Campus Guadalajara

Verano 2022

Act 4.3 - Actividad Integral de Grafos

Fernando López Gómez | A01639715

Programación de estructuras de datos y algoritmos fundamentales

TC1031.570

Dr. Eduardo Arturo Rodríguez Tello

Fecha de entrega: 27/07/2022

Para el desarrollo de esta actividad se utilizó la estructura de datos Grafo.“Un grafo en el ámbito de las ciencias de la computación es un tipo abstracto de datos (TAD), que consiste en un conjunto de nodos (también llamados vértices) y un conjunto de arcos (aristas) que establecen relaciones entre los nodos.” (Wikipedia, s.f).

La bitácora en esta actividad nos arroja nuevos datos: La ip de origen del posible ataque y la ip de destino, es decir, la ip del dispositivo que recibe el ataque. Existen diferentes maneras de abordar esta situación, siendo la mejor de ellas la implementación de un grafo, esto debido a que nos ayuda a relacionar los datos de las Ips de origen y de destino para encontrar qué direcciones producen la mayor cantidad de ataques, al mismo tiempo que se pueden realizar otro tipo de operaciones de una manera más eficiente que en la implementación de otros ADT’s. Poniendo un ejemplo, si guardamos en un vector la información que queremos obtener de cada dirección Ip, aparte de que tendríamos que iterar n cantidad de veces la misma bitácora para poder extraer n cantidad de veces los mismos datos para cada dirección Ip, si es que queremos modificar la información dentro del vector tendríamos que modificarla en cada una de las casillas del vector, haciéndolo un proceso con complejidad temporal demasiado alta. Ahora bien, con el uso de los grafos podemos encontrar los datos de una manera más rápida y eficiente, esto debido a que como se ha creado una relación entre 2 nodos, ya sea unidireccional o bidireccional, podemos extraer información en una complejidad que se acerca mucho a la complejidad constante, siendo su complejidad de búsqueda de O(|V| +|E|) donde V es el número de vértices y E el número de enlaces o vértices. Tampoco se optó por utilizar el ADT de BST debido a que los nodos de los árboles tienen un número limitado de relaciones.

Además de que las operaciones dentro de un grafo son significativamente más eficientes, también nos ayuda a implementar el algoritmo de búsqueda de rutas óptimas denominado Algoritmo de Dijkstra. Utilizando este algoritmo podemos encontrar la ruta más corta entre un nodo del grafo y otro, a la distancia entre un nodo y otro se le denomina peso y es el factor que utiliza para evaluar cuál ruta es mejor que otra. Es un algoritmo muy eficiente debido a que si se buscara encontrar esta información, la complejidad temporal subiría mucho al tener que estar buscando en cada nodo los diferentes nodos con los que se tiene relación además de ir guardando diferentes datos en memoria, lo que a su vez aumenta la complejidad espacial; el algoritmo de dijkstra recorre el grafo utilizando como referencia la búsqueda BFS, logrando una complejidad de O(E+VlogV).

Ahora bien, para implementar este grafo primeramente se determinó utilizar los grafos ponderados para poder establecer relaciones direccionadas, ya que para los grafos no ponderados no podríamos determinar cuál es la ip de origen y cuál es la ip de destino dentro de una relación.
Para la creación de este grafo ponderado se establecieron las direcciones IP para cada uno de los nodos (junto con su índice en la bitácora para poder realizar una búsqueda aún más rápida con complejidad temporal de O(1)) para así, poder relacionarlos con la información contenida en los registros. Se determinó el uso de la representación del grafo como lista de adyacencia debido a su complejidad temporal y simplicidad, ya que esta representación tiene una complejidad temporal de O(n) mientras que la matriz de adyacencia tiene complejidad de O(n^2) en grafos

dirigidos debido a que se debe de recorrer cada celda de la matriz para encontrar si existe alguna relación.

Gracias a la implementación de todo lo anterior, pudimos crear una función que obtuviera de una manera mucho más rápida el grado de salida de cada nodo, ya que simplemente es obtener el número de elementos dentro de la lista de adyacencia de cada nodo, haciéndolo así una función de complejidad temporal de O(|V|).

Para la extracción de las ip's con mayor grado de salida se implementó ADT tipo binary Heap de objetos tipo dirección Ip, lo cual nos permite crear un arreglo que se ordene por los grados de salida y así simplemente obtener el primer dato y eliminarlo posteriormente para asegurar que el siguiente dato que se extraiga sea en efecto el siguiente mayor. La implementación del binary heap tiene una complejidad temporal de O(n log n), misma con la que se cuenta si se hubiera utilizado el método mergesort, la ventaja de utilizar el binary heap sobre un mergesort es que los métodos para añadir y remover elementos del heap están dentro de la complejidad de O(n log n), mientras que el mergesort se tendría que ejecutar cada que realizamos un cambio dentro del arreglo para que dicho arreglo se mantenga ordenado, además, la complejidad espacial del mergeSort es mayor a la del heap debido a que el heap cuenta con menos llamadas recursivas y no necesita crear nuevos arreglos.

Ahora bien, sabiendo que el boot master es aquella dirección Ip con el mayor grado de salida, podemos obtener las distancias más cortas que le tomaría al boot master atacar a las otras direcciones Ip gracias al algoritmo de Dijkstra mencionado previamente. Simplemente establecemos al índice del boot master como el punto de partida y de ahí evaluar la distancia hacia cada uno de los nodos en su lista de adyacencia. Estas distancias se van almacenando en un vector para poder obtener la distancia más corta encontrada mientras se exportan los valores dentro de dicho vector para reducir la complejidad temporal, ya que si se creaba una función adicional para poder determinar la mayor distancia, se tendría que recorrer el mismo arreglo de distancias 2 veces, una para exportarla y otra para evaluar la mayor distancia.

Referencias:

Wikipedia Editor(s.f) *Grafo (tipo de dato abstracto)*. Wikipedia. https://es.wikipedia.org/wiki/Grafo_(tipo_de_dato_abstracto)#:~:text=Un%20grafo%20en%20el%20%C3%A1mbito,del%20concepto%20matem%C3%A1tico%20de%20grafo.

Rashmi Raj (s.f) *Analysis of Algorithms.* Stanford University http://www-cs-students.stanford.edu/~rashmi/projects/Sorting#:~:text=HeapSort%3A,because%20of%20the%20second%20array.