Instituto Tecnológico y de Estudios Superiores de Monterrey

Act 4.1 - Actividad Integral de Grafos (Evidencia Competencia)

Programación de estructuras de datos y algoritmos fundamentales (Gpo 570)

Integrantes:

Noreth Sofia Villalpando Saldaña A01368579

Héctor Calderón Reyes A01350637

Campus:

Toluca y Guadalajara

Profesores:

Dr. Eduardo Arturo Rodríguez Tello

Fecha de entrega:

26 de Julio de 2023

Reflexión Héctor Calderón:

Los grafos son una de las estructuras de datos más poderosas y versátiles disponibles para los informáticos y son fundamentales para una amplia gama de aplicaciones, incluyendo el análisis de redes, la planificación de rutas, la programación y mucho más. (Graph Everywhwer s.f) En el contexto de este problema, nos enfrentamos a un conjunto de datos que representa las interacciones entre diferentes direcciones IP en una red. Nuestro objetivo es descubrir patrones y características importantes de esta red utilizando conceptos y algoritmos de grafos. (Rojas K, s.f)

Comenzamos leyendo la información del archivo de entrada "bitacoraGrafos.txt" y almacenándola en una lista de adyacencias organizada por la dirección IP de origen. Cada nodo del grafo representa una dirección IP y cada arista representa una interacción entre dos IPs. Esta representación nos permite modelar la red de una manera que facilita el análisis posterior. Sin embargo, la creación de esta representación implica una complejidad de tiempo de O(n^2 log n), debido al proceso de lectura del archivo y a la búsqueda binaria para insertar cada elemento en su lugar correspondiente.

A continuación, determinamos el grado de salida de cada nodo del grafo, que representa el número de IPs adyacentes a cada IP de origen. El grado de salida de un nodo puede ser un indicador importante de su papel en la red. Calcular y almacenar los grados de las IPs tiene una complejidad de tiempo de O(n), ya que necesitamos recorrer cada nodo una vez.

Una vez que tenemos los grados de salida de todas las IPs, determinamos las 5 IPs con mayor grado de salida. Utilizamos una cola de prioridad (o un "heap") para hacer esto de manera eficiente. La complejidad de tiempo de esta operación es O(n log n), ya que necesitamos insertar cada elemento en la cola de prioridad y luego extraer los 5 elementos máximos.

El nodo con el mayor grado de salida es identificado como el "boot master". Esta es una suposición basada en el hecho de que un nodo con muchas conexiones podría tener un papel central en la red. Sin embargo, hay que tener en cuenta que esta suposición podría no ser siempre válida, ya que el grado de salida no es el único factor que determina la importancia de un nodo.

Utilizando el algoritmo de Dijkstra, encontramos el camino más corto entre el "boot master" y todos los demás nodos del grafo. Este algoritmo tiene una complejidad de tiempo de O((n+m) log n) cuando se usa una cola de prioridad binaria, donde n es el número de nodos y m es el número de aristas. Este algoritmo es muy eficiente para encontrar los caminos más cortos en un grafo ponderado y es uno de los algoritmos de grafos más conocidos y utilizados.

Finalmente, identificamos la dirección IP que requiere más esfuerzo para que el "boot master" la ataque. Esta es la IP que tiene la mayor distancia del "boot master" en el grafo. Esta información podría ser útil para entender las vulnerabilidades de la red y planificar medidas de seguridad.

Reflexión Noreth Villalpando:

Los grafos son una herramienta esencial en la resolución de problemas relacionados con conexiones, redes y relaciones entre elementos. En este contexto, utilizamos grafos para representar desde una bitácora de registros los ataques de direcciones IP a otras direcciones IP . Al modelar la red como un grafo, se simplifica el análisis de la estructura y se posibilita la identificación de patrones y comportamientos relevantes (GraphEverywhere, 2021).

Para la actividad utilizamos una lista de adyacencia como la estructura de datos para representar el grafo, ya que el grafo es disperso debido a que no todas las IPs se comunican con todas las demás IPs. Al usar la lista de adyacencia se ahorra memoria y permite un acceso eficiente a los vecinos de cada IP. De esta forma, se logra un equilibrio entre espacio y tiempo, lo que es especialmente relevante al trabajar con registros de gran tamaño.

Para lectura de la bitácora de grafos usamos el método de readGraph que lee la bitácora y construye la lista de adyacencia, tiene una complejidad de O(n^2 log n) debido a la lectura y el ordenamiento de las direcciones IP (Baeldung, 2022). Dentro de este método también se van almacenando los grados de salida, que representan el número de aristas que salen de cada nodo. Con esta información podemos identificar fácilmente el bot master, pues el IP con mayor out degrees es el que ha atacado más veces.

También usamos el algoritmo de búsqueda binaria ya que destaca por su complejidad (log n). Esta eficiencia se logra gracias a la propiedad de la búsqueda binaria de dividir el espacio de búsqueda en cada iteración. Así, se obtiene una búsqueda rápida incluso en listas o estructuras de datos con muchos elementos, como las que se usaron en la actividad.

El método de Dijkstra fue para encontrar las distancias más cortas entre el nodo de origen (ip bot master) y el resto de los nodos (ips), aunque eficiente, puede enfrentar desafíos en grafos densos con muchas aristas. Sin embargo, al implementarlo con una priority queue para seleccionar el nodo con la distancia mínima en cada iteración, se logra una complejidad de O((V+E)logV) (Pandey, 2021). Los resultados obtenidos a través del algoritmo de Dijkstra proporcionan información sobre la distancia desde el bot master hasta cada uno de los demás nodos en la red y esta información nos permite saber que IP que requiere el mayor esfuerzo para ser alcanzada o que tienen una seguridad de baja calidad por el número de ataques que ha tenido.

La implementación de estos algoritmos ayuda a comprender más profundamente los mismos registros de la bitácora, lo que podría ser de gran utilidad para identificar comportamientos anómalos, detectar ataques y de esta forma mejorar la seguridad.

Referencias

- Baeldung. (2022). Time and Space Complexity of adjacency Matrix and List | Baeldung on Computer Science. *Baeldung on Computer Science*. https://www.baeldung.com/cs/adjacency-matrix-list-complexity
- Graph Everywhere. (n.d.). Grafos para prevenir ataques a tu organización. Recuperado el 26 de julio de 2023, de https://www.grapheverywhere.com/grafos-para-prevenir-ataques-a-tu-organizacion/
- GraphEverywhere, E. (2021). El rol de los grafos en la ciberseguridad. *GraphEverywhere*. https://www.grapheverywhere.com/el-rol-de-los-grafos-en-la-ciberseguridad/
- Pandey, M. (2021). Dijkstra's Algorithm - Scaler topics. *Scaler Topics*. https://www.scaler.com/topics/data-structures/dijkstra-algorithm/
- Rojas, K. (n.d.). Análisis de redes. En Ciencia de Datos. Recuperado el 26 de julio de 2023, de https://bookdown.org/keilor_rojas/CienciaDatos/an%C3%A1lisis-de-redes.html