“Instituto Tecnológico y estudios superiores de Monterrey”

EVIDENCIA

“Reflexión Actividad 4.3”

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

Profesor:
Dr. Eduardo Arturo Rodríguez Tello

Alumno/a:
Ayetza Yunnuen Infante García | A01709011
Luis Carlos Rico Almada | A01252831

24/Julio/2023

Ayetza Yunnuen Infante García | A01709011

La utilización de grafos en esta situación problema es de gran importancia debido a que los grafos ofrecen una estructura de datos poderosa para modelar y analizar relaciones entre objetos *(Autho by Ouka; sf)*. En este caso, el grafo representa conexiones entre diferentes direcciones IP, lo cual es valioso para entender la comunicación entre distintos nodos en una red o sistema.

La eficiencia del uso de grafos radica en su capacidad para realizar búsquedas, análisis y cálculos complejos de manera óptima *(Autho by Ouka, s.f)*. En este caso, se utilizan algoritmos como Dijkstra y otras operaciones relacionadas con grafos:

1. Leer la bitácora y construir el grafo: Se recopilan datos de eventos y registros de la bitácora y se convierten en aristas y nodos en el grafo. El grafo permite almacenar y acceder a estos datos de manera eficiente, permitiendo búsquedas y análisis posteriores.

2. Cálculo del grado de salida de cada nodo: El algoritmo para calcular el grado de salida de cada nodo, sólo requiere contar las aristas salientes de cada nodo en el grafo. La complejidad de este cálculo es O(V + E), donde V es el número de nodos y E es el número de aristas.

3. Búsqueda del nodo con mayor grado de salida: Este algoritmo también es eficiente, ya que solo requiere encontrar el nodo con la mayor cantidad de aristas salientes. La complejidad de esta búsqueda es O(V), donde V es el número de nodos.

4. Dijkstra: Este algoritmo es utilizado para encontrar las distancias más cortas desde un nodo fuente hasta todos los demás nodos en el grafo. La complejidad del algoritmo de Dijkstra depende del método de implementación, y la implementación proporcionada utiliza una cola de prioridad para optimizarlo *(Geeks for Geeks; 2012)*. La complejidad en esta implementación es O((V + E) * log(V)), donde V es el número de nodos y E es el número de aristas.

El uso de grafos en este caso permite analizar patrones de comunicación, encontrar nodos de interés, calcular distancias entre nodos y comprender mejor la estructura y el comportamiento de la red. Los algoritmos implementados, como Dijkstra, permiten realizar cálculos complejos de manera eficiente, lo que es fundamental cuando se trabaja con grandes conjuntos de datos *(Geeks for Geeks; 2012)*.

En conclusión, el uso de grafos en una situación problema de esta naturaleza permite analizar datos de manera estructurada y eficiente, lo que facilita la identificación de patrones y relaciones importantes en sistemas complejos. Los algoritmos implementados tienen complejidades computacionales razonables y ofrecen un buen rendimiento en este contexto específico.

Luis Carlos Rico Almada | A01252831

En una situación problema como el análisis de una bitácora o log de registros, el uso de grafos puede ser de gran importancia y eficiencia para resolver distintas tareas y obtener información relevante de manera óptima. Los grafos son estructuras matemáticas que permiten modelar relaciones entre elementos, lo que los hace especialmente útiles para representar y analizar conexiones, interacciones y flujos de información entre diferentes entidades *(G. Pelayo; s.f)*.

Gracias al uso de grafos en esta situación problema fue posible representar las conexiones como arcos dirigidos entre nodos, donde cada nodo representa una IP; lo que facilita la visualización y el análisis de las relaciones entre los distintos elementos; de igual manera en la bitácora, los registros pueden contener información sobre conexiones entre dispositivos, como IP de origen y destino. Esto facilita la visualización y el análisis de las relaciones entre los distintos elementos. Por otra parte mediante algoritmos de búsqueda en grafos, como Dijkstra, es posible encontrar la ruta más corta o eficiente entre dos nodos específicos en la red; al igual que al analizar la topología del grafo, es posible identificar nodos con alto grado de conexión *(Programiz; s.f)*. De igual forma los grafos pueden ayudar en la identificación de comportamientos anómalos o inusuales en la red.

En cuanto a la complejidad computacional de los métodos implementados se pueden analizar:

1. Lectura de Archivos: La complejidad de leer un archivo es lineal, O(n), donde n es la cantidad de líneas o caracteres en el archivo. Por lo tanto, leer el archivo de registros para construir el grafo puede ser una operación eficiente si la cantidad de registros no es excesiva.

2. Grado de Salida y Mayor Grado de Salida: Calcular el grado de salida para cada IP en el grafo implica recorrer todos los registros y contar cuántas veces una IP aparece como origen. Esto requiere recorrer todos los registros una vez, lo que tiene una complejidad de O (numIncidencias) en el peor caso. Lo mismo ocurre para encontrar los 5 nodos con mayor grado de salida, lo que llevará O (numIPs * numIncidencias) en el peor caso.

3. Dijkstra: El algoritmo de Dijkstra tiene una complejidad temporal de O((V + E) log V), donde V es la cantidad de nodos (IPs) y E es la cantidad de arcos (registros). En el peor caso, si hay muchos registros en la bitácora, esto puede volverse costoso. Sin embargo, en redes pequeñas, Dijkstra sigue siendo eficiente.

En general, el uso de grafos es una herramienta eficaz para abordar problemas de este tipo. En redes muy grandes, algoritmos más sofisticados, como algoritmos de búsqueda bidireccional o algoritmos distribuidos, pueden ser necesarios para mantener una eficiencia aceptable. Además, el análisis de grafos puede ser una tarea intensiva en recursos, por lo que la optimización y el uso de estructuras de datos adecuadas son aspectos esenciales para lograr un rendimiento óptimo en situaciones de gran escala.

Referencias

Autho by Ouka (s.f) *Implementación de grafos en C++.* Techie Delight
https://www.techiedelight.com/es/graph-implementation-using-stl/

Geeks for Geeks (2012) *Find Shortest Paths from Source to all Vertices using Dijkstra's Algorithm*
https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Programiz (s.f) *Dijkstra 's Algorithm.* Parewa Labs
https://www.programiz.com/dsa/dijkstra-algorithm

G. Pelayo (s.f) *Explicación de algoritmos y estructuras de datos de grafos*. Free Code Camp
https://www.freecodecamp.org/espanol/news/explicacion-de-algoritmos-y-estructuras-de-datos-de-grafos-con-ejemplos-en-java-y-c/