Jose Manuel Martinez Morales

A01734279

16 de julio de 2023

Actividad Integral de estructura de datos lineales

programación de estructuras de datos y algoritmos fundamentales

Jorge Enrique González Zapata

Analisis de Complejidad

isEmpty

Verifica si la lista está vacía comparando un puntero y un nullptr. Esta función es constante al solo realizar operaciones básicas, por lo que su complejidad es de O(1).

addNode

Agrega un nuevo nodo al final de la lista, como no es necesario recorrer la lista para añadir un nuevo Nodo al final es una función constante, por lo que su complejidad será de O(1).

sortedRecords

Esta función almacena los datos ordenados por IP en un nuevo archivo, como recorre todos los valores para poder escribirlos en un nuevo archivo, es una función lineal y tendrá una complejidad de O(n).

swap, partition y quickSort

Estas 3 funciones son parte del algoritmo de ordenamiento Quicksort por lo tanto su complejidad es O(n log n), primero se elige un pivote en un valor medio y después de divide el vector hasta que el vector quede ordenado.

sortRecordsByIP

Utiliza Quicksort para ordenar las ips de manera ascendente, por lo cual su complejidad es la misma que Quicksort, O(n log n).

printRecordsByIP

Esta función imprime las ips que están dentro del rango de búsqueda del usuario, lee cada uno de los datos para encontrar lo que se pide, por lo tanto, es una función lineal y tiene una complejidad de O(n).

FindIP

Esta función se utiliza para separar cada registro en valores individuales para acceder más fácilmente a ellos y fue de ayuda a la hora de ordenar las IP ya que ocupaban el lugar 3, esta función utiliza un for y es lineal, por lo que la complejidad será de O(n).

readFile

Esta función lee todos los datos del archivo y los implementa a la lista, por lo cual es una función lineal y su complejidad será de O(n).

Reflexión

Una lista ligada es una estructura de datos lineal la cual nos permite almacenar elementos enlazados a través de los punteros, las listas ligadas tienen un tamaño dinámico y son eficientes a la hora de añadir y eliminar datos de la lista.

Para este proyecto decidí que la mejor manera de realizarlo era a través de una lista doblemente ligada por las ventajas que ofrece como:

La facilidad para la insertar los datos a la lista de manera efectiva y fácil, ya que las operaciones para añadir datos a la lista tienen una complejidad constante, con esto nuestro programa tendrá un rápido que no será afectado por el crecimiento de los datos.

Otra forma en la cual la lista ligada es útil es a la hora de leer los datos de la bitácora, ya que con la función addNode pude almacenarlas en la lista de forma sencilla.

De igual manera la implementación de una lista doblemente ligada nos permite buscar y filtrar los datos de manera más fácil, esto lo utilizamos en el método printRecordsByIP la cual recorre la lista de manera eficiente y muestra las Ips dentro de un rango especifico.

La lista doblemente ligada también tiene algunas desventajas que pueden afectar nuestro código algunas desventajas que se presentan en la lista doblemente ligado son:

La lista doblemente ligada requiere almacenar un puntero que apunta al siguiente nodo y al nodo anterior, lo que necesita más memoria en comparación a un vector o una lista ligada.

El algoritmo de ordenamiento de Quicksort en una lista doblemente ligada puede ser un poco mas ineficiente que otros algoritmos de ordenamiento debido al mayor numero de enlaces durante el ordenamiento.

El utilizar una lista doblemente fue útil en esta situación debido a su eficiencia al añadir nodos y recorrer datos de la bitácora. De igual manera facilito la implementación del ordenamiento por IP y su búsqueda por rango. Sin embargo, es importante considerar las desventajas de la lista doblemente ligada, como puede ser el rendimiento inferior en algunas operaciones.