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

Reflexión Act 3.4

Para la entrega de la tercera evidencia de nuestra actividad integral, teníamos que utilizar la estructura de datos Binary Search Tree (BST) para almacenar la información de ataques cibernéticos en forma de Botnet por la red de su IP para poder ordenar los datos en base al número de accesos que cad red tiene. Una vez almacenada la información en esta estructura de datos, teníamos que mostrar en pantalla las cinco redes con más accesos junto con la cantidad de accesos e imprimir en una bitácora nueva todos los IPs pero ahora ordenados en base a su número de accesos de mayor a menor.

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.

Los BST o Binary Search Trees son una estructura de datos basada en nodos conectados a un padre los cuales pueden tener dos, uno o ningún hijo. Primero se establece una raíz y dependiendo de los datos que se van insertando al árbol, se determina si dicho valor por insertar se irá a la izquierda o a la derecha. Si es menor que el nodo padre, se irá a la izquierda, si es mayor, se irá a la derecha. Este proceso se repite para cada nodo por insertar Esta estructura de datos es jerárquica, no lineal ya que todos el resto de los nodos surgen de el nodo raíz. Los BST se utilizan para organizar información de manera que las búsquedas se

puedan hacer eficientemente. La implementación de un BST en C + + se hace con punteros. Cada nodo tiene un objeto como valor y dos punteros hacía sus nodos hijos. Si el nodo no tiene hijos se declara como NULL. Existen diferentes funciones para un BST como búsqueda, insertar, eliminar y de recorridos, cada una con su propia complejidad.

- search(), busca el valor de un nodo del árbol, complejidad para BST no balanceado O(n), complejidad para BST balanceado O(logn)
- insertar(), inserta un nodo al árbol, complejidad para BST no balanceado O(n), complejidad para BST balanceado O(logn)
- modificar(), modifica el valor de un nodo dentro del árbol, complejidad para BST no balanceado O(n), complejidad para BST balanceado O(logn)
- eliminar(), elimina un nodo del árbol, complejidad para BST no balanceado O(n), complejidad para BST balanceado O(logn)
- preOrder(), recorre el árbol de manera nodo-izquierda-derecha, complejidad O(n)
- inOrder(), recorre el árbol de manera izquierda-nodo-derecha, complejidad O(n)
- postOrder(), recorre el árbol de manera izquierda-derecha-nodo, complejidad O(n)

Algunas ventajas de los árboles BST son que es una estructura de datos que se basa en una jerarquía haciendo que se puedan ver las relaciones entre los nodos. Además la inserción y la eliminación de nodos es más rápida que en arreglos o linked lists, haciendo de los BST, una estructura de datos muy eficiente. Sin embargo se tiene que tomar en consideración que para que los BST realmente sean así de eficientes se debe crear un BST balanceado, si no, realmente no hay beneficio en usar un BST.

Los BST son útiles cuando hablamos de botnets ya que al tener una manera eficiente de almacenar los datos en base a accesos, se pueden almacenar todas las redes en un bst donde se vaya midiendo el número de accesos si se repite la red. Al tener toda información en un árbol, es bastante rápida estas funciones. Luego se puede analizar las redes dependiendo del número de accesos asociada. Si una red tiene muchos accesos, se puede llegar a la conclusión de que podría ser una red infectada. De esta manera es una buena opción un BST para el análisis de botnets.

Para nuestra evidencia 3, decidimos implementar una solución de la siguiente manera. Creamos las siguientes clases, la clase Datos, donde se almacena toda la información relevante de la bitácora, la clase NodeTree, que se utiliza para crear los nodos que se van a

utilizar en el BST y la clase BST la cual es el árbol en sí y contiene todas las funciones de inserción y para desplegar. En nuestro main creamos una función obtenerRed() para que una vez que tengamos el IP, lo convierta a un int donde ese valor es la red del IP ya que eso es lo que usaremos para comparar y ordenar. Después empezamos con la creación del BST. De la bitácora obtenemos todos los datos línea por línea y los insertamos a nuestro BST con la función insertarDatos(). Lo que esta función hace es moverse a través del árbol para encontrar la red. Si una red de ese valor ya existe en el árbol, se incrementa el atributo de accesos por uno, si no, se inserta al árbol un nodo de esa red. Una vez el árbol completo usamos la función inOrder para poner de manera ordenada todos los nodos del árbol en un vector ya que vamos a tener que ordenar el árbol ahora por número de accesos no por la red como teníamos el primer árbol. Una vez con este vector de nodos, creamos un BST nuevo y recorremos el vector uno por uno insertándolos al árbol con la función insertAccesos() la cual los inserta pero ahora en base a sus accesos. Después usamos inOrderAccesos() para volver a poner todos los nodos del árbol en un vector y desplegar la red y accesos en consola de cada nodo. Finalmente buscamos los 5 últimos valores del vector y desplegamos en pantalla ya que estos son los valores con más accesos. Finalmente, imprimimos en la bitácora nueva la información de cada nodo ahora ordenada por accesos.

Referencias:

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

GeeksforGeeks. (2022, October 25). *Binary Search Tree*. GeeksforGeeks. Recuperado de https://www.geeksforgeeks.org/binary-search-tree-data-structure/

Sharma, R. (2020, May 21). *Binary Tree in Data Structure: Properties, Types, Representation & Benefits*. upGrad. Recuperado de https://www.upgrad.com/blog/binary-tree-in-data-structure/

GeeksforGeeks. (2022, June 13). *Applications, Advantages and Disadvantages of Binary Search Tree*. GeeksforGeeks. Recuperado de https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/