Tecnológico de Monterrey, Campus Monterrey
Programación de estructuras de datos y algoritmos fundamentales
Rodrigo Llaguno Cárdenas, A01198067
Grupo 206

Reflexión Act 2.3

Para la entrega de la segunda evidencia de nuestra actividad integral, teníamos que utilizar la estructura de datos de listas enlazadas para almacenar la información de ataques cibernéticos en forma de Botnet. Una vez almacenada la información en esta estructura de datos, teníamos que utilizar un algoritmo de ordenamiento para ordenar dicha estructura de datos en base a su IP de en orden ascendente para mostrar la información ordenada en una bitácora nueva.

Existen diferentes tipos de ciberataques pero el tipo con el cual estamos trabajando en esta evidencia es un botnet. Las botnets vienen de las palabras robot y network y sirven para describir dichos ataques los cuales son conectados por un malware y controlado por un propietario de los robots. Primero, estos robots infectan a las víctimas como computadoras mediante malware o incluso ingeniería social, término que describe la acción de infiltrarse usando técnicas sociales. Después se debe ampliar la botnet para infectar aún más otros equipos donde finalmente se activa la red. Con esto se pueden hacer desde acciones muy simples como leer y escribir datos de un sistema, enviar mensajes, instalar aplicaciones, recopilar información personal, hasta cosas más complejas como espiar en víctimas, crear ataques DDoS o hacer actividades de cripto minería (Belcic, 2021). Es importante identificar una botnet y poder tomar acción contra ella.

Las listas enlazadas son un tipo de estructura de datos en donde, a diferencia de un arreglo u otro tipo de estructura estática, es dinámica por lo que no se debe saber el número de elementos que debe contener desde un inicio. Existen diferentes tipos de listas enlazadas, su diferencia depende de la característica principal de esta estructura de datos, esto siendo, los apuntadores a los siguientes o previos elementos. Toda lista enlazada, tiene un valor almacenado. Si es una lista enlazada simple, adicionalmente contiene un apuntador al valor próximo. Si es una lista enlazada doble, contiene dos apuntadores, uno al valor próximo y uno al anterior. De esta manera, se puede construir una cadena de nodos apuntando a los nodos siguientes o pasados. Cada lista enlazada puede contener funciones para insertar,

eliminar, actualizar, encontrar, etc y cada una tiene una complejidad consecuente. Como en el proyecto utilicé listas enlazadas dobles, hablaré de la complejidad de sus funciones.

- insertFirst(), inserta un nodo nuevo al inicio de la lista encadenada, complejidad O(1)
- insertAtIndex(), inserta un nodo en un índice específico, complejidad O(n)
- insertLast(), inserta un nodo al final de la lista encadenada, complejidad O(1)
- deleteFirst(), elimina el primer nodo de la lista encadenada, complejidad O(1)
- deleteAtIndex(), elimina un nodo en un índice específico, complejidad O(n)
- deleteLast(), elimina el nodo al final de una lista encadenada, complejidad O(1)
- find(), regresa el nodo donde se encuentra el valor a buscar, complejidad O(n)
- update(), cambia el valor del nodo en un índice específico, complejidad O(n)
- showList(), muestra la lista en la consola, complejidad O(n)

Algunas ventajas de las listas enlazadas son que puede ser bueno para almacenar información de ciberataques en donde no se sabe la cantidad exacta de valores por almacenar. Además es fácil de acomodar y organizar la memoria, dándole cierta flexibilidad adicional. Además tiene funciones fáciles de implementar y útiles como invertir la lista y eliminar. Sin embargo, se debe tomar en consideración que se utiliza más memoria en comparación a un arreglo ya que se deben guardar dos punteros adicionales por cada valor.

Una pila es otra estructura de datos tipo *last in, first out*. Esto significa que una pila permite agregar nodos a la pila y solamente se pueden eliminar el nodo que fue más recientemente agregado. Esta función permite la implementación de cosas como el *undo* en editores y otro tipo de funciones de este estilo. Además pueden ser más seguras y permiten el control sobre la memoria. Sin embargo, se debe considerar que normalmente se debe mencionar el tamaño de la pila y no se puede eliminar un nodo que no sea el último ni tener acceso aleatorio.

El algoritmo de ordenamiento el cual debíamos de utilizar en el programa es Quicksort utilizando pilas con listas enlazadas. Este algoritmo funciona en base a un pivote donde hace una partición y se utilizan las funciones pop y push de la pila dentro de un while para ir ordenando la pila. El resultado de este algoritmo es una pila ordenada conteniendo listas doblemente enlazadas solamente utilizando funciones de una pila. Dicho algoritmo tiene una complejidad de O(n*logn).

Para la evidencia 2, se utilizó como base lo que se había construido en la primera evidencia. Se creó una clase tipo Datos donde se almacena toda la información de la bitácora incluyendo: mes, día, hora, ip y mensaje. Además se crearon otros atributos como mesI, ipTotal, etc para poder hacer una comparación numérica a la hora de utilizar los algoritmos de ordenamiento y búsqueda ya que estos datos originalmente eran de tipo string. Para poder obtener el ipTotal, es decir, un número total en versión de IP, se creó una función llamada conversionIpTotal. Esta función recibe como parámetro el string del ip a cambiar. Se separaban en 4 partes el ip utilizando la función substr y el delimitador "." para poder partir el ip. Después se verificó que cada uno de los 3 strings fuera de 3 caracteres. Si no lo era se le insertaba 1 o 2 ceros a la izquierda dependiendo del caso. Finalmente se sumaban los nuevos strings y se convertía a long.

A lo largo del desarrollo del programa nos encontramos con complicaciones en la función de QuickSort utilizando pilas con listas enlazadas. Se cree que hay problemas con los punteros y/o el tipo de datos que estamos usando, haciendo que no pueda correr el programa. No se pudo corregir dichos errores y es por eso que se usó mergeSort con listas enlazadas dobles para ordenar los datos mediante su IP, el cual lo logra hacer de una manera muy eficiente ya que mergeSort tiene una complejidad de O(n*logn). Aunque no es exactamente lo que se pidió en la entrega, la documentación de lo que se intentó en el quickSort está en los archivos que se entregaron y se buscó entregar algo que hiciera lo que la entrega pidiera en vez de entregar algo lleno de errores.

En cuanto al programa principal que si corre, se establecen todas las variables y se pide el rango de búsqueda del ip. Estos rangos los almacena en un objeto de tipo Datos y lo convierte a ipTotal. Después se crea la lista doblemente enlazada, se crea un nodo para cada línea de la bitácora y se inserta en la lista utilizando un insertLast(). Después ordena la lista utilizando merge sort, mismo algoritmo que la entrega pasada, siendo mergeSort, pero adaptado a listas enlazadas, y finalmente escribe los datos en la bitácora ordenada utilizando un for donde sea mayor que el ip inicial y menor que el ip final. Esto se logra usando la función find() de la lista enlazada y la función mostrarDatos() de la clase Datos.

Referencias:

Belcic, I. (2021, October 8). *¿Qué es una botnet? | Buscador gratuito de botnets*. Avast. https://www.avast.com/es-es/c-botnet

Estructuras de datos: listas enlazadas, pilas y colas. (n.d.). Estructuras de datos: listas enlazadas, pilas y colas. https://calcifer.org/documentos/librognome/glib-lists-queues.html

GeeksforGeeks. (2022, July 26). *Advantages, Disadvantages, and uses of Doubly Linked List*. GeeksforGeeks. https://www.geeksforgeeks.org/advantages-disadvantages-and-uses-of-doubly-linked-list/

GeeksforGeeks. (2022, June 13). *Applications, Advantages and Disadvantages of Stack*. GeeksforGeeks. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-stack/