Tecnológico de Monterrey

Act 4.3 – Actividad Integral de Grafos (Evidencia Completa)

Programación de Estructuras de Datos y Algoritmos Fundamentales

Instituto Tecnológico y de Estudios Superiores de Monterrey

Fecha de entrega: 25 de julio de 2023

Estudiantes: Alyson Melissa Sánchez Serratos - A01771843

Profesor: Dr. Eduardo Arturo Rodríguez Tello

Reflexión: Actividad Integral de Grafos

Hoy en día, la tecnología y la conectividad se han vuelto fundamentales para la sociedad, impulsando avances significativos en diversos campos. Sin embargo, junto con estos avances tecnológicos también han surgido crecientes amenazas relacionadas con ataques cibernéticos donde delincuentes especializados utilizan diversas tácticas para infiltrarse en sistemas y redes, comprometiendo la seguridad y privacidad de individuos, empresas y organizaciones.

Por lo tanto, detectar y prevenir estos ataques se ha convertido en una prioridad crítica para cualquier entidad conectada a la red. Entre las diversas técnicas de defensa, los grafos se han posicionado como una herramienta poderosa y eficiente para identificar patrones de comportamiento malicioso y descubrir posibles ataques cibernéticos provenientes de direcciones IP.

De acuerdo con el Instituto Nacional de Astrofísica, Óptica y Electrónica (s/f):

> Un grafo es un conjunto no vacío de objetos o entes físicos que tienen relación entre ellos. Un grafo está formado por vértices, muchas veces también llamados nodos siendo estos los que representan a los objetos), y un conjunto de arcos que representan las relaciones entre los vértices, también llamadas aristas o edges. (p. 02)

En el contexto de la ciberseguridad, los grafos se han convertido en una herramienta esencial para abordar la complejidad de las interacciones y flujos de datos en una red. Su capacidad para proporcionar una representación visual y analítica de las conexiones entre distintos elementos es invaluable para detectar patrones de comportamiento malicioso y posibles ataques cibernéticos.

En el desarrollo de la actividad integral, se optó por implementar un grafo utilizando la representación de lista de adyacencia. Esta elección se debe a los diversos beneficios que esta estructura de datos aporta en el contexto específico de la detección de ataques cibernéticos.

La representación de lista de adyacencia permitió organizar eficientemente las conexiones de las direcciones IP de origen en la bitácora. Al utilizar esta estructura, se pudo acceder rápidamente a los nodos adyacentes a cada IP, lo que facilita el cálculo de los grados de salida y el análisis de la topología de la red.

La elección de representar el grafo mediante una lista de adyacencia resultó en un código más eficiente en cuanto al uso de memoria, especialmente para grafos dispersos. Esto se debe a que únicamente se almacenó información sobre las conexiones existentes, evitando desperdiciar espacio para almacenar conexiones ausentes, como lo haría una matriz de adyacencia. Al optar por esta estructura de datos, se logró una optimización significativa en la gestión de recursos, lo que favoreció el rendimiento y escalabilidad de la aplicación, especialmente considerando que el grafo tenía 13,370 nodos y 91,910 aristas.

Por otro lado, la implementación del algoritmo de búsqueda binaria fue de vital importancia para la correcta lectura y almacenamiento de los índices y la información de las direcciones IP de cada vértice. Gracias a este algoritmo eficiente, se pudo realizar búsquedas rápidas y precisas en la lista de direcciones IP ordenada, que contuvo 13,370 elementos.

La búsqueda binaria es un algoritmo eficiente para encontrar un elemento en una lista ordenada. Su eficiencia se basa en la reducción exponencial de la búsqueda en cada paso. A medida que se itera, el rango de búsqueda se reduce a la mitad en cada iteración, lo que permite encontrar el elemento objetivo en un número máximo de iteraciones logarítmico en relación con el tamaño de la lista. Por lo tanto, Al reducir la complejidad de tiempo a O (log n), donde n es el número de elementos en el vector ordenado *ips*, el algoritmo de búsqueda binaria proporcionó una solución óptima para trabajar con una gran cantidad de datos de direcciones, facilitando la tarea de identificar conexiones y construir el grafo de forma eficiente.

De igual forma, una herramienta clave que desempeñó un papel fundamental en la solución de la Actividad Integral fue el Algoritmo de Dijkstra, el cual permite encontrar el camino más corto entre un nodo fuente y los demás nodos en un grafo con pesos no negativos.

De acuerdo con Programiz (s/f), el Algoritmo de Dijkstra utiliza la propiedad de que cualquier subtrayecto en un camino más corto también es el camino más corto entre los nodos involucrados. Utilizando esta propiedad y sobreestimando inicialmente las distancias, el algoritmo va explorando y actualizando de manera incremental los caminos más cortos desde el nodo fuente hacia los demás nodos del grafo. Este enfoque voraz (que significa que, en cada paso, selecciona la siguiente mejor solución en función de la información disponible hasta ese momento) permite encontrar eficientemente el camino más corto a todos los nodos, lo que lo convierte en una herramienta fundamental para resolver problemas de rutas y optimización en grafos ponderados.

Este planteamiento permitió encontrar el camino más corto desde el Boot Master, es decir, desde la dirección con mayor grado de salida, hacia los demás nodos en la red. Los nodos más cercanos son de especial interés para los administradores de seguridad, ya que su vulnerabilidad puede ser mayor, mientras que los más lejanos podrían requerir más esfuerzo para que el Boot Master la ataque.

Por lo tanto, el Algoritmo de Dijkstra permitió priorizar la protección y fortalecimiento de las medidas de seguridad en estos nodos, reduciendo así la superficie de ataque potencial.

En cuanto a su complejidad computacional, el Algoritmo de Dijkstra es conocido por tener una complejidad temporal razonablemente eficiente, especialmente cuando se utiliza una lista de adyacencia para representar el grafo con pesos no negativos.

Su complejidad temporal depende en gran medida de cómo se implemente la estructura de datos que representa el grafo. En este caso, al utilizar una lista de adyacencia, la complejidad temporal puede ser expresada como O ((V + E) log V), donde V es el número de nodos (vértices) y E es el número de aristas en el grafo.

La parte principal de la complejidad proviene de la exploración de los vecinos de todos los nodos y las operaciones con la cola de prioridad utilizada para mantener los nodos ordenados por distancia. Aunque la complejidad puede aumentar en grafos densos (con muchas aristas), para grafos dispersos como el que se utilizó en la Actividad Integral, el Algoritmo de Dijkstra sigue siendo eficiente y ofrece un rendimiento óptimo.

Finalmente, para lograr un correcto ordenamiento en la impresión y guardado de los grados y distancias de las direcciones IP, se hizo uso de la estructura de datos Heap. La implementación de un Heap permitió manejar eficientemente las prioridades de los elementos (en este caso, las direcciones IP) en función de sus grados de salida y distancias desde el Boot Master.

Al utilizar un Heap, fue posible ordenar de forma rápida y eficiente las direcciones IP según sus grados de salida y distancias, lo que permitió identificar de manera sencilla las 5 IPs con mayor valor y guardar esta información en el archivo correspondiente, ordenados de forma descendente. La implementación del método getTop () permitió obtener de manera eficiente el nodo con el mayor grado de salida en cada iteración, mientras que el método pop () permitió eliminar dicho nodo del Heap, asegurando que en la siguiente iteración se encuentre el siguiente nodo con el siguiente mayor grado de salida.

A continuación, se detallarán los métodos utilizados para la implementación de un Max Heap en el cual se almacenaron objetos que representan las direcciones IP en conjunto con su respectivo dato de relevancia (ya sea la distancia o su grado de salida).

Tabla 1 descripción de métodos implementados para la impresión y el guardado de la información.

push ()	La operación push () en un Heap implica la inserción de un nuevo elemento en la estructura de datos, manteniendo la propiedad de orden del Heap. En el caso de un Heap binario, la operación push () tiene una complejidad de O (log N), donde N es el número de elementos en el Heap. Durante la operación push (), el nuevo elemento se coloca inicialmente al final del Heap (es decir, en una hoja) y luego se va ascendiendo en el árbol hacia su posición adecuada en función de su prioridad en comparación con los demás elementos. Este proceso implica una serie de comparaciones y ajustes para garantizar que la propiedad de orden del Heap se mantenga intacta. La complejidad logarítmica es necesaria para asegurar que el Heap siga siendo una estructura eficiente parala obtención rápida del elemento máximo.

pop ()	La operación pop () en un Heap implica la eliminación del elemento con la máxima prioridad de la estructura. En un Heap binario, la eliminación del elemento máximo (por ejemplo, el nodo con el mayor grado de salida) se realiza de manera eficiente en O (log N), donde N es el número de elementos en el Heap. La complejidad logarítmica se debe a la necesidad de mantener la propiedad de orden del Heap al realizar una serie de intercambios y ajustes para reorganizar el Heap después de la eliminación.
getTop ()	La operación getTop () en un Heap implica obtener el elemento con la máxima prioridad sin eliminarlo del Heap. En un Heap binario, obtener el elemento máximo se realiza en O (1) tiempo, ya que el elemento máximo siempre se encuentra en la raíz del Heap. La operación getTop () es muy eficiente y no requiere reorganización o cambios en la estructura del Heap.

Como se observa, el uso de un Heap resultó ser una elección acertada para esta tarea, ya que su complejidad para la inserción, eliminación y obtención de los elementos máximos es logarítmica, lo que garantiza una eficiencia adecuada para manejar grandes conjuntos de datos y realizar un ordenamiento rápido.

En resumen, en el campo de la ciberseguridad, los grafos son herramientas esenciales para detectar patrones maliciosos y posibles ataques cibernéticos. Para la correcta realización de la Actividad Integral, la elección de una representación de lista de adyacencia para el grafo optimizó el uso de memoria en grafos dispersos. A su vez, la búsqueda binaria aceleró la lectura y almacenamiento de datos, mientras que el algoritmo de Dijkstra fue fundamental para encontrar el camino más corto desde el Boot Master, priorizando la protección de nodos cercanos. Finamente, el uso de Heap simplificó el ordenamiento y manipulación de datos para identificar las IPs más relevantes. Estas herramientas en conjunto brindaron una solución eficiente y efectiva para detectar posibles ataques cibernéticos y mejorar la seguridad en redes de direcciones IP.

Fuentes de información:

Dijkstra's Algorithm. (s/f). Programiz.com. Recuperado el 25 de julio de 2023, de https://www.programiz.com/dsa/dijkstra-algorithm

Instituto Nacional de Astrofísica, Óptica y Electrónica. (s/f). Propedéutico: Programación. Inaoep.mx. Recuperado el 25 de julio de 2023, de https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Programacion_Estructuras_Datos/Capitulo_10_Grafos.pdf