentTimeMillis()*** devuelve un entero de tipo ***long*** (64 bits), que es el milisegundo actual que *vive* el ordenador (la cuenta a 0 se puso hace más 50 años: 1 de enero de 1970) (para más información puede consultar la documentación JAVA). Llamando dos veces (al comienzo y al final del proceso a medir) a dicho método y restando los valores devueltos se obtiene el tiempo que pasó entre ambas llamadas (en milisegundos).

Teniendo en cuenta lo anterior:

SE LE PIDE:

¿calcular cuántos años más podremos seguir utilizando esta forma de contar? Explica el razonamiento seguido para realizar el cálculo.

Cuando en una toma de tiempos, efectuada como se explicó en el párrafo anterior, salgan unos pocos milisegundos (por poner un umbral, vamos a considerar un número menor a 50 milisegundos) no la vamos a tener en cuenta, por **falta de fiabilidad**. La razón es que hay procesos internos del sistema (p.ej. el llamado "recolector de basura"), que se ejecutan con mayor prioridad de ejecución que nuestro programa (de hecho, lo interrumpe, aunque no nos enteremos). Pues bien, esos procesos del sistema duran unos pocos milisegundos, que se suman a lo que tarda nuestro proceso a medir, falseando de forma sensible el tiempo obtenido en el caso de tiempos bajos.

Conclusión: Despreciaremos tiempos por debajo de 50 ms y mientras más grande sea el tiempo, más fiable (3578 es más fiable que 123).

A continuación, tomamos la clase **Vector2.java** que tiene como objetivo medir el tiempo de una operación (algoritmo) determinado, en concreto la suma de los n elementos de un vector, que tenemos implementada en la clase Vector1. Pasaremos un argumento en línea de comandos con el tamaño del vector como en el caso anterior.

Recordamos que, para ejecutarlo desde línea de comandos, desde el directorio pertinente:

```
java -Xint  p11.Vector2 n
```

SE LE PIDE:

¿Por qué a veces el tiempo medido sale 0?

¿A partir de qué tamaño de problema (n) empezamos a obtener tiempos fiables?

3)CRECIMIENTO DEL TAMAÑO DEL PROBLEMA

Realmente, es poco práctico ir variando el tamaño del problema "a mano", sobre todo si tenemos en cuenta que normalmente queremos dibujar una gráfica del tiempo en función del tamaño del problema para una determinada operación

¿Cómo podemos ir obteniendo los distintos valores temporales?: la clase proporcionada **Vector3.java** va incrementando el tamaño del vector y obteniendo tiempos, de esta manera se sigue de forma más conveniente la evolución del tiempo de ejecución.

```
java -Xint  p11.Vector3
```

Se observa que los tiempos son tan bajos que no podemos medirlos, por lo que vamos a dar un paso más para poder medir tiempos tan bajos como se precise en cada caso.

4)TOMA DE TIEMPOS PEQUEÑOS (<50 ms)

Cuando se verifica que un proceso a medir tarda poco tiempo, podemos ejecutar ese proceso repetidamente (*número de **repeticiones***), ajustando ese parámetro; ya que, por una parte, hay que lograr que el tiempo total de ejecución sea superior a esos 50 milisegundos y, por otra parte, que acabe en un tiempo razonable y no tengamos que estar esperando durante mucho tiempo.

Aunque ***repeticiones*** puede ser un valor cualquiera, es aconsejable y suficiente probar con valores que sean potencias de 10, porque la conversión de tiempos es tan fácil como aplicar la siguiente tabla:

Repeticiones	*Unidades de tiempo en:*	
1	**milisegundos**	**(10^{-3} s)**
10	décimas de msg	(10^{-4} s)
100	centésimas de msg	(10^{-5} s)
1 000	**microsegundos (µs)**	**(10^{-6} s)**
10 000	décimas de µs	(10^{-7} s)
100 000	centésimas de µs	(10^{-8} s)
1 000 000	**nanosegundos**	**(10^{-9} s)**
…….	……..	
10^9	**picosegundos**	**(10^{-12} s)**

Tabla 1. Tabla de conversión de unidades

La clase Vector4.java introduce la idea anterior, la ejecutamos:

```
java -Xint  p11.Vector4  1
java -Xint  p11.Vector4  10
java -Xint  p11.Vector4  100
java -Xint  p11.Vector4  1000
java -Xint  p11.Vector4  100000
java -Xint  p11.Vector4  10000000
```

(hasta el número de repeticiones que sea necesario en cada caso)

Los tiempos para lograr mediciones adecuadas se van alargando. Si por cualquier razón hubiera que abortar una ejecución, pulse Control+C. En Eclipse se puede utilizar el botón cuadrado rojo encima del panel *Console*.

SE LE PIDE:

¿Qué pasa con el tiempo si el tamaño del problema se multiplica por 2?

¿Qué pasa con el tiempo si el tamaño del problema se multiplica por otro k que no sea 2? (Pruebe, por ejemplo, para k=3 y k=4 y compruebe los tiempos obtenidos.)

¿Razone si los tiempos obtenidos son los que se esperaban de la complejidad lineal O(n)?

A partir de lo visto en Vector4.java midiendo los tiempos de suma, realice las tres siguientes Clases: Vector5.java para medir los tiempos del máximo, Vector6.java para medir los tiempos de coincidencias1 y Vector7.java para medir los tiempos de coincidencias2.

Con los tiempos obtenidos (en milisegundos) de las clases anteriores, rellene las dos tablas (recuerde lo visto en la práctica anterior: si para algún n tardara más de 60 segundos puede poner FdT ("Fuera de Tiempo")).

*TABLA1 (tiempos en **milisegundos** "SIN OPTIMIZACIÓN"):*

Pondremos "FdT" para tiempos superiores al minuto

n	*Tsuma*	*Tmaximo*
10000	…..	…..
20000	…..	…..
40000	…..	…..
80000	…..	…..
160000	…..	…..
320000	…..	…..
640000	…..	…..
1280000	…..	…..
2560000	…..	…..
5120000	…..	…..
10240000	…..	…..
20480000	…..	…..
40960000	…..	…..
81920000	…..	…..

*TABLA2 (tiempos en **milisegundos** "SIN OPTIMIZACIÓN"):*

Pondremos "FdT" para tiempos superiores al minuto

n	*Tcoincidencias1*	*Tcoincidencias2*
10000	…..	…..
20000	…..	…..
40000	…..	…..
80000	…..	…..
160000	…..	…..
320000	…..	…..
640000	…..	…..
1280000	…..	…..
2560000	…..	…..
5120000	…..	…..
10240000	…..	…..
20480000	…..	…..
40960000	…..	…..
81920000	…..	…..

Indique las características principales (procesador, memoria) del ordenador donde se han medido tiempos.

Una vez rellenadas ambas tablas, concluya si los tiempos obtenidos sintonizan con lo esperado, según la complejidad temporal computacional de las diferentes operaciones.

Se ha de entregar en un ***.pdf*** *el trabajo que se le pide y además las clases* ***.java*** *que ha programado. Todo ello lo pondrá en una carpeta, que es la que entregará comprimida en un fichero* ***p11ApellidosNombre.zip***.

La entrega de esta práctica se realizará, en tiempo y forma, según las indicaciones dadas por el profesor de prácticas.