Carlo Ángel Luján García

A01639847

Programación de estructuras de datos y algoritmos fundamentales (Gpo 13)

Profesor Luis Ricardo Peña Llamas

Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación TC1031 (Evidencia Competencia)

25/11/2021



***Reflexión sobre el desarrollo de subcompetencias sobre las soluciones que presentadas en las diferentes actividades integradoras:***

**Actividad integradora numero 1:**

En el desarrollo de la primer actividad integradora era necesario abrir el archivo de entrada llamado "bitacora.txt que contenía una serie de accesos a Ip’s, su fecha, hora y descripción. Una vez que los datos eran leídos y almacenados en un vector era necesario ordenar la información por fecha para la realización de las búsquedas del usuario. En este programa se le solicitaba al usuario las fechas de inicio y fin de búsqueda de información y se retornaban en un archivo ordenado con las fechas de inicio y fin especificadas, para después ser desplegada en la pantalla del programa. En problemas de esta naturaleza, en donde los sets de datos son extremadamente grandes (En nuestra actividad se tiene un número de entradas que es algo elevado), o ya no sean tan sencillos de observar tanto por el usuario ni por la computadora es necesario aplicar métodos de búsqueda y ordenamiento como parte fundamental. En empresas con inmensos volúmenes de datos como Facebook, Twitter, Instagram, WhatsApp, Amazon, y en general empresas que almacenan cantidades descomunales de datos, debido a su gran cantidad de usuarios, el hecho de tener métodos de búsqueda y ordenamiento lentos los haría molestos y por ende el usuario podría dejar de usarlos buscando algo mejor y que no les haga perder tiempo.

Durante el desarrollo de la actividad se tenían dos opciones principales para la solución del problema, implementar un método de ordenamiento con complejidad computacional n^2 que era mas sencilla de hacer, o implementar un método de ordenamiento más difícil pero eficiente en tiempo y recursos (se implementó merge sort y bubble sort por cuestiones demostrativas). en el caso de la actividad realizada en bitácora teníamos una entrada de 16700 datos y queríamos ejecutar un ordenamiento que usara una complejidad n^2 implicaría que la computadora debería realizar 278,890,000 operaciones para solo ordenarlos, que en práctica pudimos notar que

Usando método de ordenamiento por Bubble Sort:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

La computadora tardó 1:33 min en solo realizar el ordenamiento, recordemos que el método de ordenamiento por bubble Sort tiene una complejidad computacional de n^2 lo que implica lo anteriormente mencionado, una entrada como la que se tenía de 16700 datos hizo que la computadora tuviera que ejecutar operaciones 278 millones de veces.

Usando método de ordenamiento por Merge Sort:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Mientras que al usar merge sort el cual tiene una complejidad computacional de n\*log(n), que con una entrada de 16700 implicaría que tuvo que realizar 70519.36 operaciones significando que este algoritmo se pudo haber ejecutado x3942 veces para llegar al mismo número de operaciones por la misma entrada! demorándose .184 segundos, ósea x510 veces más rápido que con bubble sort.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ahora bien, por último, la búsqueda en esta fase fue implementada con búsqueda secuencial un algoritmo que en el peor de los casos recorre todas y cada una de las entradas en búsqueda del dato especificado.

**Actividad integradora numero 2:**

En la actividad integradora numero dos eras necesario tener los resultados ordenados previamente en la actividad numero uno, esta vez haciendo uso de listas doblemente enlazadas para el acomodo de las ips dentro de cada uno de los nodos. En esta actividad el equipo realizo una implementación de listas doblemente enlazadas en donde cada identificador del nodo tenia una fecha que identificaba el mes y la fecha, en el interior de cada uno de estos nodos se tenia un mapa que contenía una key a cada acceso de ip en dada fecha específica y un numero de accesos específico para tal fecha., de esta forma se optimizaba cualquier tipo de búsqueda de numero de accesos por ips y se tenia un registro con búsquedas extremadamente rápidas en donde recorrer la lista enlazada se volvía una tarea mucho más sencilla ya que en caso de que el usuario buscara una fecha solamente requiere de día y mes y en vez de iterar por cada una de las ips en esa fecha solo era necesario buscar el dato de la fecha del nodo y ya podíamos acceder a las ips en esa fecha específica que queríamos de una forma más rápida. En nuestro caso y la aplicación que realizamos el uso de las listas doblemente enlazadas nos permite almacenar por fecha cada uno de los nodos y recorrer hacia la derecha para encontrar

fechas más recientes y para atrás para encontrar fechas más antiguas. El uso de las estructuras de datos nos permite tener un mayor control de nuestros datos y optimizar la búsqueda a nuestra conveniencia, además de poder implementar métodos que nos permitan realizar operaciones de formas más eficientes, ahorrando tiempo y en casos de aplicaciones de la vida real dinero y tiempo de ejecución para los usuarios.

Diagrama

Descripción generada automáticamente

**Actividad integradora numero 3:**

Por último, en la actividad integradora número tres era necesario haber acomodado de forma eficiente los datos, ya que en esta actividad era necesario implementar un árbol binario acomodando las direcciones con mas y menos accesos en el mapa, en donde si lo hacíamos de forma correcta a este punto el programa no debería de tener mayor lógica y se podría retornar alguna clase de impresión del árbol de forma contraria al inorder en donde se empezara por los nodos de la rama de hasta la derecha y así sucesivamente subiendo hasta llegar a retornar las 5 ips con mas accesos del programa la cual era la finalidad del problema, de esta forma se solucionaba la actividad.

**Por último, para el cierre de la reflexión se pregunta acerca de en las soluciones llevadas a cabo:**

**¿Cuáles son las más eficientes?**

Sin duda al principio el haber usado un algoritmo de ordenamiento con menor complejidad computacional redujo drásticamente el tiempo espera y compilación para las pruebas durante el proyecto y la correcta implementación por cada una de las fases permitió que se llevara a cabo un programa rápido y funcional. Creo que el algoritmo más importante fue el de merge sort y el de las listas doblemente ligadas, a partir de estas dos fases el árbol binario del final era mucho más sencillo de implementar, accesar y de ejecutar.

**¿Cuáles podrías mejorar y argumenta cómo harías esta mejora?**

Creo que la única mejora que le haría a alguno de los programas sería la implementación de un árbol autobalanceable en la ultima fase, debido a que si queremos insertar alguna nueva ip esta será acomodada probablemente en algún lugar no tan adecuado, con la implementación de un árbol autobalanceable podremos seguir retornando de forma sencilla los resultados de las ips con mayor numero de entradas todo el tiempo. En cuanto a las otras dos fases considero que se hizo usando las mejores formas posibles, como anteriormente se dio explicación en cada una de las explicaciones de las actividades integradoras.

**Conclusión:**

Lo más importante que me llevo para esta materia es el hecho de que los algoritmos y las estructuras de datos en la programación son fundamentales, además de ser algo importante para la carrera y para situaciones en la vida real son problemas y conocimientos que nos permiten ver más allá que solo lo simple y a su vez nos permiten mejorar el proceso cognitivo que tendremos como desarrolladores de software competentes al encontrarnos con problemas en donde el tiempo de respuesta y la experiencia del usuario serán muy importantes, y que a su vez las cantidades de datos colosales supondrán un problema para poder lograrlo. Gracias por el curso profesor. Atte. Carlo Luján.