Instituto Tecnológico y de Estudios Superiores de Monterrey

TC1031.850

Programación de estructuras de datos y algoritmos fundamentales

Actividad

Reflexión actividad integradora 4

Profesor

Eduardo Arturo Rodríguez Tello

Presenta

Arturo Azael Godinez Rodriguez | A01641179

Fecha:

15/06/2023

Arturo Godinez:

La realización de este trabajo de tuvimos que utilizar el almacenamiento de una bitácora en una estructura grafo. Los grafos se utilizan para representar relaciones complejas entre objetos, ejemplos de ellos son redes sociales como Facebook que lo utiliza para recomendarte personas para amistades, que realmente lo que usan son que lo usuarios lo representan como nodos, y las relaciones de amistad que tienen entre ellos lo representan como aristas.

De entro de los grafos existen varios tipos, pero de los más comunes son:
Grafos no dirigidos: En estos los grafos no tienen dirección, un ejemplo podría ser que tengo nodos A y B que están conectados por un arista pero esta puede ir tanto de A-B como de B-A.
Grafos dirigidos: En este caso los grafos tienen una dirección, la cual nos dice que tiene un nodo que va de A-B, y este caso no necesariamente hay una arista que va de B-A
Grafos ponderados: En este caso las aristas tienen un peso o valor, el cual lo podemos ver como una distancia, costo, probabilidad, etc.

La importancia y eficiencia del uso de grafos en situaciones como esta radica en su capacidad para representar de manera clara y concisa las relaciones entre las cosas en un problema. Los grafos permiten identificar patrones y estructuras en los datos de manera eficiente, facilitando la resolución de problemas complejos y la toma de decisiones basadas en la información disponible.

En el contexto de la seguridad de redes, la utilización de grafos puede ayudar a identificar nodos críticos, como el "boot master", y a predecir posibles ataques o vulnerabilidades en la red. Esto permite a los expertos en seguridad tomar medidas preventivas y mejorar la robustez de la red.

Además, la implementación de grafos en C++ es eficiente y fácil de usar, lo que permite resolver problemas complejos de manera rápida y eficiente. Los algoritmos de búsqueda y optimización, como Dijkstra, pueden ser fácilmente implementados en C++ y aplicados a problemas de seguridad de red.

Ya cuando hablamos acerca de las complejidades empleadas en este trabajo son:
Split: La complejidad de este método es O(n), ya que recorre la cadena de entrada desde el inicio hasta el final una vez.
FindIpIndex: La complejidad de este método es O(1), ya que solo busca el índice del nodo de la dirección IP en la estructura de datos mapIP y devuelve el índice si se encuentra.
LoadGraph: La complejidad de este método es O(n^2), ya que recorre la cadena de entrada n-1 veces y, en cada iteración, puede realizar operaciones en las listas de adyacencia y mapas.
PrintGraph: La complejidad de este método es O(n^2), ya que recorre la lista de adyacencia para cada nodo y realiza operaciones en la lista enlazada.
Dijkstra: La complejidad de este método es O((|V|+|E|)*log(|V|)), ya que utiliza una cola de prioridad para almacenar los nodos a visitar y realiza operaciones en los mapas y listas de adyacencia. La complejidad de la cola de prioridad es O(|V|log|V|) en el peor de los casos, y la iteración a través de los nodos y sus vecinos tiene una complejidad de O(|E|). Entonces, la complejidad total es O((|V|+|E|)*log(|V|)).

En resumen, la mayoría de los métodos en el código tienen una complejidad computacional de O(n) o O((|V|+|E|)*log(|V|)). Estas son complejidades comunes en la manipulación de grafos y algoritmos de búsqueda, como anteriormente mencionado el algoritmo de Dijkstra.

Referencias:

GeeksforGeeks. (2023). Graph Data Structure And Algorithms. *GeeksforGeeks*. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Olawanle, J. (2023). Big O Cheat Sheet – Time Complexity Chart. *freeCodeCamp.org*. https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/

GeeksforGeeks. (2022). Computational Graphs in Deep Learning. *GeeksforGeeks*. https://www.geeksforgeeks.org/computational-graphs-in-deep-learning/