Estructuras de Datos y Algoritmos

Práctica I - Curso 2022/23

Fight-Radar (I)

1. Introducción

Enhorabuena!! Gracias al amiguete que trabaja en la Consejería de Cultura del Ayuntamiento de vuestra ciudad habéis conseguido que vuestra app para teléfono móvil, **Fight-Radar™**, se instale automáticamente para todo aquel que se haya descargado la aplicación de actividades de las fiestas de la ciudad. Estáis convencidos que al juntar una masa crítica de usuarios tendrá un éxito instantáneo y será descargada por millones de personas. Hay algo que os preocupa, sin embargo: En el diseño de la aplicación no se ha tenido en cuenta la eficiencia y no sabéis como va a responder el programa que controla las apps al pasar de 10 a 100.000 usuarios. Hay que tener en cuenta que como de momento sois pobres solo tenéis un servidor que ejecuta el único programa que se encarga de enviar datos a todos los móviles que tienen la app.

La idea de la aplicación **Fight-Radar** es muy sencilla, siendo su objetivo final el promover la amistad y el debate entre sus usuarios: Cada semana se plantea una pregunta de respuesta si/no a todos los usuarios (la misma pregunta para todos). La pregunta suele referirse a temas importantes de actualidad (por ejemplo: *¿Crees que Harry Styles escupió a Chris Pine?*) y su objetivo es poder dividir a los usuarios en dos **grupos** diferenciados. Luego utiliza geolocalización para detectar cuando dos usuarios pasan a estar a menos de 2 metros de distancia entre sí y en ese momento los móviles de ambos usuarios se iluminan y emiten sonidos según pertenezcan o no al mismo grupo (color verde y música melódica si son del mismo grupo y color rojo y música de batalla si son de distinto grupo). El círculo de radio 2 metros alrededor de cada usuario se denomina su **zona de control**.

La app que se ejecuta en los teléfonos móviles es muy austera, la única interacción con el usuario consiste en hacerle la pregunta de cada semana. También muestra una lista con los usuarios que están actualmente en su zona de control, coloreados según pertenezcan o no a su mismo grupo. Cada app tiene abierta de forma permanente una conexión con el servidor esperando que éste le envíe algún comando (se describirán posteriormente) y periódicamente (a intervalos prefijados de tiempo ***dt***, idéntico para todas las apps) envía la posición geográfica actual del usuario al servidor.

Es el programa que se ejecuta en el servidor al que vamos a prestar más atención. Este programa mantiene una estructura de datos con la información de los ***n*** usuarios que tienen activada la app, mantiene una conexión abierta con cada uno de ellos y cada vez que recibe un mensaje de nueva posición de un usuario calcula el conjunto de usuarios en su zona de control y si detecta que ese conjunto es distinto del anterior (han entrado o salido usuarios de la zona de control) envía los mensajes apropiados.

El servidor tiene 3 programas que se ejecutan en paralelo: El primero de ellos va recibiendo las actualizaciones de posición de los usuarios y las añade a una **cola de entrada** para que el programa principal las procese una tras otra, en secuencia. El programa principal consiste, por tanto, en un bucle donde en cada iteración extrae una actualización de la cola y la procesa (calculando variaciones en la zona de control de ese usuario), generando (si es el caso) los mensajes adecuados, que se añaden a una **cola de salida**. El tercer programa va extrayendo mensajes de la cola de salida y se los envía al app del usuario correspondiente. Solo nos vamos a interesar por el programa principal.

Como cada intervalo de tiempo ***dt*** el servidor va a recibir ***n*** mensajes de actualización de posición (uno por cada usuario activo), esta claro que el programa principal debe ser capaz de procesar ***n*** mensajes en un tiempo menor que ***dt***, en caso contrario la cola de entrada crecería continuamente hasta que se agotara la memoria disponible.

La principal tarea de esta primera práctica es averiguar cual el valor adecuado para ***dt*** en función del número de usuarios activos, ***n***, y de lo agrupados que estén los usuarios, ya que cuanto más agrupados estén (por ejemplo, en un concierto) mayor será el número de usuarios en cada zona de control y eso puede afectar a la complejidad del algoritmo.

Supondremos que para que la aplicación proporcione una experiencia adecuada al usuario ***dt*** (que representa la periodicidad con que toda app actualiza su información) no debería ser mayor que un segundo.

2. Descripción Técnica

Nuestros objetivos son (1) crear una **réplica** que simule al programa principal del servidor y (2) realizar medidas de su tiempo de ejecución en función del número de usuarios activos y su densidad.

No vamos a tener en cuenta el código de las apps ni las partes del programa del servidor que se encargan de las comunicaciones de entrada y salida (recepción y envío de los mensajes y gestión de las colas), supondremos que se ejecutan correctamente, que ningún usuario sufre pérdidas o retraso de comunicación, y que su impacto en la eficiencia es despreciable. Tampoco nos vamos a preocupar de la gestión de altas y bajas de usuarios, supondremos que su número es constante.

La estructura de datos principal de la réplica es un array o lista **no ordenada** con la información actual de los ***n*** usuarios. Cada usuario está representado por los siguientes campos:

Campo(s)	Tipo de datos	Descripción
id	String	Nombre elegido por el usuario. Este nombre es único, no se permite que dos usuarios tengan el mismo nombre.
x, y	Número real	Coordenadas de la posición actual del usuario[1]
grupo	Booleano	True si el usuario ha respondido "si" a la pregunta semanal y False en caso contrario
vecinos	Lista de strings	Lista **no ordenada** de los nombres de aquellos usuarios que en el momento actual están dentro de la zona de control de éste usuario (la lista no incluye al propio usuario).

Al comienzo de su ejecución, la réplica pedirá el nombre del fichero que indica el **estado inicial** de la prueba que se va a realizar. El contenido del fichero permitirá al programa crear y rellenar la lista de usuarios descrita en el párrafo anterior. Es un fichero de texto, de nombre "*inicial_**n**_**d**_.txt*" donde ***n*** indica el número de elementos y ***d*** la densidad redondeada a un decimal. Es decir, su nombre ya proporciona los valores de esos parámetros.

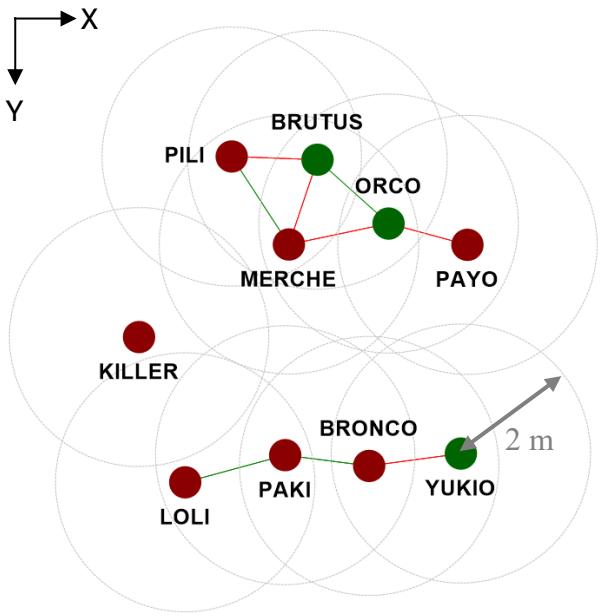
El contenido del fichero son *n* líneas de texto cada una de ellas almacenando la información de un usuario: su id, sus coordenadas, un carácter 'S' o 'N' indicando su grupo y la secuencia de

[1] Para simplificar se supone que los usuarios se disponen de forma bidimensional (no se tiene en cuenta si estan en edificios, etc.). Las coordenadas se dan en metros respecto a un determinado origen de coordenadas común a todos.

identificadores de sus vecinos. Todos estos valores están separados entre si por un espacio en blanco. A continuación podéis ver un ejemplo de estado inicial y el contenido del fichero que lo especifica:

```
LOLI 1.474 8.104 N PAKI
PAKI 3.016 7.684 N LOLI BRONCO
KILLER 0.762 5.861 N
MERCHE 3.073 4.432 N BRUTUS ORCO PILI
BRUTUS 3.514 3.122 S ORCO MERCHE PILI
ORCO 4.610 4.108 S PAYO BRUTUS MERCHE
PAYO 5.826 4.437 N ORCO
YUKIO 5.724 7.660 S BRONCO
BRONCO 4.310 7.851 N PAKI YUKIO
PILI 2.191 3.072 N MERCHE BRUTUS
```

Fichero **inicial_10_0.4_.txt**



En el gráfico los círculos pequeños representan los usuarios, su color el grupo al que pertenecen y se ha añadido un círculo gris alrededor de cada uno representando su zona de control. Las líneas unen los usuarios que son vecinos y su color indica si pertenecen o no al mismo grupo.

En este ejemplo la densidad es baja (0.3925 personas/m^2, redondeado en el nombre a 0.4), para facilitar la visualización. La densidad se calcula como el número de usuarios dividido por la superficie del menor rectángulo que los contiene, aunque vosotros no tenéis que preocuparos por su cálculo ya que se proporciona en el nombre del fichero. A la hora de generar los casos de prueba (se explica más adelante) es necesario probar con varios valores de densidad para analizar como depende el tiempo de ejecución de ese parámetro. Tomando como base lo que se explica en https://www.gkstill.com/Support/crowd-density/CrowdDensity-1.html, deberíamos evaluar casos de hasta **4** personas/m^2.

El siguiente paso es simular la recepción de los *n* mensajes de actualización de posición de los usuarios y medir el tiempo que se tarda en procesarles. El proceso actualiza las listas de vecinos de cada usuario y genera la lista de mensajes que se deben enviar a las apps. Estos mensajes son cadenas de texto con el siguiente formato:

- *Usuario1*+=*Usuario2*: Es un mensaje al app del *Usuario1* indicándole que el *Usuario2* ha entrado en su zona de control y que éste usuario es de su mismo grupo.
- *Usuario1*+/*Usuario2*: Es un mensaje al app del *Usuario1* indicándole que el *Usuario2* ha entrado en su zona de control y que éste usuario es de un grupo distinto.
- *Usuario1*-*Usuario2*: Es un mensaje al app del *Usuario1* indicándole que el *Usuario2*, que estaba en su zona de control, ha salido de ella.

Las actualizaciones de posición se tratan secuencialmente, en el orden proporcionado. Cada actualización consta del nombre de un usuario y su nueva posición *x* e *y*. Su proceso es el siguiente:

1. Se busca el usuario movido en la lista de usuarios.

2. Se recorre toda la lista de usuarios (salvo el movido) calculando su distancia con el movido, de forma que se pueda obtener una nueva lista de vecinos con aquellos que estén a menos de 2 metros.
3. Se compara la nueva lista de vecinos con la antigua, detectando los cambios. Por ejemplo si el usuario `ORCO` tenía como lista de vecinos `[BRUTUS, MERCHE, PAYO]` y al moverse su nueva lista de vecinos es `[PAYO, MERCHE, KILLER]` se deberán generar los siguientes mensajes:

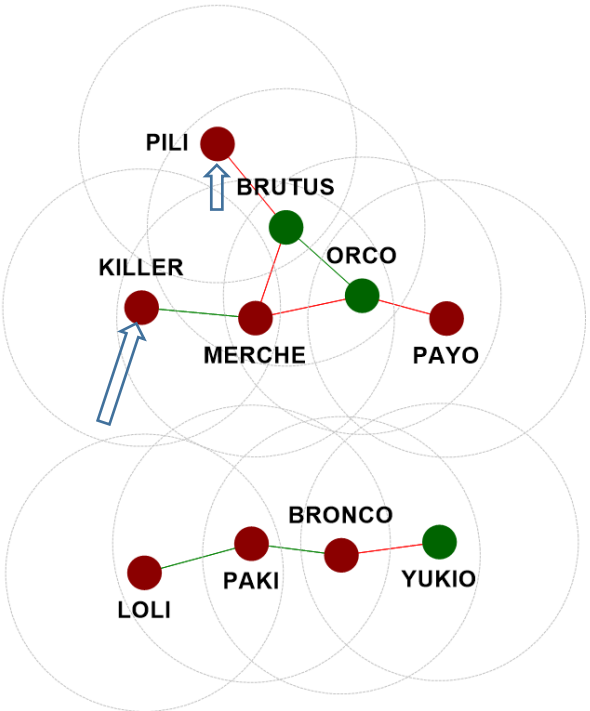
   `"ORCO+/KILLER", "KILLER+/ORCO", "ORCO-BRUTUS", "BRUTUS-ORCO"`
4. Se actualizan las listas de vecinos, no solo del usuario movido sino también las de aquellos usuarios para los que se han detectado cambios. En el ejemplo anterior, a `ORCO` se le asignaría la nueva lista `[PAYO, MERCHE, KILLER]`, al usuario `KILLER` se le añadiría `ORCO` a su lista de vecinos y al usuario `BRUTUS` se le eliminaría `ORCO` de su lista de vecinos.

De cara a la comprobación de vuestro programa, debéis tener en cuenta que dependiendo de la forma en que se implemente el paso 3 la lista de mensajes generados para cada usuario movido puede estar en un orden distinto al del fichero de comprobación (aunque deben ser los mismos mensajes). Para facilitar la comprobación visual, justo después de añadir los mensajes correspondientes a cada usuario movido se incluirá una línea "---".

A continuación se muestran los mensajes que se generarían dado el ejemplo anterior y suponiendo los siguientes movimientos (para simplificar solo cambian de posición dos usuarios, `KILLER` y `PILI`, en general todos los usuarios cambiarán su posición, aunque sea ligeramente):

```
KILLER 1.432 4.280
LOLI 1.474 8.104
PAKI 3.016 7.684
PILI 2.526 1.920
BRONCO 4.310 7.851
PAYO 5.826 4.437
BRUTUS 3.514 3.122
YUKIO 5.724 7.660
ORCO 4.610 4.108
MERCHE 3.073 4.432
```

Fichero/lista de movimientos




```
KILLER+=MERCHE
MERCHE+=KILLER
KILLER+=PILI
PILI+=KILLER
---
---
---
PILI-MERCHE
MERCHE-PILI
PILI-KILLER
KILLER-PILI
---
---
---
---
---
---
---
```

Mensajes generados

Un detalle a tener en cuenta es que el orden en que se procesan los movimientos (que debe ser en el que están en el fichero o lista) puede determinar que mensajes se generan. En el caso propuesto primero se procesa el movimiento de `KILLER`, que pasa a ser vecino de `PILI`. Luego, al procesar el movimiento de `PILI`, ésta deja de ser vecina de `KILLER`. Por eso en la lista de mensajes aparecen `KILLER+=PILI` y `KILLER-PILI` (y sus simétricos) lo que a primera vista podría parecer paradójico ya que si el movimiento de `PILI` hubiera sido anterior al de `KILLER` esos mensajes no aparecerían. De todas formas nosotros no vamos a preocuparnos de estas situaciones y los movimientos se van a procesar en el orden en que aparecen.

La réplica va a tener dos posibles modos de actuación, **depuración** y **medición**. Tras la lectura del fichero de estado inicial se pregunta al usuario en que modo se va a trabajar.

En el **modo de depuración** los movimientos se proporcionan en un fichero, cuyo nombre se pregunta al usuario. Se evalúan las actualizaciones de posición almacenando los mensajes en una lista de strings (esta es la parte en la que se mide el tiempo). Tras terminar la evaluación se escriben los mensajes por pantalla y el tiempo empleado. El objetivo es poder verificar, al comprobar la lista, que el programa se ha programado correctamente.

En el **modo de medición** los movimientos se calculan **cambiando la posición de cada usuario al azar**, añadiendo un valor aleatorio entre -0.5 y 0.5 a la coordenada *x* y a la coordenada *y* (distintos para cada coordenada, claro). No se pide un fichero sino que al usuario se le pregunta cuantas medidas quiere realizar. Para cada medida se generan nuevos movimientos al azar y se evalúa las actualizaciones, escribiendo por pantalla únicamente el tiempo que tarda cada una de ellas. El objetivo es poder obtener varios tiempos para un mismo valor de *n* y *d* para ver si varían poco o mucho entre ellos y usar su promedio en el análisis.

Como último ejemplo de éste apartado se muestra la salida del programa en ambos modos (se muestran en azul los datos introducidos por el usuario):

```
PRÁCTICA EDA 2022-23
Fichero de estado: inicial_10_0.4_.txt
Parámetros: n = 10, d = 0,4
Modo[D]epuración o [M]edición? D
Fichero de movimientos: movs_10_0.4_.txt
Procesando...
KILLER+=MERCHE
MERCHE+=KILLER
KILLER+=PILI
PILI+=KILLER
---
---
---
PILI-MERCHE
MERCHE-PILI
PILI-KILLER
KILLER-PILI
---
---
---
---
---
---
---
Tiempo: 0,000168 seg.
```

```
PRÁCTICA EDA 2022-23
Fichero de estado: inicial_10_0.4_.txt
Parámetros: n = 10, d = 0,4
Modo [D]epuración o [M]edición? M
Número de ciclos: 10
Procesando...
0,00043
0,00018
0,00016
0,00009
0,00013
0,00020
0,00011
0,00014
0,00013
0,00008
Promedio: 0,00017
```

Casos de Prueba

En la página auxiliar de la asignatura se ha creado un apartado para que podáis generar casos de prueba. Se introducen los valores de *n* (número de usuarios) y *d* (densidad de población) y se generarán 3 ficheros: El fichero de estado inicial, un fichero de movimientos y un fichero con la lista de mensajes que se debería obtener en el modo de depuración con esos movimientos[2].

[2] Aunque me repita quiero volver a avisar de que hay que tener cuidado al compararlos ya que los mensajes asociados a cada movimiento (separados entre sí por líneas ---) pueden estar en distinto orden en vuestra aplicación que en el fichero. Esto no supone un problema, pero hay que verificar que son exactamente los mismos mensajes en distinto orden.

3. Análisis de Eficiencia

Cuando estéis razonablemente seguros de que vuestro programa funciona correctamente el siguiente y último paso es obtener medidas para analizar la eficiencia del programa en función de *n* y *d*. Como son básicamente independientes entre sí, habría que generar varios casos de prueba con varios valores de *n* y *d* fijo, evaluar el tiempo y obtener la dependencia del tiempo respecto a *n*. Análogamente se generarán varios casos de prueba con distintos valores de *d* y *n* fijo, para obtener la dependencia del tiempo respecto a *d*.

Para la representación gráfica y obtención de estimaciones se aconseja usar programas tipo hoja de cálculo o similares. Aunque es posible usar scripts en R o programas avanzados como Statistica, Derive o Mathematica la tarea es suficientemente sencilla para que no sean necesarios y suelen dar problemas de sobreajuste a la hora de inferir las funciones que indican la complejidad.

Se pedirá un informe sencillo en formato PDF para la presentación de los resultados. Más adelante, si es necesario, se indicará un plantilla de como debería ser el informe.

4. Detalles

- En esta primera práctica las únicas estructuras de datos que se van a utilizar son **listas secuenciales no ordenadas** (las listas de Python o los arrays o la clase ArrayList de Java). Está prohibido usar otro tipo de estructuras, como conjuntos o diccionarios (en caso de duda pregunte al profesor). Hay que tener en cuenta que ahora nuestro objetivo es simplemente replicar y medir la implementación original, que no usa estructuras de datos ni algoritmos avanzados.
- Si usáis Java tened cuidado con la comparación entre Strings, para no tener problema debería hacerse siempre usando el método `equals`.
- Dados dos usuarios, u_1 y u_2, su distancia se calcularía como:

  $$d = \sqrt{(u_1.x - u_2.x)^2 + (u_1.y - u_2.y)^2}$$

- Podéis suponer que el formato de los ficheros inicial y de movimientos es correcto, no necesitáis realizar ninguna comprobación.
- Tanto el fichero inicial como en el de movimientos tienen *n* líneas, una por cada usuario, aunque es posible que el orden en que aparezcan los usuarios en ambos ficheros sea distinto.
- Aunque lo correcto sería medir el número de operaciones elementales, para simplificar la práctica se ha elegido medir el tiempo de ejecución en segundos. Los tiempos se miden contando únicamente el proceso de los datos, no hay que incluir lecturas o escrituras de fichero o escritura de mensajes en pantalla. En Java se puede usar `System.nanoTime()` y en Python `time.time()`.
- A la hora de obtener los resultados de eficiencia puede ser útil "hacer trampa" y realizar un análisis teórico previo para tener un spoiler de cuales deberían ser los resultados correctos. No es obligatorio realizarlo pero si lo incluís en el informe puede mejorar vuestra calificación (si es correcto, claro).
- En la evaluación de vuestro código se tendra en cuenta su correcta estructuración y que esté documentado adecuadamente.

5. 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** (en PDF) con el análisis de la eficiencia.

Para una correcta evaluación de la práctica el alumno deberá:

1. Presentar electrónicamente (se creará una tarea en el Campus Virtual[3]), 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, **se pedirá una modificación sencilla**, y se presentarán 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.

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