Tecnológico de Monterrey

**Programación de estructuras de datos y algoritmos fundamentales**

**(Gpo 850)**

Actividad Integradora 4 - Grafos, algoritmos

Yael García Morelos | A01352461 | Campus Guadalajara

28 de mayo del 2025

A mi parecer, los grafos han sido la estructura más complicada de implementar, pues a pesar de conocer su estructura, su implementación con la bitácora llego a ser complicada por usar en los nodos únicamente la IP como key y no usar los datos completos.

Analizando esta actividad desde el inicio, fue el paso uno al que se le dedicó más tiempo. Al querer reutilizar un constructor para integers y querer adaptarlo para los registros resultó más complicado de lo esperado, tras tiempo de intentos y no conseguir resultados coherentes, se tomó la decisión de realizar el constructor desde cero, de esta manera se logró crear el grafo de manera correcta. Como ya se mencionó anteriormente, dentro del grafo no se encontraba toda la información de la bitácora, para almacenar esta información se implementaron "std::map", pues al ser una hash table y funcionar con keys, se realizó lo siguiente.

Un map para almacenar la IP y el índice donde se encuentra en el segundo map, el cual esta conformado por dos índices, el índice de la ip de origen y la del destino, entonces, al usar estos dos datos para generar una key compuesta puedo obtener el registro deseado.

La razón por la que se decidió implementar este tipo de estructura fue por su complejidad en la búsqueda de datos, pues como ya se sabe, las hash table tienen una complejidad aproximada de O(1) en búsquedas.

Tras finalizar con el constructor, la complejidad del resto de la actividad realmente no fue tan desafiante. Ya con el grafo hecho, determinar la salida de cada nodo fue sencillo, pues al leer un nodo te desplegaba la ip junto con el índice al que apuntaban (otra ip) y el peso que este implicaba, al guardar esta información en un vector, era más fácil obtener su tamaño haciendo uso de .size().

La dificultad de obtener las cinco ips con un mayor grado de salida recaía en la adaptación del grafo para crear un max_heap, se considera que hubiera sido más rápido la implementación de priority_queue, sin embargo, para esta actividad no estaba permitido. Tras crear un max_heap, obtener las 5 ips fue realmente sencillo pues usando .pop() con una complejidad de O(1) permitía obtener la información deseada de manera rápida y eficaz.

Deducir cual era el posible Bot Master también fue realmente sencillo, pues es aquel con mayor recurrencias. Para poder acceder fácilmente a su valor, se guardó la ip en un variable global de la clase Grafo.

Como ya se mencionó anteriormente, se tiene una función para convertir una ip en el índice en el que se encuentra dentro del segundo map, usando esta función, de manera iterativa se checó cual fue la primera conexión del Bot Master, se pudo haber implementado un ordenamiento previo para optimizar esta parte, sin embargo, como únicamente se iba a realizar una consulta y no era un numero alto de datos, se considero realizarse iterativamente porque era más sencillo y rápido de aplicar.

Utilizando el método de Dijkstra, fue posible encontrar los caminos más cortos que tiene el Bot Master a cada dirección IP, cada vez que finalizada un camino, se guardaba en distancia_botmaster.txt con la finalidad de optimizar espacios. De manera simultánea, se iba comparando los caminos con el pasado, de esta manera se determinó cual era el más grande,

pues así, no tenemos que volver a llamar el método de Dijkstra y optimizamos recursos. La ventaja de identificar el camino más corto entre las IP y el Bot Master y la IP que presenta un mayor esfuerzo de atacar fue que se mantiene la complejidad computacional del método de Dijkstra, pues los accesos tienen una complejidad promedio de O(1), mientras que Dijkstra de O((V+E)logV).

Por otro lado, tras conocer cual era la dirección IP con mayor distancia, se usó una función para imprimir el camino que este debe de pasar (direcciones IP) hasta su destino. Se usaron los nodos y sus índices para poder conocer las IP sin comprometer la complejidad computacional.

En conclusión, considero que si se planea correctamente la manera en la que realizaran los métodos y funciones se puede ahorrar tiempo a la hora de realizar el programa. Puedes optimizar recursos y tiempo de ejecución, además de realizarlo de manera más sencilla y en menos tiempo, pues con una planeación eficaz se pueden reducir errores en el programa. Considero que este programa se pudo haber realizado con una mayor calidad si fuera únicamente con un MaxHeap y HastTables, descartando el uso de grafos, esto, desde mi punto de vista y la facilidad con la que trabajo con otras estructuras de datos.