Tecnológico de Monterrey

Act 4.3 - Actividad Integral de Grafos (Evidencia Competencia)

Nikole Morales Rosas - A01782557

TECNOLÓGICO DE MONTERREY

Campus Santa Fe

Grupo: 570

Fecha de entrega: 24 de julio de 2023

Profesor: Dr. Eduardo A. Rodríguez Tello

Programación de estructuras de datos y algoritmos fundamentales

Grafos

Reflexión

Dentro de la informática existen los grafos que son una estructura de datos no lineales, está compuesta por vértices y aristas.

Vértices: Es el elemento básico a partir del cual se construyen los grafos, ya que en los vértices (también llamados nodos) es donde se almacena la información.

Aristas: Son las líneas que unen a los vértices.

Desde un punto de vista más práctico los grafos se representan con un conjunto de puntos unidos con líneas formando una red en la que se almacena la información. Según explica un artículo de la Universidad de Oviedo, existen dos tipos de grafos: grafos dirigidos y no dirigidos. Mientras que los grafos dirigidos se representan gráficamente mediante flechas asociadas de forma unidireccional, los grafos no dirigidos solo se conectan mediante líneas no direccionales lo que significa que las líneas que conectan a los vértices van en ambos sentidos. (UNIOVI, s/f).

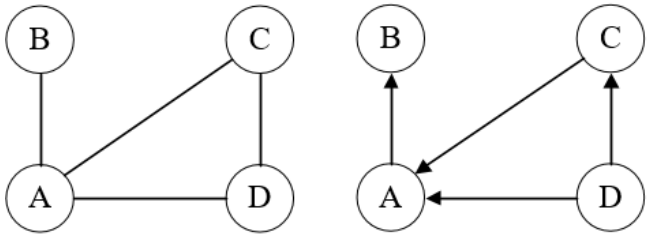

Fig 1.1: Ejemplo de una representación de un grafo no dirigido y uno dirigido.

Para este caso se empleó una estructura de lista enlazada para almacenar el número de conexiones entre nodos, en este caso cada nodo guarda la dirección IP. Se utilizó un método llamado ***getAllDegreeOut*** para obtener y guardar información sobre los grados de salida de cada nodo en el grafo para posteriormente generar un archivo *"grados_ips.txt"* con esos datos.

Después de esto, se obtuvieron los grados de cada vértice para poder acomodarlas de mayor a menor utilizando un método Max Heap llamado ***ipSort***, para almacenar objetos ***IpDegreeOut*** que contienen información sobre las direcciones IP y sus grados de salida, de igual manera que en el caso anterior también se generó un archivo de salida llamado "mayores_grados_ips.txt" donde de muestran los IP's con sus grados de salida.

A continuación, se guardaron las 5 IP's con mayor grado de salida en un archivo denominado ***topFile***, y todos los grados de salida se guardan en un archivo

denominado ***allFile***, al igual que también guarda la información sobre el nodo que tiene el grado de salida máximo (Boot Master) para imprimirla en pantalla.

Cabe mencionar que para este punto se utilizó la clase Heap que se implementó en la Actividad 3.3. En lo que respecta a la creación y ordenación del Max Heap en el programa implementado, inicialmente, la construcción del Max Heap ocurre en un bucle for dentro de la función ***heapSort***. Durante este proceso, se ejecuta la función ***heapifyHelper*** para cada nodo que no sea una hoja en el árbol.

Como se explica en un artículo de Geek For Geeks, este proceso comienza desde el último nodo interno y retrocede hacia el primer nodo. En cada llamada a ***heapifyHelper***, la función ajusta los nodos descendiendo a través del árbol para asegurar que se mantenga la propiedad del Max Heap. Una vez que el Max Heap ha sido construido, se procede con la fase de ordenación real. Para ello, se extrae el elemento máximo (la raíz) del Heap y se coloca en la última posición del vector. Luego, se invoca ***heapifyHelper*** nuevamente para mantener la propiedad del Max Heap en el subárbol restante.

(GeekForGeeks, 2021).

En términos de complejidad temporal, HeapSort tiene una complejidad de O(n log n), a comparación de métodos como el BubbleSort o InsertionSort que estos tienen una complejidad cuadrática, es decir o (n^2), sin embargo hay otros algoritmos como MergeSort o QuickSort que son un poco más rápidos, pero a pesar de esto el heapSort sigue siendo el más eficiente y adecuado a la hora del ordenamiento.

Según la ANIEI (Asociación Nacional de Instituciones de Educación en Tecnologías de la Información): “Heap sort alcanza esta complejidad temporal de O(n log n) debido esencialmente a que utiliza una estructura de datos de modo que cada operación de salida requiere a lo sumo *log i* pasos, donde *i* es el número de elementos restantes.”

(ANIEI, s/f).

Finalmente se utilizó un algoritmo de Dijkstra para encontrar el camino más corto entre la IP que identificó como el boot master y el resto de los nodos del grafo. Los datos se almacenaron en un archivo llamado *"distancia_bootmaster.txt"* el cual es una lista con que contiene la distancia total del camino más corto entre el nodo que tiene el mayor grado de salida y la IP.

En una situación de esta naturaleza los grafos son una estructura bastante útil y tiene varias ventajas al momento de modelar las relaciones entre los elementos. A la hora de buscar los nodos con mayor grado de salida, al usar los grafos se puede recorrer todos los nodos del grafo y calcular el grado de salida de cada uno en tiempo lineal O(N + E), donde N es el número de nodos y E el número de aristas. Otra de las ventajas es al usar el algoritmo de Dijkstra cuando se busca

el camino más corto el cual tiene una complejidad computacional de O((n + m) * log(n)), con este algoritmo hace que la operación sea más rápida y eficiente aún cuando la red es muy grande o son muchos datos.

En general el uso de grafos en esta situación es muy importante ya que gracias a los grafos se logra una representación más eficiente de los datos, así como también es más fácil detectar algún comportamiento sospechoso al analizar los grados de salida, se puede detectar una IP que intenta ingresar a la red repetidas ocasiones. Por otro lado permiten una visualización más intuitiva para identificar nodos críticos.

En esta situación, el uso de grafos proporciona una manera eficiente y estructurada de almacenar y analizar los datos relacionados con las direcciones IP y sus conexiones en la red. Al utilizar grafos, es posible identificar nodos con alto grado de salida, encontrar el camino más corto entre el boot master y otros nodos, y detectar patrones de comportamiento sospechosos. Esta información permite mejorar significativamente la detección y respuesta ante posibles amenazas de seguridad en la red, brindando una solución efectiva y eficiente para protegerla contra actividades maliciosas. En resumen, el uso de grafos en esta situación facilita el análisis de la red, la identificación de patrones y el seguimiento de caminos, lo que permite tomar decisiones informadas para mantener la seguridad y proteger los activos de la red.

Referencias

Anónimo. Uniovi. (s/f). *Grafos. 6.1 Conceptos previos y terminología*. España. Recuperado el 24 de julio de 2023 en:
https://www6.uniovi.es/usr/cesar/Uned/EDA/Apuntes/TAD_apUM_07.pdf

Sin Autor. Heap sort - data structures and algorithms tutorials. (2013, marzo 16). GeeksforGeeks. Recuperado el 24 de julio de 2023 en:
https://www.geeksforgeeks.org/heap-sort/

Sin Autor. Graph data structure and algorithms. (2016, mayo 17). GeeksforGeeks. Recuperado el 24 de julio de 2023 en:
https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Sin Autor. Graph and its representations. (2012, noviembre 13). GeeksforGeeks. Recuperado el 24 de julio de 2023 en:
https://www.geeksforgeeks.org/graph-and-its-representations/