INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

TC1031

Reflexión

“Actividad 1.3”

Alumna:

Sarah G. Martínez Navarro

A01703113

Fecha:

02 / Junio / 2022

Profesor:

Eduardo Arturo Rodríguez Tello

ÍNDICE

Introducción

¡El internet se encuentra bajo ataque! Con el avance de la tecnología, también han avanzado los ataques cibernéticos de los diferentes bots que buscan datos personales, información bancaria, entre otros datos que confiamos que deben estar seguros dentro de nuestros dispositivos. Sin embargo, las amenazas de ciberseguridad se encuentran ahora más presentes que antes y toda nuestra información está bajo el peligro de robo. En este reto debemos filtrar e identificar los posibles accesos maliciosos a través de los diferentes algoritmos que aprenderemos a lo largo de nuestra materia “Programación de estructuras de datos y algoritmos fundamentales”. Para desarrollar un programa que identifique exitosa y eficientemente estos accesos tendremos que tener un conocimiento profundo de las posibilidades que tenemos para programar para realmente elegir los algoritmos que nos entregarán los resultados correctos en el menor tiempo posible. Este proyecto estará dividido en múltiples secciones y actividades para llevar a cabo el constante desarrollo del programa que entregaremos al final del semestre. Con esto, pasamos al análisis de las actividades que conformarán el resultado final.

Actividad 1.3 (Búsqueda y Ordenamiento)

a) Estructura del Código

En nuestra primera actividad, tuvimos que darle forma al programa que íbamos a desarrollar, la base de la pirámide que será nuestra entrega final. Para esto, primero se crearon los documentos que almacenan las diferentes fases del código para mantener la limpieza y poder acceder fácilmente a las diferentes partes del mismo.

En esta versión del código se encuentran 3 áreas principales:

- Main: El main consiste sólo de un documento con terminación *cpp* donde llamamos las funciones que desencadenan todos los procesos que tenemos desarrollados en nuestros otros documentos. No debe ser muy largo porque la información puede empezar a esconderse por ahí y empieza a perder coherencia. Se convertiría en lo que llamamos “código espagueti”. Se nos ha enseñado en algunas otras materias que la meta es tener un “código lasagna”, que quiere decir, en capas. Y de ahí, se desarrollan las siguientes secciones de código.
- Registro: La sección registro contiene 2 documentos, uno con terminación *cpp* y otro con terminación *h*. En el documento *Registro.h* declaramos lo que son las clases en las que vamos a guardar las funciones de la sección. Podemos declarar las funciones como públicas o privadas, dependiendo de si queremos que se vean en otras clases o si queremos mantener la limpieza del sistema. En *Registro.cpp* desarrollamos las funciones que se declararon en el documento *h* para que sean funcionales para el programa que estamos desarrollando. Esta sección se encarga de separar y analizar los registros individuales dentro de la bitácora que ingresamos como datos. Para esto, definimos el constructor de la clase Registro para partir el registro en partes para poder ordenar después la lista. Un constructor Registro contiene mes, día, hora, minutos, ip, puerto y razón de falla del acceso.
- Bitácora: Similar a la sección registro, la sección bitácora también se compone de un documento *cpp* y uno *h* pero mantiene un objetivo diferente a la sección registro, ya que el área bitácora se encarga de la bitácora ya

completa para organizar todos los registros ya que estos han pasado por la sección registro. Dentro de bitácora es donde también declaramos el vector ListaRegistros para guardar los datos (constructor) de registro.

b) Algoritmo de Ordenamiento
Tras tener ya nuestros datos divididos en registros dentro de bitácora, podemos empezar a trabajar con ellos. Antes de intentar buscar un dato en la lista enorme de registros que tenemos es necesario ordenarlos por las fechas del suceso para facilitar el proceso de búsqueda. En la clase, vimos múltiples tipos de algoritmos para ordenamiento, pero los 2 que más consideré para esta actividad fueron el Bubble Sort y el Quick Sort. Estos son los algoritmos más eficientes de su respectiva área, el Bubble Sort siendo más fácil de implementar que el Quick Sort.

El Bubble Sort es el más simple de los algoritmos de ordenamiento. Funciona creando una burbuja alrededor de uno de los elementos de la lista que está ordenando (empieza seleccionando el primer número) y cuando encuentra un número menor la burbuja ahora selecciona ese número, continuando este proceso hasta atravesar todo el arreglo. Una vez que termina, pasa el número seleccionado (que es el que ha definido es el más pequeño de todo el arreglo) hasta adelante y repite el mismo proceso hasta que el arreglo está completamente ordenado. Así de tedioso como suena, suele ser.
Este algoritmo tiene una complejidad de tiempo promedio de $O(n^2)$ que es bastante alto, pero hace sentido comparando con la cantidad de veces que recorre el mismo arreglo para ordenarlo.

Mientras, tenemos también el Quick Sort. Como lo dice su nombre, suele ser mucho más rápido y, por lo tanto, más eficiente que el Bubble Sort. Sin embargo, es un poco más complicado de implementar. El Quick Sort es un algoritmo que cae en la categoría de “Divide y Vencerás”. Este proceso toma un número pivote, que es la mediana del listado y coloca los números menores a la izquierda y los mayores a la derecha del número pivote. Este procedimiento se repite de ambos lados de la lista y así sucesivamente hasta terminar de ordenar, pero como empieza a ordenar de ambos lados, el proceso se repite menos veces que el Bubble Sort.
El algoritmo de Quick Sort tiene una complejidad de tiempo de $O(n \log n)$ que es mucho más eficiente de lo que es el Bubble Sort.

Sin embargo, no se iba a definir hasta haberlo probado por lo que, con ayuda del profesor, implementamos ambos algoritmos y los probamos tanto con la bitácora acortada como con la bitácora completa. Los resultados fueron definitivos. El algoritmo Quick Sort

c) Algoritmo de Búsqueda
Tenemos 2 algoritmos muy sencillos de entender que estudiamos en esta materia para la búsqueda de elementos en un arreglo o, en este caso, un vector. Tenemos lo que son la búsqueda lineal y la búsqueda binaria.

La búsqueda lineal es cuando, por ejemplo, tenemos un arreglo y estamos buscando un valor pasando el índice uno por uno. El proceso empieza desde el índice **0** (el primer valor del listado) y se va moviendo a través de un *for* que determina "mientras que el valor del índice sea menor a la longitud del arreglo, se recorre el arreglo hasta encontrar el valor que buscamos". La complejidad del algoritmo de búsqueda lineal es de $O(n)$. No tiene que recorrer el arreglo (o en este caso, la listaRegistros) múltiples veces, por lo que no es un procedimiento muy complejo, sin embargo, debido a que tiene que ir espacio por espacio para encontrar el valor ingresado, es más tardado que la búsqueda binaria y, por lo tanto, menos efectivo.

Por el otro lado tenemos lo que es la búsqueda binaria, que fue el algoritmo utilizado en este reto. La búsqueda binaria consiste de un proceso de "Divide y Vencerás" (similar a lo visto con el Quick Sort). El primer paso es definir el medio del arreglo (en este caso, la mediana del arreglo ordenado) que será nuestro punto para filtrar rápidamente nuestro arreglo. Una vez con la mediana (que por ahora llamaremos el valor '*x*'), tenemos, claro, 2 lados en el arreglo: los valores mayores a '*x*' y los valores menores a '*x*'. Digamos que estamos buscando un valor '*y*' en nuestro arreglo. Lo que hace el proceso es definir si nuestro valor '*y*' es mayor o menor que nuestro valor '*x*', una vez con esa definición, el algoritmo filtra los valores que no requiere y se mueve al lado que en el que sí encaja el valor '*y*' y así sucesivamente. La razón por la que este método es tan efectivo es porque en cada paso, el proceso elimina la mitad del arreglo seleccionado mientras que la búsqueda lineal tiene que ir uno por uno. Además, la búsqueda binaria tiene una complejidad de $O\ (log\ n)$ que, bajo comparación, es mejor que la complejidad $O(n)$ de la búsqueda lineal.

Siguiendo un pensamiento lógico, es posible asegurar que, como la búsqueda binaria elimina la mitad (50%) del arreglo con cada paso que da y la búsqueda lineal tiene que recorrer el arreglo completo (100%), la búsqueda binaria tarda al menos la mitad del tiempo que tardaría la búsqueda lineal, en especial si se trata de un arreglo, vector o lista larga como la que tenemos en nuestra bitácora original. Tomando las dos opciones en cuenta, no es difícil encontrar la solución más efectiva para nuestro reto.

d) Conclusiones

Parte de nuestro trabajo como programadores consiste en determinar cuál de todos los caminos es el más eficiente y fácil de comprender para nuestros clientes, tomando también en cuenta que este debe ser un método consistente que no falle cuando pasan diferentes cantidades de información. En esta primera parte del reto aplicamos esta teoría para filtrar y ordenar una bitácora con una cantidad grande de datos que tenían que presentarse al usuario, claro, sin tardar demasiado tiempo. Para esto tuvimos que evaluar nuestras opciones en los algoritmos a usar y compararlos en cuanto a resultados, tiempo y complejidad. A lo largo de esta actividad aprendí los muchos métodos que se pueden utilizar tanto para ordenar como para buscar información. Tuve muchas dudas, pero con asesorías, se pudo resolver el problema. Había tenido mucha dificultad con entender cómo acceder a la información de un archivo aparte a través del código para poder usar los datos

dentro de él desde el semestre pasado, pero ahora he aprendido mucho más y creo que es algo que ya puedo implementar sola.
Una de las áreas de oportunidad que me gustaría mencionar es que me gustaría estudiar más tiempo sola (ver videos, leer, hacer ejercicios extra) para llegar con dudas más concretas a clases y asesorías y poder implementar más código yo sola sabiendo lo que estoy haciendo. Programar es algo que disfruto mucho una vez que entiendo lo que debo hacer, espero que con las actividades que haremos a lo largo de esta materia pueda agarrar más la onda de cómo es que funcionan estos algoritmos.

e) Referencias

- GeeksforGeeks (2022) *Bubble Sort*. Recuperado el 29 de Mayo, 2022 de https://www.geeksforgeeks.org/bubble-sort/?ref=lbp
- GeeksforGeeks (2022) *QuickSort*. Recuperado el 29 de Mayo, 2022 de https://www.geeksforgeeks.org/quick-sort/?ref=leftbar-rightbar
- GeeksforGeeks (2022) *Linear Search*. Recuperado el 02 de Junio, 2022 de https://www.geeksforgeeks.org/linear-search/
- GeeksforGeeks (2022) *Binary Search*. Recuperado el 02 de Junio, 2022 de https://www.geeksforgeeks.org/binary-search/