GUION DE LA Práctica 0 (directorio p0)

0) INTRODUCCIÓN

Como es natural, cuando se resuelve un problema mediante un programa de ordenador interesa implementarlo de la mejor forma posible.

Los parámetros fundamentales que interesa optimizar son estos dos:

*** tiempo de ejecución**: tiempo (p.e. en milisegundos) que tarda el ordenador en resolver o ejecutar el programa que se diseñe para resolver el problema.

*** memoria ocupada**: memoria (p.e. Bytes) que ocupa el programa más la que ocupan las diferentes estructuras de datos que nos sirven para modelizar el problema.

Entre los dos factores anteriores (tiempo y memoria), en la práctica suele ser más interesante tender a minimizar el tiempo de ejecución; al menos mientras la memoria no se desborde por no caber en ella las estructuras de datos que utilice el programa realizado.

Pues bien, en esta sesión inicial vamos a ir viendo diferentes factores que influyen en el tiempo de ejecución y a través de un sencillo problema ejemplo se irá analizando en qué medida afectan al tiempo de ejecución.

***PROBLEMA EJEMPLO*:**

"calcular todos los números primos que hay entre 2 y un entero n dado"

1)FACTOR1: TAMAÑO DEL PROBLEMA

Mientras más grande sea el tamaño del problema, mayor será el tiempo que se tarde en resolverlo para cualquier algoritmo que se diseñe. Desafortunadamente, la mayoría de los problemas interesantes en la vida real tienen un tamaño considerable y no es posible disminuir esa magnitud del problema, sin que éste pierda su esencia.

Para el problema ejemplo antes propuesto, si nos retrotraemos a la asignatura Fundamentos de Informática del primer curso, está claro que procederíamos a abordar su análisis y codificación utilizando el lenguaje de programación Python.

Procediendo de forma incremental, se adjuntan los siguiente módulos Python:

***PythonA1v0.py**: se aborda la función **primoA1,** que calcula y devuelve de una determinada manera (algoritmo **A1**) si el entero que se le pasa, vía parámetro, es primo o no.

***PythonA1v1.py**: se aborda otra función **listadoPrimos**, que utilizando la función anterior acaba resolviendo el problema pedido para un **n** dado.

***PythonA1.py**: a través de este módulo pretendemos precisamente ver cómo el tiempo de ejecución va cambiando y aumentando a medida que el problema a resolver aumenta de tamaño (cuando **n** crece).

SE LE PIDE:

Haga una tabla en la que refleje los tiempos de ejecución (en milisegundos) del módulo PythonA1.py, para los valores de n expuestos (10000, 20000, 40000, 80000, 160000, 320000, 640000 y 1280000). Si para alguna n tardara más de 60 segundos, ponga FdT ("Fuera de Tiempo"), tanto en este apartado como en los apartados posteriores.

2)FACTOR2: POTENCIA DEL ORDENADOR

Es evidente que un mismo programa va a tardar en ejecutarse más o menos tiempo dependiendo de la potencia del ordenador en el que se ejecute. Esa potencia de un ordenador (trabajo o instrucciones que ejecuta por unidad de tiempo) depende de sus componentes, sin duda el más importante es la CPU que disponga, si bien en menor grado influyen también la cantidad memoria principal y secundaria, sistema operativo, programa de energía que se configure, …

Es obvio que si disponemos de acceso a varios ordenadores, para minimizar el tiempo nos interesa ejecutar cualquier programa en el ordenador más potente.

SE LE PIDE:

Haga una tabla en la que refleje, al menos para dos ordenadores a los que tenga acceso, los tiempos de ejecución (en milisegundos) del módulo PythonA1.py para los valores de n expuestos (10000, 20000, …, 640000 y 1280000). Referencie claramente para cada ordenador qué CPU es la existente y cantidad de Memoria principal RAM.

3)FACTOR3: ENTORNO DE IMPLEMENTACIÓN

Ahora vamos a comparar lo hecho antes en el entorno del intérprete Python con otro entorno de desarrollo, en concreto se va a resolver lo mismo en Java y así podremos comparar los tiempos obtenidos para ambos casos.

La conclusión trivial que al final sacaremos es: si para la resolución de un problema tenemos la posibilidad de desarrollarlo en varios entornos, para minimizar el tiempo nos interesa seleccionar el más eficiente.

De forma preliminar, se va a recordar cómo compilar y ejecutar programas en Java.

3.1) JAVA con LÍNEA DE COMANDOS (cmd)

La **ejecución básica de programas en Java** se hace desde la **ventana DOS**. Para lograr dicha ventana tiene que ejecutar el comando:

```
cmd
```

Cuando queramos cambiar de unidad sólo tenemos que poner:

```
x:                  // irá a la unidad x:
```

Cuando estemos en un directorio y nos queramos colocar en un hijo, haremos:

```
cd hijo
```

Para subir de un directorio al padre:

```
cd ..
```

Una forma de configurar el entorno es crear con un editor un **fichero de lotes (.bat)** con el PATH (apuntando al directorio bin de JDK) y el CLASSPATH (apuntando a los directorios de clases Java donde vayamos a trabajar) adecuados al ordenador concreto. Ese fichero de lotes se debe ejecutar cada vez que abramos una ventana DOS.

El comando **SET PATH** permite saber los PATH activos en ese momento.
El comando **SET CLASSPATH** permite saber los CLASSPATH activos en ese momento**.**

Si se cierra la ventana DOS desaparecen las rutas establecidas, por lo que cuando abramos otra nueva ventana DOS existe la necesidad de volver a establecer la ruta.

Cuando creamos un programa o clase JAVA lo *normal* es que forme parte de un paquete (conjunto de clases JAVA con finalidad común). La primera sentencia válida de la clase establece precisamente a qué paquete pertenece esa clase. Además, el directorio en el que se almacena esa clase debe coincidir exactamente con el nombre del paquete.

Para editar vale cualquier **editor de programas** (vale hasta el Bloc de Notas de Windows, en cuyo caso hay que grabar el programa con la **opción de "Todos los Archivos"** y la extensión .java, para que no se añada automáticamente la extensión **.txt**). En los laboratorios del centro también se encuentran instalados otros editores más potentes que proporcionan ventajas como el coloreado de sintaxis o completado de código. Algunos de los disponibles actualmente son: Notepad++, Visual Studio Code y Sublime text.

Tras ejecutar **cmd**, se procederá a la ejecución de fichero rutalaboratorio.bat (que ha de recoger "ad hoc" el PATH y el CLASSPATH pertinente) situándose en la carpeta que lo contiene:

rutalaboratorio

Posteriormente, en el directorio que contiene a **JavaA1v0.java**, se compilará:

javac JavaA1v0.java

Y se ejecutará desde cualquier directorio:

java p0.JavaA1v0 *// activado optimizador JIT*

O así:

java -Djava.compiler=NONE p0.JavaA1v0 *// desactivado optimizador JIT*

// ¡OJO! con las minúsculas y mayúsculas también en ***-Djava.compiler=NONE***

Java es un lenguaje de programación que data de 1996, pero que ha ido adaptándose y mejorando con el paso del tiempo, lo cual hace que sea uno de los lenguajes de programación más utilizados y potentes. Para compilar los programas, Java utiliza el compilador javac. Dicha herramienta convierte el programa en código máquina (también llamado bytecode). Además, Java cuenta con un componente interno que ha ido mejorando con el paso del tiempo: Java-In-Time Compiler (JIT). Este componente optimiza la ejecución de los programas consiguiendo mejoras de tiempo muy importantes. Por ejemplo, si el JIT se da cuenta que hemos implementado un bucle en nuestro código pero que ese bucle no cambia ninguna variable que vayamos a utilizar después en el mismo código, es muy posible que el JIT "elimine" ese bucle en tiempo de compilación y así optimice la ejecución del programa. Es muy importante tener esto en cuenta porque cuando midamos tiempos puede ser que en algunas ocasiones se produzcan mediciones muy inferiores a lo esperado. Eso significaría que el JIT ha encontrado código "optimizable" y lo ha cambiado para que se ejecute lo más rápido posible. Así, es muy recomendable utilizar el JIT, pero para evitar inconsistencias debidas a optimizaciones realizadas por el JIT en fragmentos de código que no hemos escrito adecuadamente, podemos desactivarlo y así comprobar que el código escrito siempre tiene la complejidad teórica esperada.

En lo sucesivo, denominaremos tiempos ***"SIN_OPTIMIZACIÓN"*** a los obtenidos con el JIT desactivado y tiempos ***"CON_OPTIMIZACIÓN"*** a los obtenidos con el JIT activado.

3.2)JAVA con ENTORNOS DE DESARROLLO INTEGRADOS (EDI o IDE)

Además de en forma básica (directamente con el JDK), se puede trabajar en JAVA con un IDE de los distintos que existen, que permiten integrar múltiples características dentro de un entorno gráfico.

En los laboratorios de prácticas disponemos del **entorno Eclipse** que está muy extendido tanto en entornos académicos como en profesionales. Este entorno de desarrollo integrado permite realizar desde una interfaz común todas las tareas relacionadas con el desarrollo: codificación, análisis de código y eliminación de errores, ejecución y depuración.

En este entorno la unidad mínima de desarrollo es el proyecto, así que debemos crear un nuevo proyecto para empezar a trabajar con el código. Cuando disponemos de código fuente de clases creadas previamente, sólo hace falta, *arrastrar y soltar* la carpeta correspondiente al paquete de esas clases Java a la carpeta *src* dentro del proyecto.

Una de las grandes ventajas del entorno Eclipse es que realiza una compilación automática al modificar y al grabar el código e informa al usuario de posibles errores mediante marcas, tanto subrayando el código implicado, como estableciendo un icono al margen, los cuales proporcionan información extra sobre el error o *warning* si nos situamos encima con el cursor.

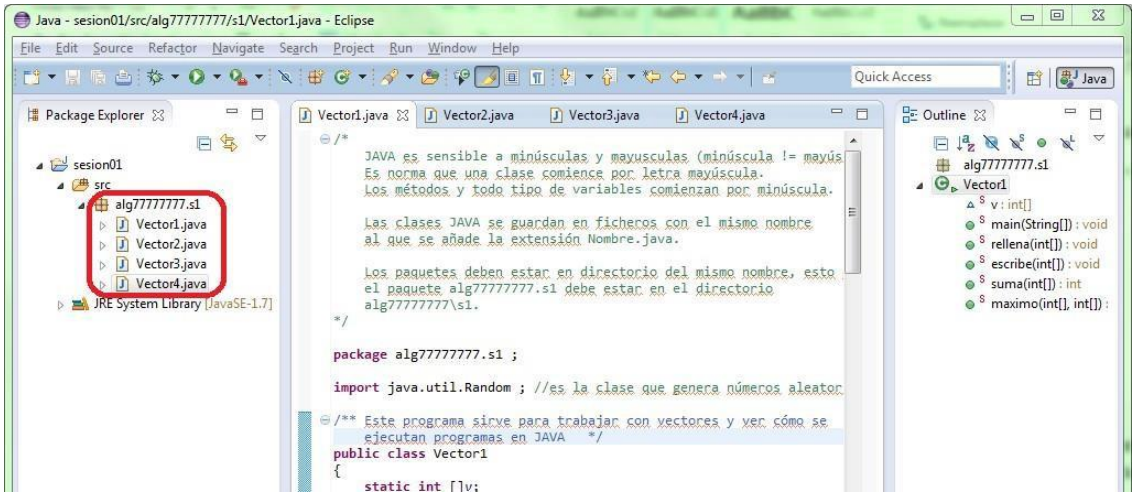

Figura : Espacio de trabajo de Eclipse

Para ejecutar y depurar debemos utilizar los iconos correspondientes en la barra de herramientas o bien el menú contextual sobre el proyecto y utilizar la opción "**Run as…**". Para realizar esta operación se deben guardar todos los cambios realizados en los ficheros del proyecto y el entorno se encargará de realizar las operaciones necesarias para ejecutarlo. Existen varios modos de ejecución, el que nos interesa es "**Java Application**" que ejecuta las clases Java sin ningún tipo de contenedor ni servidor. Además, Eclipse permite establecer los argumentos que acompañan la llamada a la clase principal para ejecutarla. Esto lo podemos hacer a través de la opción "**Run configurations…**" a la que podemos acceder bien desde el botón de ejecución de la barra de herramientas o bien desde el menú contextual. A partir de aquí deberemos seleccionar en el menú de la izquierda la clase relacionada y en las pestañas de la derecha la denominada "**Arguments**". Aquí podemos introducir los valores para los argumentos en el campo "Program arguments" separados por espacios.

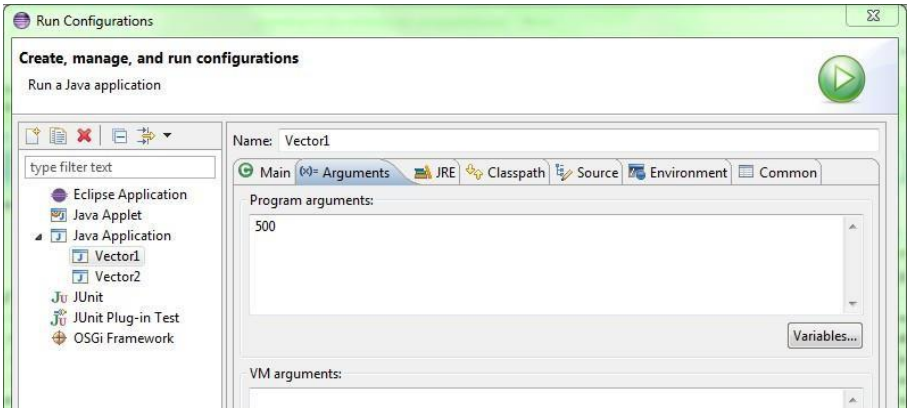

Figura: Ejecución de un programa con argumentos

SE LE PIDE:

Programe una clase de nombre JavaA1.java que calcule el problema de los números primos expuesto y que utilice el mismo algoritmo A1 para ver si un número es primo que utilizaba PythonA1.py.

Posteriormente, hay que hacer una tabla en la que refleje los tiempos de ejecución (en milisegundos) SIN_OPTIMIZACIÓN de JavaA1.java, para los valores de n expuestos (10000, 20000, …, 640000 y 1280000).

Para finalizar, compare esos tiempos con los obtenidos en Python (en un apartado anterior) para ese mismo algoritmo A1.

4)FACTOR4: ALGORITMO EMPLEADO

El tiempo de ejecución va a depender críticamente del algoritmo y las estructuras de datos empleadas en resolver el problema. Este factor es determinante y es el objetivo básico de esta asignatura (Algoritmia).

Frecuentemente, este factor incide más en la búsqueda de buenos tiempos de ejecución que los otros factores antes vistos y veremos en este curso ejemplos paradigmáticos de ello (por ejemplo, en el tema 2 dedicado a Ordenación).

Para ilustrar lo anterior, resolvemos nuestro problema de otras dos formas: Python:

***PythonA2.py**: se introduce la función **primoA2,** que calcula y devuelve de otra manera (algoritmo **A2**) si el entero que se le pasa, vía parámetro, es primo o no.

***PythonA3.py**: se introduce la función **primoA3,** que calcula y devuelve de una tercera manera (algoritmo **A3**) si el entero que se le pasa, vía parámetro, es primo o no.

SE LE PIDE:

Haga una tabla en la que refleje los tiempos de ejecución (en milisegundos) de los módulos PythonA1.py, PythonA2.py y PythonA3.py, para los valores de n expuestos (10000, 20000, …, 640000 y 1280000).

Implemente esos mismos algoritmos A2 y A3 en Java, en dos clases de nombres respectivamente JavaA2.java y JavaA3.java.

Realice una tabla en la que refleje los tiempos de ejecución (en milisegundos), ejecutando "SIN_OPTIMIZACIÓN", de las clases JavaA1.java, JavaA2.java y JavaA3.java, para los valores de n expuestos (10000, 20000, …, 640000 y 1280000).

Realice una tabla en la que refleje los tiempos de ejecución (en milisegundos), ejecutando "CON_OPTIMIZACIÓN", de las clases JavaA1.java, JavaA2.java y JavaA3.java, para los valores de n expuestos (10000, 20000, …, 640000 y 1280000).

Finalmente, razone las conclusiones finales obtenidas comparando los tiempos antes obtenidos: con Python, con Java "SIN_OPTIMIZACIÓN y con Java "CON_OPTIMIZACIÓN".

Si diseñara algún otro algoritmo A4 que resuelva el mismo problema sería bienvenido, sobre todo si mejora los tiempos del mejor de los tres algoritmos (A1, A2, A3) que aquí se le indican.

*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 **p0ApellidosNombre.zip***

*La entrega de esta práctica se realizará, en tiempo y forma, **junto a las de las dos semanas siguientes** (Práctica1.1 y Práctica1.2), en la tarea que se creará a tal efecto en el Campus.*