Actividad 4.3: Reflexión individual

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

La situación problema consiste en analizar y manipular registros almacenados en bitácoras, los cuales poseen información sobre los intentos de acceder a una red. Los registros de la bitácora contienen el mes, día, hora, IP origen, IP destino, costo y el estatus del intento de acceso. En este caso, se solicita almacenar las IPs en un grafo; y a partir de ello, se pide determinar las 5 IPs con mayor grado con el objetivo de encontrar la IP en la que se encuentra el boot master. Finalmente, una vez encontrado el boot master, se solicita encontrar la distancia entre cada nodo y el boot master, y a partir de ello determinar cuál es dirección IP que el boot master requiere más esfuerzo para atacar.

En primer lugar, es importante definir la estructura de datos que se utilizará para almacenar los registros. En este caso se utilizará una lista de adyacencias (utilizada en actividades anteriores) para representar el grafo. Un grafo consiste en un conjunto de vértices y arcos [1]. Cada arco consiste de un par (v, w), pero en este caso en particular presentará una tercera componente que almacena el costo entre ambos vértices. De este modo, cada IP estará almacenada en cada nodo y se pueda determinar cuántos accesos realizó al calcular su grado. Además de ello, se posee la información del costo entre nodos y se puede acceder de forma rápida a ello. Cabe mencionar que almacenar los datos en un grafo nos permite implementar algoritmos de forma sencilla, como el algoritmo de Dijkstra que se explicará más adelante.

En segundo lugar, se hace uso de un *maxHeap* representado como un árbol binario, y se aprovechan sus propiedades para obtener las IPs con mayores grados. El árbol binario 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 [2]. 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 $O(n\,log(n))$, y de este modo obtener las 5 IPs con mayor acceso.

Finalmente, se implementa el algoritmo de Dijkstra para encontrar la distancia más corta entre el boot master y cada nodo. Dicho algoritmo es conocido por sus diversas aplicaciones en ciencias computacionales, robótica, optimización, logística, etc. Y es que debido a su complejidad temporal de $O(n\,log(n))$ y su rápida implementación, nos permite encontrar de forma eficiente el camino más corto de un nodo (el boot master en este caso) al resto de nodos. El algoritmo de Dijkstra utiliza la técnica de algoritmos voraces, donde la idea es básicamente buscar el camino más corto a partir del nodo origen y detenerse una vez que se llega al último nodo [1]. En este caso se utilizó una fila priorizada (con prioridad de menor costo) y se recorre cada vecino del nodo fuente realizando una actualización del costo solo

cuando encuentre los nodos con menor costo. Una vez determinada la mínima distancia entre el boot master y cada nodo, se ordenan dichas distancias en orden decreciente y se determina que la IP con mayor costo es la que presumiblemente requiere más esfuerzo para que el boot master la ataque.

REFERENCIAS

[1] Weiss, M. A. (2014). Data structures & algorithm analysis in C++. Reading, Mass: Addison Wesley.

[2] 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.