Estructuras de Datos y Algoritmos

Práctica II - Curso 2022/23

Fight-Radar (II)

1. Objetivos

Los análisis realizados en la práctica anterior han revelado que el programa del servidor tardaría minutos en procesar los movimientos de un gran número de usuarios. Es imprescindible mejorar de forma drástica su eficiencia, y la única opción viable es la utilización de estructuras de datos adaptadas al problema.

En esta segunda práctica **no se van a modificar** ni el formato de los datos disponibles (campos que representan al usuario y a los movimientos, y el de sus ficheros asociados) ni la forma de trabajo de la aplicación (por ejemplo, los movimientos se siguen procesando uno a uno). Los únicos cambios posibles serán en el tipo de estructuras de datos que almacenan la información y en la posible introducción de estructuras auxiliares que permitan calcular o filtrar datos de forma más eficiente.

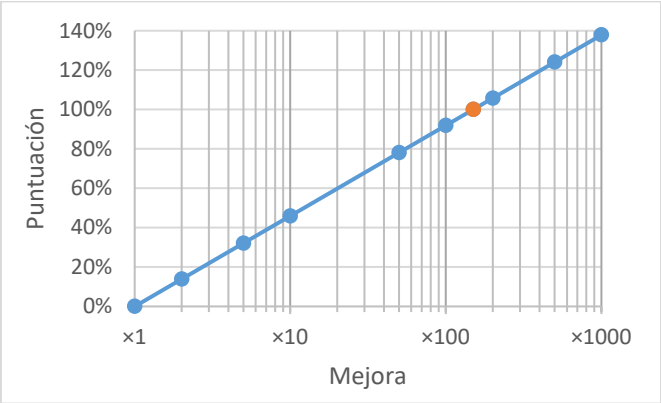
2. Evaluación de la ganancia en eficiencia

La evaluación de la ganancia en eficiencia se realizará de la siguiente manera:

- Para establecer el tiempo inicial que se debe mejorar, se debe ejecutar una solución de la primera práctica con un determinado caso de prueba. Para unificar resultados, en vez de usar vuestra solución se utilizará una solución de referencia, igual para todos.
- Para ello se ha depositado en el Campus Virtual de la asignatura dos soluciones (una en Java y la otra en Python) de la primera práctica, debéis descargaros la que corresponda al lenguaje de programación que vayáis a usar en la segunda práctica.
- En la página de la asignatura (https://www.infor.uva.es/~cvaca/asigs/eda.html) está disponible el caso de prueba que se debe usar, distinto para Java (n = 50.000) y Python (n = 14.000), ya que en Python la ejecución es más lenta.
- Debéis ejecutar el código de referencia con el caso de prueba, en modo depuración (se han incluido los ficheros de movimiento), y ese será el tiempo inicial, t_{ini}, que se debe mejorar. Dependiendo de lo rápido que sea vuestro ordenador tardará entre 100 y 700 segundos en ejecutarse.
- Las mejoras las podéis realizar usando vuestro código de la primera práctica o directamente sobre el código de referencia, como deseéis.
- Podéis haceros una idea de la ganancia en eficiencia midiendo el tiempo que tarda vuestro código mejorado con el caso de prueba de referencia, t_{fin}, y dividiendo el tiempo inicial por el obtenido: $mejora = t_{ini}/t_{fin}$. **El objetivo es obtener una mejora de al menos 150**: El nuevo código debe ser 150 veces más rápido que el antiguo. Por ejemplo, si el código antiguo tardaba 300 seg. el nuevo código debe tardar menos de 2 seg.

Atención: En la defensa de la práctica se utilizarán casos de prueba distintos a los proporcionados (aunque del mismo tamaño *n*)

La mejora en eficiencia es la parte más importante de la nota (habrá otros elementos que pueden influir en la calificación, como la estructuración del código o la calidad del informe), y para determinar el porcentaje de esta nota se utilizará la siguiente fórmula:



$$puntuación = 0.46 * \log_{10} mejora$$

Podéis observar que si el nuevo código tarda lo mismo (*mejora* = 1) no se consigue nota en este apartado. Si va el doble de rápido, se consigue un 14% de la nota, si va 5 veces más rápido el 32% de la nota y así hasta 150 veces más rápido (el punto rojo) donde se consigue el 100% de la nota. A partir de esa cota se consigue **nota extra**: Si va 1000 veces más rápido se consigue un 138% de la nota.

3. Estructuras de datos que se pueden usar

Se pueden usar directamente las estructuras disponibles en la librería **java.util** y los conjuntos y diccionarios de Python. Por supuesto, podéis (y debéis, si queréis que todo vaya bien) definir vuestras propias estructuras de datos en el código. El uso de cualquier otra librería (incluida **numpy**) esta sujeto a la aprobación de vuestro profesor de prácticas.

Haciendo spoiler os aviso de que al sustituir algunas listas por estructuras de datos más adecuadas (mapas, conjuntos, etc.) se consigue una mejora en la eficiencia, pero bastante modesta. Es necesario usar una estructura de datos auxiliar para conseguir mejoras relevantes. Hay varias opciones válidas para esa estructura de datos, pero ninguna de ellas está incluida en las librerías estándar de Java ni de Python, y por lo tanto debéis codificarla vosotros.

Se aconseja la utilización de *perfiladores de código* ***(profilers)*** para averiguar las partes de vuestro código que tardan más tiempo y por tanto son más prioritarias a la hora de buscar optimizaciones. En Java estas herramientas suelen estar incluidas en la mayoría de los entornos de programación, y en Python se puede realizar en modo consola con la siguiente línea de comando:

```
python -m cProfile programa.py
```

4. Sobre la no-uniformidad de la distribución de usuarios

En la práctica anterior los casos de prueba se generaban distribuyendo aleatoriamente a los *n* usuarios en un cuadrado de lado *l*, el cual se calculaba como $l = \sqrt{n/d}$ (siendo *d* la densidad, número de usuarios por metro cuadrado). De esa forma se garantizaba que la densidad fuera la indicada. Esto hace que los usuarios estén situados en un único grupo uniforme, como se muestra en el gráfico 1 (cada punto es un usuario en su posición):

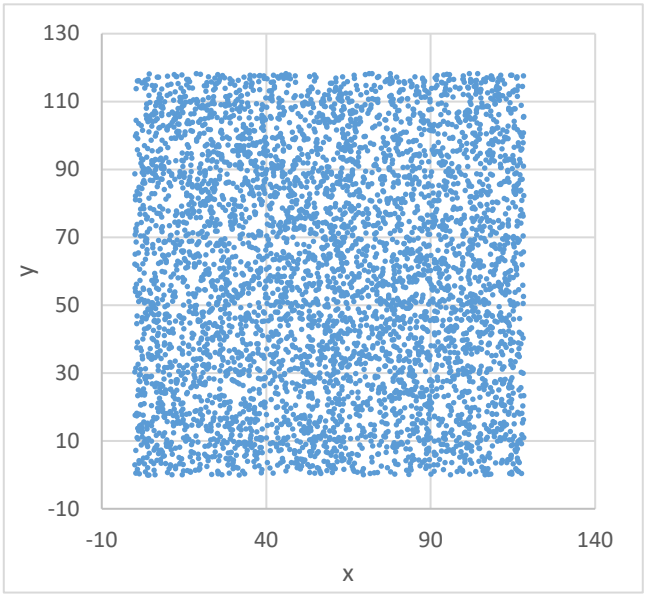


Gráfico 1

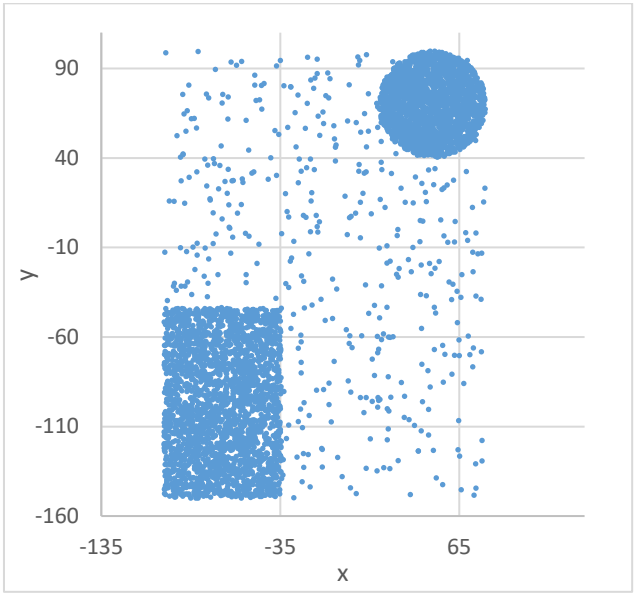


Gráfico 2

Sin embargo en la vida real lo más lógico es que los usuarios se encuentren concentrados en distintos grupos, de una forma mucho menos uniforme, tal como se muestra en el gráfico 2.

En la primera práctica esto no tenía relevancia ya que el algoritmo era insensible al hecho de que los usuarios estuvieran o no en un único grupo. En esta segunda práctica, sin embargo, el que la distribución de usuarios sea o no uniforme si que es relevante ya que existen estructuras de datos y/o algoritmos muy eficientes para el caso del gráfico 1 pero que no se pueden utilizar (o pierden su eficiencia) para el caso del gráfico 2. **En esta práctica vamos a suponer que la distribución puede no ser uniforme** (gráfico 2).

Redefiniremos el parámetro *d*, densidad de usuarios, para que en vez significar el número de usuarios dividido por el área del menor rectángulo que los contiene (eso significaría que el gráfico 2 sería mucho menos denso que el gráfico 1) ahora signifique la **longitud promedio de la lista de vecinos de los usuarios** (en este caso ambos gráficos tendrían aproximadamente la misma densidad). Fijaros que esta nueva definición es más conveniente desde el punto de vista de la medida de eficiencia del algoritmo.

5. Penalizaciones

Existe la posibilidad de crear algoritmos eficientes pero que requieran condiciones específicas para trabajar adecuadamente, por lo que se establecen las siguientes penalizaciones:

1. Si vuestro algoritmo solo puede funcionar correctamente o ser eficiente para distribuciones uniformes (caso del gráfico 1 del apartado anterior), tendrá una penalización del 75%: La nota se multiplicará por 0.75
2. Si vuestro algoritmo solo puede funcionar correctamente o ser eficiente para el caso de que los movimientos de los usuarios sean pequeños (es decir que los usuarios siempre permanezcan acotados en un determinada región), tendrá una penalización del 75%: La nota se multiplicará por 0.75
3. Para el caso de que se den a la vez las dos penalizaciones anteriores, la nota se multiplicará por $0.75^2 = 0.56$

6. Informe de la práctica

El informe de la práctica consistirá en un documento PDF (de 1 o 2 páginas) en el que se indiquen las estructuras de datos utilizadas y la mejora de eficiencia obtenida.

7. Presentación y Evaluación de la práctica

Las prácticas se realizarán individualmente o en parejas. En el caso de presentación en pareja es posible que la nota que reciba cada alumno sea diferente, según su desempeño individual en la defensa. Las prácticas constan de dos partes, el **código** del programa descrito (se puede codificar en Java o Python) y el **informe** de la práctica (en PDF).

Para una correcta evaluación de la práctica el alumno deberá:

1. Presentar electrónicamente (se creará una tarea en el Campus Virtual[1]), antes de la fecha límite un fichero comprimido (zip) que contenga el código fuente y el documento PDF con el informe. En el código fuente debe aparecer (como comentario) en las primeras líneas el nombre del alumno o los alumnos que han realizado la práctica.
2. Presentarse a la sesión de evaluación que le corresponda según su grupo de laboratorio en la semana establecida. En esta sesión se probará la aplicación, puede solicitarse **una modificación sencilla** del código y se obtendrán los resultados de su análisis de eficiencia.

Es en la defensa de la práctica donde se produce la evaluación, la presentación electrónica es simplemente un requisito previo para garantizar la equidad entre subgrupos y la comprobación preliminar de autoría.

Nota: El no poder realizar la modificación requerida puede dar lugar (según las circunstancias) a que se considere la práctica como no presentada por no haber podido acreditar su autoría.

[1] Si falla el Campus Virtual, se enviará por correo electrónico al profesor responsable de vuestro subgrupo