ReflexAct5.2 Didier
Didier Aarón Ricardo Hernández Ferreira A01663817
En el contexto de este problema, donde necesitamos almacenar y recuperar información de direcciones IP junto con su cantidad de salidas y entradas, las tablas hash son una estructura de datos fundamental debido a su eficiencia en la búsqueda y recuperación de datos.
Las tablas hash proporcionan un acceso rápido a los elementos almacenados. Con una función hash bien diseñada, la búsqueda y recuperación de información se puede hacer en tiempo constante en promedio, lo que significa que la búsqueda no depende del tamaño de la tabla.
Sin embargo, las colisiones pueden afectar significativamente la eficiencia de la tabla hash. Cuando dos o más elementos se asignan a la misma posición de la tabla hash, se produce una colisión. En este caso, es importante implementar técnicas de resolución de colisiones, como la prueba lineal, la prueba cuadrática o la resolución por encadenamiento, para garantizar un rendimiento óptimo.
La complejidad computacional de las operaciones en una tabla hash depende en gran medida de la calidad de la función hash y del método de resolución de colisiones utilizado. En general, la inserción, eliminación y búsqueda en una tabla hash tienen una complejidad de tiempo promedio de O(1), pero esto puede aumentar si hay muchas colisiones y se utiliza una técnica de resolución de colisiones ineficiente.
Si el número de colisiones en la tabla hash aumenta, la eficiencia de las operaciones también puede disminuir. Las colisiones pueden provocar un aumento en el tiempo de búsqueda y recuperación, lo que lleva a un peor rendimiento general del algoritmo. Por lo tanto, es crucial diseñar la función hash de manera que distribuya los elementos de manera uniforme en la tabla hash y elegir una técnica de resolución de colisiones que minimice las colisiones.
Complejidades computacionales:
hash(std::string ip): Complejidad: O(n), donde n es la longitud de la cadena ip. Esto se debe a que esta función recorre cada carácter de la cadena para calcular la suma de los valores ASCII.
hashWithQuadraticProbing(std::string ip, int attempt): Complejidad: O(1) en promedio. Aunque el método de resolución de colisiones utiliza un enfoque cuadrático, el tamaño de la tabla hash es constante, lo que hace que el tiempo de ejecución promedio sea constante.
insert(IPInfo* ipInfo): Complejidad: O(1) en el mejor caso y O(n) en el peor caso, donde n es el tamaño de la tabla hash. En el mejor caso, la inserción se realiza en una posición vacía de la tabla hash sin colisiones. En el peor caso, todas las posiciones de la tabla están ocupadas, lo que requiere una búsqueda lineal completa para encontrar un espacio vacío.
getInfo(std::string ip): Complejidad: O(1) en el mejor caso y O(n) en el peor caso, donde n es el tamaño de la tabla hash. En el mejor caso, la IP buscada se encuentra en la posición calculada por la función hash sin colisiones. En el peor caso, se deben recorrer todas las posiciones de la tabla hash debido a colisiones.
getTotalCollisions(): Complejidad: O(1). Este método simplemente devuelve el valor de una variable miembro, por lo que su complejidad es constante.

Referencias:
Hash Collisions Explained. (2022, October 4). Freeman Law.
https://freemanlaw.com/hash-collisions-explained/

Separate Chaining Collision Handling Technique in Hashing. (2015, July 31). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/separate-chaining-collision-handling-technique-in-hashing/
Hash Table Data Structure. (2023, April 24). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/hash-table-data-structure/
Quadratic Probing in Hashing. (2020, May 10). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/quadratic-probing-in-hashing/