Jaime Eduardo López Castro - A00833173

Reflexión:

En esta implementación utilizamos 4 estructuras de datos destacables: Un grafo ponderado dirigido, listas enlazadas, maps de la implementación del STL y MaxHeaps (listas de prioridad). Utilizamos el grafo por su funcionalidad, ya que no hay mejor estructura para representar diferentes elementos interconectados entre sí, en este caso direcciones de Ip que realizan ataques cibernéticos a otras. En la implementación del grafo, utilizamos una lista enlazada simple, ya que la forma en la que almacenamos a este es mediante una lista de adyacencia, esto porque es más fácil extender las conexión entre nodos mientras vamos leyendo nuestro txt, gracias a que la complejidad de añadir elementos a una lista enlazada es O(1). Luego utilizamos el map, que está implementado en una estructura de tipo árbol binario de búsqueda que se ordena en base a un índice, lo que utilizamos para buscar la posición de ciertas ips en el grafo, la complejidad de búsqueda de un map es O(logn). Por último tenemos al MaxHeap, que utilizamos para el algoritmo de HeapSort, que tiene una complejidad de O(nlogn) y además nos sirve para obtener los valores máximos de un conjunto de datos en complejidad O(1).

La complejidad final del programa es O(V * E * logn), número de vértices por el número de enlaces por log de n. Leer el txt inicial y crear un grafo en base a él tiene complejidad de O(n * V), el número de líneas multiplicado por el número de nodos; ya que hacemos una operación por cada línea, cuando estamos añadiendo nodos creamos una lista enlazada vacía por cada uno y la ponemos en el vector de la lista de adyacencia. Cuando exportamos al txt las ip con sus grados la complejidad es O(n), ya que tenemos que recorrer todo el vector de datos. Para encontrar el Boot Master tardamos O(logn) ya que buscamos un valor dentro de nuestro map. En la función de shortestPath tenemos una complejidad de O(V * E * logn), número de vértices por el número de enlaces por log de n, ya que recorremos todo el número de vértices, recorremos sus vecinos y comparamos sus distancias, por lo que en el peor de los casos también recorremos todos los elementos del grafo, después añadiendolo a un MaxHeap con complejidad de O(logn), esto para que después sea más fácil encontrar el valor con más conexiones de todos. En la función TopFive tardamos O(logn) ya que tenemos que sacar 5 elementos del heap y luego reacomodarlo. Por último para encontrar el nodo con la mayor distancia al Boot Master tardamos un tiempo constante, por lo que es solamente acceder al top de nuestro MaxHeap, esto es extremadamente rápido.

La mayor aportación a la eficiencia de nuestro programa es el uso de maps y MaxHeaps para buscar elementos entre el universo y para obtener el elemento más grande del conjunto. La complejidad de búsqueda en un vector es O(n), mientras que en un map es O(logn), una grandísima mejora; de otra forma, la mayor diferencia es que en un MaxHeap tardamos un tiempo constante en obtener el valor más grande, mientras que en un vector la complejidad es lineal, una diferencia abismal.

Jaime Eduardo López Castro - A00833173

Fuentes:

- Kiao, U., PhD. (2022, 20 marzo). Time Complexity Analysis of Linked List. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/time-complexity-of-linked-list/
- GeeksforGeeks. (2022a, mayo 7). Graph and its representations. https://www.geeksforgeeks.org/graph-and-its-representations/
- GeeksforGeeks. (2022d, julio 6). map find() function in C++ STL. https://www.geeksforgeeks.org/map-find-function-in-c-stl/#:%7E:text=The%20time%20complexity%20for%20searching,searching%20is%20O(1).