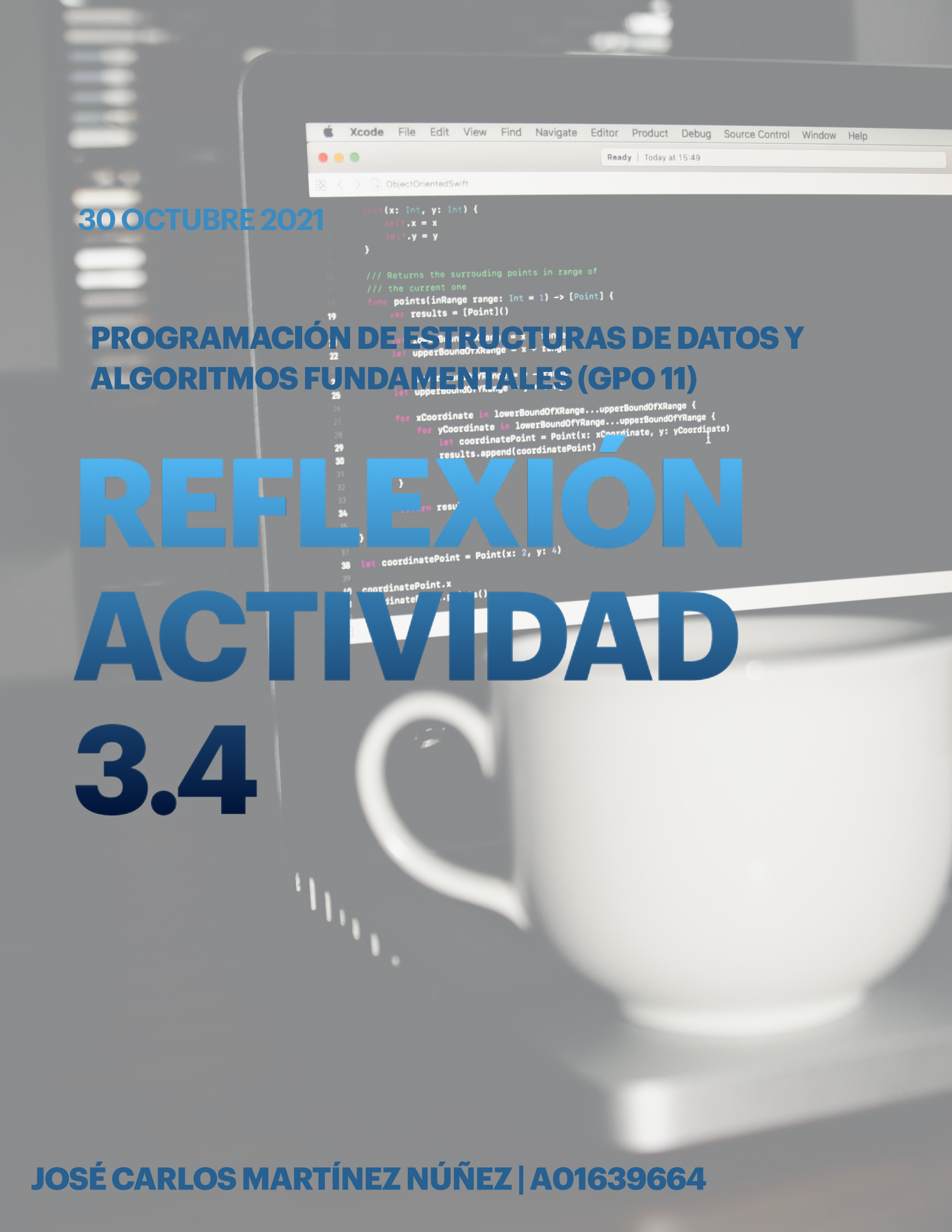
30 OCTUBRE 2021

PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS FUNDAMENTALES (GPO 11)

REFLEXIÓN ACTIVIDAD 3.4



JOSÉ CARLOS MARTÍNEZ NÚÑEZ | A01639664

Índice

PLANTEAMIENTO DEL PROBLEMA

El programa a realizar representa una bitácora con varias entradas. Cada entrada representa una falla en cierto sistema, nuestro programa debe contar las IP's únicas dentro de dichas entradas para determinar las que contienen el mayor número accesos.

Se nos dio un archivo de texto inicial con varias entradas de la bitácora, este archivo tiene el siguiente formato:

- **Mes** en formato de 3 letras.
- **Día**
- **Hora** en formato hh:mm:ss
- **Dirección IP** origen en formato ###.##.###.###:####
- **Razón** de la falla.

Ejemplo:

```
Aug 4 03:18:56 960.96.3.29:5268 Failed password for admin
```

ESTRUCTURA DE CLASES

Se tomaron en cuenta 4 clases para la realización del programa:

`IPBitacora`: representa una IP dentro de nuestra bitácora.

`IPNode`: representa un nodo de objetos `IPBitacora`.

`BitacoraAVL`: representa nuestro árbol AVL en el cual ordenamos nuestras IP's.

`Helper`: cuenta con los métodos de ordenamiento y otros métodos de utilidad que se utilizan a lo largo del programa.

PROCESO DE BÚSQUEDA

Primero que nada leemos el archivo "bitacora.txt" y creamos objetos `IPBitacora` por cada entrada. Como en el planteamiento del problema se nos menciona que solamente debemos desplegar las IP's con mayor número de accesos simplemente podemos enfocarnos en la IP e ignorar el resto de los datos dados por cada entrada.

Todos nuestros objetos `IPBitacora` son guardados dentro de un vector para comenzar el proceso de eliminación de duplicados y conteo de los accesos. Para esto utilizamos varios de los métodos de nuestra clase `Helper` principalmente el método de ordenamiento merge ($O\left(n \log n\right)$) y el método de `removeDuplicates()` una vez ordenamos nuestro vector eliminamos los duplicados comenzamos a agregarlos a nuestro árbol AVL.

Una vez tengamos todas las IP's con su número de accesos dentro de nuestro árbol basta con simplemente hacer un recorrido inorder comenzando por el sub-arbol derecho y esto nos dará de resultado las IP's con más accesos.

¿POR QUÉ UN ÁRBOL AVL?

Para que nuestro programa tenga un buen desempeño y para que al imprimir un recorrido inorder nuestras IP's se encuentren en el orden de número de accesos es vital que nuestro árbol se encuentre balanceado, de no ser así simplemente no funcionaría. Es por esto que aprovechamos la propiedad de los árboles AVL a diferencia de otras variantes de árboles binarios de búsqueda de autobalancearse.

CASOS DE PRUEBA

Ya que en el archivo "bitacora.txt" original no contamos con IP's duplicadas, es decir con más de un acceso, se tomaron varias al azar y se duplicaron las entradas.

```
Leemos bitácora... Completado en 2099155 microsegundos.
Ordenamos la bitácora... Completado en 158710 microsegundos.
Removemos duplicados... Completado en 38580 microsegundos.
Creamos árbol AVL... Completado en 11889 microsegundos.
11 - 484.5.450.83:5926

10 - 395.94.569.55:5230

9 - 94.38.339.86:5805

8 - 205.95.506.71:6482

7 - 907.61.915.80:6498

Buscamos las 5 IP's  con más accesos... Completado en 44 microsegundos.
Tiempo total: 2311909 microsegundos.
```

Como se puede observar en la captura nuestro programa encuentra las IP's con mayor número de accesos en un tiempo bastante corto, esto se da al uso de los árboles AVL. Además despliega utilizando la librería `<chrono>` el tiempo de ejecución de cada parte esencial del programa.