Documento de reflexión

Una vez más, en esta segunda actividad integradora podemos analizar y explorar el funcionamiento de diferentes algoritmos y estructuras en los que nos podemos apoyar para la mejor análisis y visualización de la información que tenemos en nuestra manos. De nuevo, el trabajo a realizar en esta actividad se trató sobre leer un archivo bitacora.txt en el que tenemos diferentes registros incluyendo una fecha, hora, IP y descripción los cuales nos encargamos de cargar a una estructura de datos llamada DoubleLinkedList. El objetivo de esto es más adelante implementar un método de ordenamiento para así poder crear un programa que nos dé la opción de buscar dos fechas determinadas y desplegar el todos los registros que se encuentran en ese rango. Esto haciendo uso de una binarySearch ya que hará que la búsqueda sea más práctica y eficiente.

En el caso de tus el algoritmo de ordenamiento, se implementó un quickSort. En la ocasión anterior, el algoritmo que se empleó funcionaba recursivamente ya que esa era, en su momento, la opción más rápida y eficiente pero en esta ocasión se hizo uso de una estructura extra, un Stack, para usarlo al momento de ordenar nuestra liga doblemente ligada. De ese modo pudimos hacer que nuestro método de sort fuera iterativo pero sin dejar de ser practicamente igual de eficiente que su implementacion recursiva. Teniendo una complejidad temporal de O(n log n). QuickSort realiza menos operaciones que otros algoritmos de ordenamientoya que este no necesariamente recorre siempre todo el arreglo gracias a las particiones que genera en cada iteración.

El método de búsqueda binarySearch tiene una complejidad de O(log n) empieza comparando el elemento del medio del arreglo con el valor que queremos encontrar. Si coincide y es igual, se regresa esa posición del arreglo. De lo contrario, se revisa si nuestro valora buscar es mayor o menor a ese elementomedio y dependiendo de eso, se empieza a iterar hacia delante o hacia atrás respectivamente, dejando así de lado a toda una mitad del arreglo fuera del algoritmo. Este algoritmo nos regresa la posición -1 en caso de que no se encuentre el valor que queremos buscar, de lo contrario se regresa una posición real del arreglo.

Despúes de encontrar nuestros valores enteros resultantes de nuestra binarySearch verificamos si una de las dos búsquedas realizadas regresó -1 como resultado (no se encontró el elemento). De ser así, se despliega un mensaje de error, de lo contrario, se imprimen las fechas en pantalla y se escruben en un archivo .txt el cual tenemos la posibilidad de cambiar el nombre sin problema. Podemos ver en el texto adjunto una muestra de cómo funcionaría esto:

```
if (res1 == -1 || res2 == -1){
        std::cout << "Las fechas o alguna de las fechas introducidas no han sido
encontradas en los registros.\nIntenta buscar esas fechas más adelante" <<
std::endl;
    } else {
        myLogList.printRange(res1, res2);
        myLogList.writeToNewTxt("test01.txt", res1, res2);
    }
```

Fue un poco complicado al principio tomar mi actividad anterior y adaptarla para que funcionara con las DoublyLinkedLists ya que al principio había cometido muchos errores. Mi código funcionaba correctamente acorde a los requerimientos básicos de la entrega pero, junto con mi profesor, pude darme cuenta que a la larga sería un poco complicado reutilizar mi código para adaptarlo. Es por ello por lo que tuve que cambiar gran parte de mi código pero al final pude lograrlo y hacer que todo funcionara correctamente. Al principio es un poco abstracto aprender el funcionamiento de diferentes estructuras de datos a la perfección pero me siento muy satisfecho ya que considero que fue una actividad, como su nombre lo dice, integral, retadora, completa y satisfactoria al final de todo.