Mariana Aguilar Requena
A01569276

Tecnológico de Monterrey

Act 4.3 - Actividad Integral de Grafos (Reflexión)

En esta actividad trabajamos con una bitácora de red donde se registran conexiones entre distintas direcciones IP. En actividades anteriores donde los datos estaban organizados por fecha, aquí el reto fue analizar las relaciones entre las IPs, ya que una misma dirección podía estar conectada con muchas otras. Primero, leímos el archivo **bitacoraGrafos.txt** y guardamos la información en una estructura de grafo usando una lista de adyacencias. Cada IP se manejó como un nodo y cada conexión como una arista dirigida desde la IP de origen hacia la IP destino. Esta representación fue importante porque nos permitió organizar los datos de forma más clara y eficiente que si se hubiera usado un arreglo o una lista simple.

¿Por qué lista de adyacencias?

La lista de adyacencias fue una mejor opción que una matriz de adyacencias, ya que el grafo no es completamente denso. En una matriz se tendría que reservar cada espacio para todas las posibles conexiones, aunque muchas no existan, lo que desperdicia memoria. En cambio, la lista de adyacencias solo guarda las conexiones reales de cada IP, haciendo el programa más eficiente en espacio. La construcción del grafo tiene una complejidad aproximada de O(n).

Uso de Heap para encontrar las IPs con mayor grado

Para obtener las 7 IPs con mayor grado de salida, utilizamos la estructura Heap que habíamos implementado en actividades anteriores. Esta decisión fue clave, ya que de otra manera hubiera sido ordenar todas las IPs por su grado usando algún algoritmo de ordenamiento. Por ejemplo, usar Bubble Sort tendría una complejidad de $O(n^2)$, lo cual no es eficiente para grandes cantidades de datos.

El Heap nos permite insertar y extraer elementos en O(log n), lo que hace mucho más rápido encontrar los valores más grandes sin ordenar toda la lista. Gracias a esto, fue posible identificar de forma eficiente al posible boot master, que corresponde a la IP con mayor grado de salida.

Caminos más cortos

Después de identificar al boot master, se analizó qué IP requiere mayor esfuerzo para el algoritmo de caminos más cortos tiene una complejidad aproximada de O(V + E), donde *V* es el número de nodos y *E* el número de aristas. Este método es mucho más eficiente que recorrer el grafo de manera desordenada o probar todas las rutas posibles.

El uso de grafos permitió analizar no solo datos individuales, sino también las relaciones entre ellos. A diferencia de estructuras más simples, los grafos permiten responder preguntas más complejas, como identificar nodos importantes. Esto demuestra que los

grafos son una herramienta fundamental en problemas reales relacionados con redes y seguridad informática.

Para concluir, no tenía tan claro cómo se podían aplicar los grafos a problemas reales. Al trabajar con esta bitácora, entendí que los grafos no solo sirven para representar datos, sino para analizar relaciones y tomar decisiones. También aprendí que elegir y usar estructuras de datos eficientes, como listas de adyacencias y Heaps, hace una gran diferencia en el rendimiento del programa.

Referencias

GeeksforGeeks. (s. f.). Graph data structure. Recuperado de https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

GeeksforGeeks. (s. f.). Heap data structure. Recuperado de https://www.geeksforgeeks.org/heap-data-structure/

Wikipedia. (s. f.). Grafo. Recuperado de

https://es.wikipedia.org/wiki/Grafo

Khan Academy. (s. f.). Representación de grafos. Recuperado de

https://es.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs