Reflexión

El documento de reflexión incluya la explicación de diferentes algoritmos de ordenamiento y búsqueda para esta situación problema, así como la complejidad computacional de cada uno de ellos.

El desarrollo de esta actividad fue relativamente más simple que los demás, pues involucró partes parecidas a las de tareas o actividades anteriores. Comenzamos analizando la actividad y estableciendo cómo adaptar lo que sabemos a la actividad.

En esta tarea se nos pide hacer un árbol que almacene las direcciones IP que hayan sido infectadas, contar cuantas veces fueron accedidas y mostrar las 5 con mas accesos en orden descendente.

Empezamos con las clases NodeT y BST para crear un Árbol de Búsqueda Binaria. La clase NodeT tiene establecidos los métodos normales de un nodo para un ABB como setters y getters de nodos hijos de derecha e izquierda y de la información que contiene el nodo. La información dentro del nodo la establecimos como un struct llamado ip, que contiene el IP en string, sin considerar el puerto, como lo pide la actividad, la cantidad de repeticiones de la dirección IP en la bitácora en la variable num y la sobrecarga de los operadores == y > para comparar IPs y cantidades de repeticiones. En NodeT también se encuentra la función ipToLong que convierte el IP de string a long long para usarlo con mayor facilidad. Todos los métodos dentro de esta clase son de complejidad constante, solamente la función ipToLong es de complejidad lineal.

Para la clase BST incluimos los métodos visit, con tres de sus diferentes recorridos, y add. Visit recibe un numero que elige el tipo de recorrido que se quiere hacer sobre el árbol, siendo las opciones preorden, inorden y posorden. Para facilitar la tarea a realizar, en el inorden lo limitamos a los primeros 5 números para mostrar solo las 5 direcciones más accedidas. Preorden y posorden se permanecen igual a los normalmente implementados. El método add es el mismo método add implementado en cualquier ABB. Ambos métodos son de complejidad lineal.

Finalmente establecimos el main, creamos nuestro objeto de la clase BST llamado total y lo mandamos a una función llamada leerArchivo que lee el archivo bitacoraOrdenada y almacena la información en el ABB total. La función leerArchivo lo que hace es que lee el archivo con un ifstream, y lo guarda con getlines en variables auxiliares. Estas variables se guardan en data si se trata del nodo actual y en aux si se trata del nodo anterior, posteriormente se comparan para aumentar el contador si son iguales y luego los añade en nodos NodeT al ABB total. Así se recorre todo el documento y se crean los elementos NodeT del ABB total hasta finalizarlo.

Después se llama al método visit con el numero 2 de parámetro para mostrar el recorrido en inorden con los 5 elementos más grandes del ABB total, es decir, las 5 direcciones IP más accedidas.

Fue interesante poner en práctica una vez más el uso de Árboles de Búsqueda Binaria, pues hacen que operaciones como el recorrido y búsqueda sean más eficientes. Esto es especialmente importante en situaciones como la del problema planteado en la que afecta directamente la seguridad de la información de personas o empresas mediante intentos de accesos a IPs.

Desarrollar esta actividad fue muy interesante, ver como mi equipo resuelve los problemas me sigue impresionando y me gusta aprender con y de ellos.