Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación

Luis Fernando Tarango Felix A00827678

1.- ¿Cuáles son las más eficientes?

Depende de lo que se quiera realizar los algoritmos de búsqueda y ordenamiento un poco mas rudimentarios que utilizamos en la actividad 1.3, los cuales dependían de un array para almacenar sus datos y para lograr una mayor eficiencia las funciones eran recursivas o iterativas, sin embargo todos estos procesos se encuentran separados, esto nos dio resultados similares a la 2.3 con la double linked list, pues se utilizaban similares algoritmos para búsqueda y ordenamiento, pero tiene la añadida ventaja que gracias a sus apuntadores el travesar la lista de manera progresiva o regresiva es mucho mas fácil, ayudando principalmente a las funciones de print, una vez ya estén ordenados los datos.

Por otro lado, tenemos los arboles binarios con la gran ventaja de que los datos se van ordenando conforme son ingresados a la lista, y la manera en que se manejan sus ramas también puede ayudar a la hora de buscar un dato, además de que nos da una variedad de ordenes de prints que podemos realizar por medio de funciones recursiva de alta eficiencia.

Sin embargo para esta situación problema de bitácoras, las dos estructuras que sobre salen son grafos y hash tables, los grafos debido a que pueden ser altamente eficientes si hechas con un Unordered Map, a la vez que debido a que puede representar de manera mucho más fácil interconexiones, como lo fue en este caso con los fan outs y fan ins de las ips, es seguro decir que esta es una estructura de alta utilidad para almacenar datos, en especial si se desea que existan arcos entre ciertos datos, mientras que las hash tables creadas con un unordered map demostró ser una de las estructuras más sencillas de programar, además de que es capaz de realizar búsquedas en una complejidad de O(1), tan solo dándole la llave que se desea insertar, aunque desafortunadamente sufre a la hora de imprimir, donde se tiene que limitar a un método lineal, debido a los rápido que se puede implementar, además de poder lograr un código de complejidad O(n) en la actividad 5.2, lo hace una de las estructuras más eficientes.

2.- ¿Cuáles podrías mejorar y argumenta cómo harías esta mejora?

En cuanto a las actividades integradoras que realizamos hay varias mejoras que podríamos implementar. En la 1.3, mientras que optamos por un algoritmo estilo merge sort que termino teniendo una complejidad de O(n log n), para la búsqueda utilizamos un método secuencial, cuando pudimos implementar uno binario para reducir la complejidad de O(n) a O(log n). En la actividad 2.3, utilizamos un algoritmo bubble sort para ordenar los datos, sin embargo, con un poco mas de tiempo y trabajo se pudo haber adaptado

un merge sort que funcionase con una lista doblemente ligada que hubiera disminuid significativamente el tiempo que le tomaba correr al programa, además de que el bubblesort implementado era fallido, pues comparaba datos que ya estaban ordenados.

Para la actividad 4.3 creamos un grafo utilizando una lista de adyacencia, que mientras esto es de gran atildad si deseáramos representar los fan outs y fan ins de cada ip en una lista para imprimir, tenía una complejidad de O(n^2), para resolver este problema de complejidad pudimos utilizar simplemente un unordered map para reducir significativamente la complejidad y realizar el conteo de los fan outs y fan ins por medio de iteradores.

El hash table que realizamos era casi instantáneo y con una complejidad de O(n) en todas sus funciones, por lo que una optimización sería difícil de encontrar y probablemente no muy significativa debido a que las funciones de carga y print son O(n) y con pocas posibles optimizaciones, al menos con los conocimientos aprendidos en este curso.