Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Programación de estructuras de datos y algoritmos fundamentales (Gpo 573)

Profesores: Dr. Eduardo Arturo Rodríguez Tello

Andres Romo Castañeda A01234579

Act 4.3 - Actividad Integral de Grafos (Evidencia Competencia)

30/01/2026

Monterrey, Nuevo León

Reflexión Individual - Actividad 4.3

Para resolver este reto, lo primero que tuvimos que decidir como equipo fue cómo representar el grafo. Teníamos dos formas, la primera era matriz de adyacencia y la segunda era lista de adyacencia. Al analizar el bitacoraGrafos.txt, vimos que aunque hay muchas IPs, no todas están conectadas con todas. Es un grafo disperso. Si hubiéramos usado una matriz, habríamos desperdiciado muchísima memoria reservando espacios en cero para conexiones que no existen. Por eso, nos fuimos por la Lista de Adyacencia. En nuestra implementación basada en la clase Graph, usamos un std::vector donde cada índice guarda una LinkedList. Luego, el siguiente paso fue identificar al boot master. Aquí lo que hicimos fue buscar el nodo con mayor grado de salida. Por eso utilizamos el max heap. En lugar de ordenar toda la lista de IPs con un algoritmo O(n log n) para luego solo tomar las primeras 7, simplemente metimos los grados de salida a nuestra prioridad queue. Esto nos permitió extraer el top 7 y encontrar al boot master de una forma super directa. Aquí es donde las estructuras del grafo modela la red y el Heap nos ayuda a procesar prioridades sobre esa red rápidamente.

Después tuvimos que calcular distancias. Para esto implementamos el algoritmo de Dijkstra. Ver a Dijkstra correr sobre nuestro grafo fue interesante porque a diferencia de un BFS que solo cuenta saltos, Dijkstra está preparado para caminos ponderados. Si hubiéramos intentado hacer esto a la fuerza o con una implementación de arrays, la complejidad se hubiera disparado más compleja y con más procesos, y el programa se hubiera tardado. Al final, el algoritmo nos imprimió la lista de distancias y pudimos ver que la IP más difícil de atacar es, lógicamente, la que está más lejos del boot master.

Algo que discutimos mucho mientras programamos fue la direccionalidad. Al principio no estábamos seguros si el grafo debía ser dirigido o no, pero al leer bien la bitácora entendimos que si la IP A ataca a la B, no significa que B ataque a A. Definir el grafo como dirigido fue clave porque si lo hubiéramos hecho no dirigido, el cálculo del grado de salida hubiera sido incorrecto. Me voy de esta actividad entendiendo que los grafos son la estructura más versátil que hemos visto. Sirven para Google Maps, para redes sociales, Uber, etc. Implementar esto en C++ y ver cómo el código procesa miles de líneas de logs para darte una respuesta clara te da una sensación de poder muy distinta a solo hacer programas de hola mundo. Me gusta trabajar en este tipo de actividades donde hay cosas del día a día que no pensamos mucho pero detrás de ellas hay un proceso complejo e interesante. Siento que cada vez que ordene un Uber o pida direcciones en GPS, me dara una sensación de curiosidad para identificar qué tipo de algoritmos usaron para recorrer mi trayectoria, que eficiencia, etc.

Referencias:

- GeeksforGeeks. (2023). *Heap Sort - Data Structure and Algorithms Tutorials*. Recuperado de https://www.geeksforgeeks.org/heap-sort/
- Cplusplus.com. (2024). *std::priority_queue*. Recuperado de https://cplusplus.com/reference/queue/priority_queue/
- Cloudflare. (2024). ¿Qué es el análisis de logs y cómo ayuda en la seguridad?. Recuperado de https://www.cloudflare.com/es-es/learning/network-layer/what-is-log-analysis/