Actividad 3.4: Reflexión individual

Estudiante: Diego Palma Rodríguez **Matrícula:** A01759772

La situación problema consiste en analizar y manipular registros, contenidos en bitácoras, que poseen información sobre los intentos de acceder a una red. Los registros de la bitácora contienen la fecha, hora, IP y el estatus del intento de acceso. En este caso, se solicita utilizar los datos de las IPs para realizar el ordenamiento de los registros, y posteriormente encontrar las 5 IPs con mayor acceso a la red. Para lograr nuestro objetivo es importante definir la estructura de datos que se utilizará para almacenar los registros y el algoritmo de ordenamiento. En esta situación problema se utilizó un *maxHeap* representado como un árbol binario, y se aprovecharon sus propiedades para implementar el algoritmo de ordenamiento *HeapSort*.

Un árbol binario es una estructura de datos que está compuesta por nodos que tienen exactamente 2 sucesores, pero a diferencia de la *double linked list* estos se conectan de forma jerárquica y no lineal. De este modo se conoce como nodo padre al nodo con un nivel más de jerarquía y como nodos hijos a los nodos sucesores. Asimismo, los nodos que no tienen hijos son conocidos como nodos hoja [1]. Además de ello, en los árboles binarios se tiene un un nodo raíz y el resto de nodos deben seguir la siguiente regla para mantener la estructura balanceada: El nodo hijo derecho siempre debe ser mayor al padre, mientras que el nodo hijo izquierdo debe ser menor al padre. De acuerdo con [1], la importancia de mantener la estructura balanceada radica en que se reduce el costo computacional de los algoritmos de búsqueda y ordenamiento. En ese sentido, la ventaja de un BST es que proporciona facilidad en el ordenamiento y búsqueda de datos. De ser un árbol balanceado puede llegar a ordenar los datos con una complejidad computacional de O(nlog(n)) lo cual la hace una estructura de datos eficiente para esta aplicación.

El árbol binario también nos permite implementar un *MaxHeap*, en el cual se mantiene el dato con mayor valor (prioridad) como nodo raíz, y el resto de nodos se van ordenando en base a las características de prioridad, donde el nodo izquierdo tiene prioridad de llenado por encima del nodo derecho [1]. Esto es de gran importancia para el problema que queremos resolver, pues el *MaxHeap* nos permitirá acceder a la IP con mayor acceso de forma sencilla y rápida. Además de ello, al almacenar los datos en este tipo de estructura nos permite utilizar el algoritmo *HeapSort* para ordenarlos con un costo computacional de O(nlog(n)), y de este modo obtener las 5 IPs con mayor acceso.

Por otro lado, la situación problema nos plantea cómo determinar si la red está infectada o no. Para brindar una respuesta a ello, es importante saber cómo se accede a una red. De acuerdo con [2], cada vez que un usuario ingresa a una red, se realiza una solicitud que es enviada a un solucionador DNS (Sistema de nombres de dominio), el cual, en líneas generales, traduce los nombres de dominio a direcciones IP. Cada red y dispositivo tienen una IP única, y esta brinda información sobre la ubicación de los servidores o los dispositivos cada vez que se accede a la red. En este sentido, es posible detectar si una red se encuentra infectada al

determinar la cantidad de accesos que ha tenido. Si esta cantidad es anómala, se puede concluir que la red está infectada.

REFERENCIAS

[1] L. H. González Guerra, E. G. Salinas Gurrión, V. M. de la Cueva Hernández. (2020). *Estructuras de datos y algoritmos fundamentales. Capítulo 9, Estructura de datos jerárquicos*. Instituto Tecnológico y de Estudios Superiores de Monterrey, 2020.

[2] Amazon Web Services. *¿Qué es DNS?, Amazon*. [Online]. Disponible en: https://aws.amazon.com/es/route53/what-is-dns/. [Accessed: 13-May-2022].