ReflexAct4.3
Marco Antonio García Mendoza
A01026487

El uso de grafos en situaciones problema como la que estamos abordando, relacionada con la gestión de registros de direcciones IP y la determinación de la eficiencia del ataque del boot master, es fundamental debido a su capacidad para modelar relaciones entre entidades de manera estructurada y eficiente. Tiene varias características para determinar su eficiencia:

1. Modelado de relaciones: Los grafos permiten modelar relaciones entre entidades de una manera natural y flexible. En este caso, las direcciones IP pueden considerarse como nodos en el grafo, mientras que las conexiones entre ellas (por ejemplo, la comunicación entre diferentes IPs) pueden representarse como aristas ponderadas. Esta representación permite entender fácilmente la topología de la red y las interacciones entre las direcciones IP.
2. Eficiencia en el cálculo de rutas y análisis de la red: Los algoritmos de grafos, como el algoritmo de Dijkstra para encontrar el camino más corto o el algoritmo de búsqueda en profundidad (DFS) para determinar la conectividad, son eficientes y proporcionan resultados precisos en situaciones complejas. En este contexto, estos algoritmos pueden ser utilizados para calcular rutas óptimas entre diferentes nodos, lo que es crucial para determinar la eficiencia del ataque del boot master.
3. Complejidad computacional: La complejidad computacional de los algoritmos de grafos puede variar dependiendo de la estructura del grafo y el problema específico que se esté abordando. Por ejemplo, el algoritmo de Dijkstra tiene una complejidad computacional de O((V + E) log V), donde V es el número de nodos y E es el número de aristas en el grafo. Esto significa que su rendimiento no se degrada significativamente incluso para grafos de gran tamaño.

Imaginemos una red de computadoras en una organización, donde cada nodo representa una computadora y las aristas representan las conexiones de red entre ellas. Utilizando grafos, es posible determinar eficientemente la ruta más corta entre dos computadoras en la red, lo que es útil para optimizar la transmisión de datos o para identificar puntos críticos de la red que podrían ser vulnerables a ataques.

Algoritmos usados:

- Carga del grafo desde un archivo: O(V + E), donde V es el número de vértices y E es el número de aristas en el grafo. Esto se debe a que debemos leer cada línea del archivo una vez para construir la representación del grafo.
- Shortest Path: El algoritmo Dijkstra es comúnmente utilizado para calcular el camino más corto en grafos ponderados. Complejidad temporal: O((V + E) * log V) utilizando una cola de prioridad implementada con un heap binario.
- Boot Master: Para encontrar el vértice con el mayor grado, necesitamos calcular el grado de cada vértice y luego identificar el que tenga el mayor grado. Complejidad: O(V + E).

- Max Heap: La construcción de un montículo máximo a partir de una lista de elementos toma tiempo lineal. Complejidad temporal: O(n).
- Bubble sort: compara repetidamente cada par de elementos adyacentes y los intercambia si están en el orden incorrecto. Complejidad: O(n^2)
- Insertion Sort: construye una lista ordenada de elementos uno por uno, tomando cada elemento y colocándolo en la posición correcta en la lista ordenada. Complejidad: O(n^2)
- Merge Sort: Divide recursivamente la lista en mitades más pequeñas, ordena esas mitades y luego las fusiona para obtener una lista ordenada. Complejidad: O(n log n)
- Quick Sort: elige un elemento como pivote y particiona la lista alrededor del pivote, de modo que los elementos menores que el pivote estén a su izquierda y los elementos mayores estén a su derecha. Complejidad: peor caso O(n^2) y su mejor caso O(n log n)
- Búsqueda Lineal: busca secuencialmente cada elemento en una lista hasta encontrar el elemento buscado o llegar al final de la lista. Complejidad: O(n)
- Búsqueda Binaria: Compara el elemento buscado con el elemento en la mitad de la lista y descarta la mitad no necesaria del arreglo en cada paso. Complejidad: O(log n)
- Búsqueda en árboles binarios de búsqueda (BST): utiliza la estructura de un árbol binario de búsqueda, donde los nodos están organizados de manera que los elementos menores están a la izquierda y los elementos mayores están a la derecha. Complejidad: En promedio O(log n), pero en el peor caso podría ser de O(n) si el árbol no está balanceado.

Referencias:

Bryan Salazar López. (2019, June 12). Algoritmo de Dijkstra. Ingenieria Industrial Online; Ingenieria Industrial Online. https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/algoritmo-de-dijkstra/

MBR Master Boot Record. (2023, February 22). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/mbr-master-boot-record/

Graph Data Structure And Algorithms. (2024, January 17). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Estefania Cassingena Navone. (2020, September 28). Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction. FreeCodeCamp.org; freeCodeCamp.org. https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/#:~:text=Dijkstra's%20Algorithm%20finds%20the%20shortest,node%20and%20all%20other%20nodes

Binary Heap. (2014, November). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/binary-heap/

¿Qué es un algoritmo de ordenamiento? (2023). Platzi. https://platzi.com/tutoriales/1469-algoritmos-practico/6031-que-es-un-algoritmo-de-ordenamiento/

Guillermo, J. (2018, June 23). ¿Qué es la complejidad algorítmica y con qué se come? Medium; Medium. https://medium.com/@joseguillermo_/qu%C3%A9-es-la-complejidad-algor%C3%ADtmica-y-con-qu%C3%A9-se-come-2638e7fd9e8c