Reflexión Actividad 1.3

En esta actividad realizamos un código de organización y de búsqueda para filtrar una gran cantidad de datos. Lo que realizamos fue implementar el método de ordenamiento Selección Directa ya que era el método menos complejo y que consideramos fue de los más eficientes para organizar los datos con un orden(n^2). Para la búsqueda utilizamos el método de Búsqueda Binaria con complejidad O(log n). Esto es de gran importancia ya que para que funcione de una manera óptima es importante evitar una gran cantidad de repeticiones, como lo haría una de orden lineal. Ya que utilizamos este método, hubo un problema. Al haber datos repetidos, no sabíamos si el método fuera a tomar el primer dato, o el último en el orden. Para esto hicimos una condición de que al conseguir el número, había uno igual después o antes, lo considerábamos y repetimos esto hasta que el número fuera diferente. Esto solo para validar que obtuvieramos todas las fechas repetidas dentro del rango establecido.

Tomando en cuenta el main, lo primero que realizamos fue leer el archivo txt y lo convertimos en un vector de tipo datos(el cual lo definimos con una estructura). Tras tener este vector, generamos una clave para ordenar las fechas. Esta consiste en formar el mes y día a un entero que pudiéramos usar para después ordenarlo. Teniendo esto listo, lo único que faltaba era preguntar el rango de fechas a desplegar. Creamos un vector resultante en donde guardaremos los datos de dado rango. Tuvimos unos problemas en esta parte pero cuando encontramos la solución, fue mucho más fácil de lo que esperábamos. Teniendo esto solo llamamos la función de imprimir para desplegar el vector resultante dentro de la consola. Repetimos esto dentro de un loop cuantas veces lo desee el usuario. Una vez terminando con las búsquedas, se genera un archivo txt con todos los datos de la bitácora original de manera organizada. Es con esto con lo que termina el código.

Fue buena práctica esta actividad ya que pudimos implementar lo visto en clase en un caso muy particular y retador. La implementación de la lectura de archivo en lo personal es un poco complejo, pero considero que pude desarrollarlo de una manera más eficiente.

Reflexión Actividad 2.3

Tuvo una gran importancia el uso de listas doblemente ligadas para el desarrollo de esta actividad en particular. La lista por si sola fue de gran ayuda para almacenar todos los datos dentro de la bitácora. Mientras que no se puede utilizar un índice como en un vector, fue de utilidad para poder ordenar los datos. Sin embargo, el ordenamiento por el método merge era la una manera de de ordenar los datos dentro de esta lista por la limitación que se tiene que ir dato por dato. Esto resulta en un tiempo extremado en la realización del ordenamiento de todos los datos dentro de la bitácora. Mientras que si es funcional, el orden de O(n^2) hace el programa ineficiente, y en mi opinión, no lo utilizaría para una gran cantidad de datos como en este caso.

Fuera de esto, la mayor parte del programa de la lista doblemente ligada era directo y fácil de entender. Solo en el método de la búsqueda secuencial, tuvimos que

ingeniar un método para validar si se encontraba el dato que insertaba el usuario dentro de la lista. Lo que hicimos fue que si no estaba el dato, comparábamos que el dato previo fuera menor al dato insertado y que al mismo tiempo el dato posterior fuera mayor al dato insertado. De esta manera podríamos establecer la dirección desde la cual podríamos obtener el rango para la búsqueda del usuario.
Dicho esto no fue tan fácil, una cosa es regresar el dato, pero si es el primer dato, regresa nullptr, lo que resulta en un error. Del mismo modo por si era el último elemento dentro de la lista. Por esta razón, creamos otra condición para validar que si el número insertado por el usuario era menor al primer número de la lista, tomara el primer número en vez de retornar un error. Del mismo modo, si el usuario inserta un número mayor al último elemento de la lista, retornamos como límite el último valor. Fueron casos que tuvimos que considerar para asegurarnos de que no hubiera errores dentro de la búsqueda.

En general fue un proceso de mucho pensamiento en un inicio, de cómo íbamos a implementar los conceptos de la lista doblemente ligada con el caso de los ip. Considero que fue de gran importancia el apoyo de Lucas, mi compañero para poder encontrar las soluciones de manera más rápida y en conjunto. Siempre íbamos a la par para irnos apoyando en las dudas y tener en claro lo que realiza el programa.

Reflexión Actividad 3.4

En un comienzo, el realizar este programa tuvo sus retos, sin embargo considero que con los temas vistos en clase, y con la ayuda en equipo fuimos capaz de elaborar este programa de manera eficiente y rápida. Considero que el mayor reto fue guardar las direcciones de ip junto con un acomulador de cuantas veces se repetían dentro de la bitácora. Aparte de esto, el imprimir los datos en orden descendente de mayor a menor fue un proceso muy sencillo con el método de “InOrder Converso”. En si, esta actividad no fue de las más retadoras, incluso creo que esta reflexión es más compleja. Pero creo que es más que nada porque poco a poco aprendo a relacionar todos los conceptos y la lógica que utilizamos en clase.

En base a la situación problema, considero que hay una similitud en base a cómo se expande una red infectada, con relación a un modelo de BST. No estoy seguro aún de cómo determinar si una red está infectada o no con base a esto. En especial considerando muchos otros factores de este caso. Sin embargo, con un modelo BST, se puede buscar y determinar las propiedades de diferentes nodos, ayudando a identificar los nodos, o los dispositivos en este caso, que están infectados.

En conclusión, este modelo de árbol binario de búsqueda es de mucha ayuda para lo que dice, búsqueda. En especial con base de datos muy grandes ya que la complejidad por lo general es de O(log n), lo que lo hace más eficiente que un O(n).

Reflexión Actividad 4.3

Esta actividad me gusto mucho. En sí estuvo simple implementar el documento y las variables. Ya con las actividades previas se ha vuelto algo muy sencillo. Sin

embargo, me gusto mucho el poder haber implementado dos conceptos nuevos que no habíamos visto previo a la actividad. Estos están el “unordered_map” y el “pair”. La verdad es que tras haber analizado el material que el profesor nos proporcionó, pudimos tener un entendimiento general del funcionamiento. Sin embargo, al investigar más a fondo cómo estos conceptos funcionan, fuimos aprendiendo cómo manipularlos y accederlos de manera más eficiente. Una vez hecha la investigación, solo fue cuestión de implementar la lógica del programa para lograr que imprimiera las direcciones ip.

En si me gusto mucho esta actividad, ya que mi hermano me ha hablado mucho de los “hash-maps” y el poder empezar a implementarlos me emociona. Se que los “hash-maps” ayudan a que corra de manera más rápida un programa en comparación con un vector o una lista. Y precisamente por eso considero que el poder empezar a implementar este tema es de gran importancia. Ahora bien, ¿por qué es esto tan importante? Porque se que el dominar este tema, será de gran ayuda en el momento de las entrevistas de trabajo. En el poder implementar programas que puedan recopilar datos o hacer procedimientos de manera rápida, considerando bases de datos muy muy grandes.

En si veo esta actividad como mi introducción a los temas avanzados de programación. En sí, este concepto me emociona.

Reflexión Actividad 5.2

Esta última actividad que realizamos en equipo, fue de las más sencillas. Considerando que implementamos el uso de “unordered_map” en la actividad 4.3, el implementar la solución para esta actividad fue relativamente fácil, o por lo menos estoy aprendiendo por lo que no es tan complicado. Ahora bien, si hubo retos en el desarrollo de la actividad.

Primero que nada, implementamos un *Struct* dentro del programa llamado resumen, que tendría los atributos que se piden como características del *Resumen* (Fecha y mensaje). Lo complicado en sí llega cuando definimos un *unordered_map*, con un vector dentro de tipo de *Resumen*. Yo dije, “más complicado no puede ser, es simplemente utilizar los atributos del Struct”. Y la verdad fue que no se complicó mucho directamente. En si ya sabíamos obtener los datos del archivo, y definirlos a las variables del struct. El primer problema lógico fue el entender cómo hacer un *push_back* de un dato del vector dentro del *unordered_map*. Intentamos unas diferentes maneras de definir esto hasta que lo logramos, resultó ser mucho más fácil de lo que creíamos. En sí todo bien hasta ahí, yo sentía que todo iba bien y que terminaríamos rápido.

Después entramos a la impresión del ip del usuario y del Resumen. Ahora ya teníamos un método de imprimir todos los datos del *Unordered_map* junto con su key, que en este caso sería el ip. Lo bueno de este método era que podríamos utilizar los métodos first y second para obtener los datos. El problema es que al usar este método no podemos especificar solo un ip como lo indica en las instrucciones, tenía que

recorrer todo el *unordered_map*. Dicho esto, nos pasamos, pasé como dos horas buscando cómo implementar first y second dentro de las condiciones actuales del programa para poder correrlo de una manera que yo entendiera. Después de la busqueda me meti a mi codigo, y lo intente de redactar con las condiciones adecuadas de una manera que me tuviera sentido a mi sintaxis. Hubo errores, pero poco a poco logré llegar a la sintaxis correcta de cómo se desvía de implementar estas condiciones. Todo esto fue dentro de la función *printSpesific*, que dado a que se inserta una opción del usuario, se tiene que hacer con base a esta opción.

En conclusión, tuvo sus obstáculos este programa, pero con determinación, logre resolver este problema de una manera directa y eficiente. Siento que aprendí conceptos importantes de los *Unordered_map* en esta actividad, y me abre el mundo ya que facilita mucho los procesos involucrados en este tipo de análisis.