ReflexFinTC1031 - Reflexión

Fernando Doddoli Lankenau - A00827038

Programación de Estructuras de Datos y Algoritmos Fundamentales

Grupo 14

Prof. Luis Humberto González Guerra

Noviembre 28 2020

Introducción

Una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. En esta reflexión analizaremos la importancia y eficiencia del uso de diferentes tipos de estructuras de datos para resolver problemas de diferentes naturalezas. Estos incluyen: algoritmos fundamentales de ordenamiento y búsqueda, una lista doblemente encadenada, árbol binario de búsqueda, grafos, y tablas de hashing.

Actividad 1.3

En la actividad 1.3 se nos dió un archivo con diferente información representando accesos direcciones IP 's de computadoras con el objetivo de obtener los registros correspondientes para un rango de fechas. La información dada por cada acceso incluía el mes, día, hora, dirección IP y razón de falla.

El primer paso para obtener los registros correspondientes para un rango de fechas fue ordenar la información por fecha para realización de las búsquedas. Para hacer esto creamos una clase llamada registro y guardamos la información del archivo en un vector de tipo registro. Además, para cada elemento de tipo registro creamos su clave con el método de crearClave, la cual se llama dentro del constructor de la clase registro. Finalmente, para ordenar los datos en base a sus claves, utilizamos el algoritmo de ordenamiento de tipo merge con una complejidad de tiempo de O(n*logn). Después, para unir estos datos utilizamos un método llamado unir en nuestro programa de complejidad O(n). Para hacer las comparaciones en la función de merge hicimos uso de operator overloading en una nuestra clase de registro con el fin de poder comparar las claves directamente. Utilizamos este algoritmo porque como estamos trabajando con una gran cantidad de datos, este es más eficiente que el algoritmo de ordenamiento de intercambio y burbuja.

Finalmente para buscar registros correspondientes en un rango de fechas le pedimos al usuario la fecha inicial y final y después llamamos a la función de búsqueda. Esta función agarra las claves de la fecha inicial y final llamando al método de orden clave. Después llama el método de busquedaBinaria utilizando como parámetro fecha inicial para obtener posición inicial y fecha final para obtener posición final respectivamente. Finalmente, el método de búsqueda checa si la posición inicial y final existen (no son -1) y si sí existen hace un for

loop en donde despliega los registros desde la posición inicial a la posición final. Utilizamos estos dos métodos porque son eficientes, teniendo una complejidad de tiempo de O(n) y O(logn) para los algoritmos de busqueda y busquedaBinaria respectivamente.

Me gustó mucho poder aplicar métodos de ordenamiento y búsqueda para una situación problema de verdad. Creo que es sumamente importante el saber dominar estos algoritmos al igual que entender cómo funciona para poder tener la intuición de cómo resolver este tipo de problemas de la forma más eficiente.

Actividad 2.3

Para la actividad 2.3 se nos pidió leer un archivo y almacenar los datos en una lista doblemente encadenada con el objetivo de permitir al usuario buscar un rango de direcciones IP y mostrarlas de forma ascendente y almacenar el resultado de ordenamiento en un nuevo documento.

No obstante, es importante analizar si el uso de una lista doblemente encadenada es la más adecuada para este problema. Hay personas que argumentan que una lista doblemente encadenada es mejor que una lista encadenada para el objetivo de buscar información. Ellos tienen razón ya que el tiempo de complejidad para buscar es O(n). Sin embargo, en este problema el tener que ordenar un archivo con muchos datos hace que el programa sea muy ineficiente gracias a la complejidad del método de sortIP que es O(n^3). Esto es importante porque significa que una lista doblemente encadenada no es la mejor solución para resolver esta situación.

Un factor muy importante en la determinación de la efectividad del uso de una lista doblemente encadenada para solucionar este problema fue la funcionalidad de ordenar información de forma ascendente. Analizando este método se puede observar la complejidad de él y su ineficiencia–O(n^3). De esta manera, después de analizar los pros y contras de una lista doblemente encadenada para resolver este problema se puede concluir que no es una solución adecuada.

Actividad 3.4

La implementación de un BST para resolver un problema de la naturaleza de la actividad 3.4 no es eficiente. Esto se debe a que los IPs más accedidos tendrán los valores de key más grandes, de tal forma que serán los últimos nodos en el árbol binario de búsqueda. Esto es importante porque significa que para buscar al nodo con el IP más accesible el tiempo de complejidad será O(n), lo cual no es muy eficiente. Sin embargo, este problema se puede arreglar si decidimos resolver una situación problema de esta naturaleza a través de un splay tree envés de un BST. Un splay tree es otro tipo estructura de datos, pero a diferencia de un BST, este es ideal para problemas en dónde tenemos que acceder rápidamente los datos más utilizados o buscados. Esto se debe a que después de buscar un dato, el dato buscado se almacena en el raíz, lo que permite al programa regresar ese dato en un tiempo de complejidad de O(logn) si se busca de nuevo. Esto es importante porque significa que para un problema de esta naturaleza, el uso de un splay tree es más eficiente que el de un BST. Lo cual nos permitiría determinar si una red está infectada o no.

Una forma de detectar si una red está infectada es almacenando la información importante a través de una estructura de datos, como un splay tree, que nos permite hacer

búsquedas eficientes para detectar si existen accesos maliciosos. Por ejemplo, un ataque cibernético DDoS, donde muchos bots viniendo de varios dispositivos conectados al internet hacen que los servidores de un producto o servicio dejen de funcionar correctamente a través de una avalancha de accesos, se puede detectar fácilmente a través de un splay tree gracias a que los dispositivos causando el ataque podrán ser identificados rápidamente. Esto se debe a que, si existe un ataque cibernético de bots, los accesos maliciosos serán los que están accesando la red más veces y estarán en los niveles más altos del árbol. Esto es importante porque significa que si queremos detectar si una red está infectada, lo podemos hacer de una manera rápida y eficiente.

Actividad 4.3

La implementación de un Grafo dirigido para resolver un problema de la naturaleza de la actividad 4.3 es eficiente, dado que podemos almacenar los datos en una lista de adyacencia organizada por dirección de IP origen utilizando un unordered_map. De esta manera, podemos almacenar el IP de origen como el key value y su respectivo índice, referente a la lista de adyacencia, y outdegree en un pair como el map value del unordered_map. Todo esto es importante porque hace a nuestro programa muy eficiente ya que la complejidad de almacenar los datos en una lista de adyacencia es O(|V|+|E|)y la utilización de un unordered_map es O(1). Además, aparte de que utilizar un Grafo como estructura de datos es una estrategia eficiente para resolver problemas de esta naturaleza, hacer uso de un unordered_map, donde un pair es su map value que tiene como el número de out degrees su segundo valor, nos permite fácilmente determinar cuales IPs son los que están infectando la red.

Una forma de detectar cuales IPs son los que están infectando la red es analizar qué IPs tiene el mayor número de out degrees, es decir, cuales IPs son los que están apuntando a más IPs. Por ejemplo, un ataque cibernético DDoS, donde muchos bots viniendo de varios dispositivos conectados al internet hacen que los servidores de un producto o servicio dejen de funcionar correctamente a través de una avalancha de accesos, se puede detectar fácilmente a través de un Grafo dirigido almacenado en una lista de adyacencia gracias a que los nodos causando el ataque pueden ser identificados rápidamente. Esto es gracias a que si existe un ataque cibernético de bots, los accesos maliciosos serán los nodos que tengan el mayor número de out degrees, es decir, los que estén apuntando a los mayores números de nodos. Esto es importante porque significa que si queremos detectar qué IPs están atacando la red, lo podemos hacer de una manera rápida y eficiente con una complejidad O(n).

Actividad 5.2

Para esta actividad se nos pidió utilizar una tabla hash para almacenar datos y dado un IP desplegar el valor asociado a él desplegando la información de una manera adecuada. Esto lo pudimos hacer mediante el uso de un unordered_map para representar nuestra tabla de hash donde usamos el IP como el key value y el valor como un vector de tipo struct llamado Registro que almacenó el resumen del IP. Además, hicimos uso de la función de C + + llamada pushback() al momento de leer el archivo para meter los datos a el unordered_map adecuadamente. Esto es importante porque nos permitió resolver el problema de una forma muy eficiente. Por ejemplo, nuestros métodos para cargar los datos y almacenarlos en un

unordered_ map, y desplegar el número de accesos que tuvo la IP ingresada son de complejidad lineal. Además, nuestro método para desplegar el resumen del valor asociado al IP es constante. Todo esto es muy importante porque demuestra que utilizar un unordered_map como tabla hash es una excelente solución para resolver este tipo de problemas en donde necesitamos almacenar y buscar información rápidamente.

Conclusión

Después de haber analizado los diferentes tipos de estructuras de datos es importante reconocer que no todas son igual de eficientes para resolver los mismo problemas. Algunas estructuras de datos son mejores para resolver un tipo de problema que otras. Por ejemplo, los grafos son una excelente herramienta para almacenar y buscar información, como lo utilizan los motores de búsqueda. Por el otro lado, el uso de un splay tree es una excelente estructura de datos para encontrar quién está atacando una red en el caso de un ataque DDoS. Y como visto anteriormente, las tablas de hashing nos permiten almacenar y acceder información con mucha facilidad y eficiencia. Todo esto es muy importante porque demuestra que para resolver un problema hay que entender bien qué herramientas podemos utilizar para solucionarlo de la mejor manera posible.