# INFORME TRABAJO PRACTICO 2

- Nuestra estructura para la implementación:

Decidimos implementar una estructura Intervalo, la cual cuenta con dos double que representan ambos extremos de un intervalo, además de Interval que representa un puntero a una estructura Intervalo.

Para la estructura INodo decidimos crearla con los siguientes datos:

- Puntero a Intervalo (Interval) definido anteriormente
- Un double que representa el máximo extremo derecha que se encuentra en los nodos hijos o en el mismo nodo correspondiente
- Dos punteros a INodo que representan los hijos izquierdo y derecho de un nodo
- Un entero que representa la altura del respectivo nodo

ITree representara un puntero a INodo

- Motivo de introducir el dato de la altura de un nodo:

La decisión surge de la necesidad de mantener balanceado el Árbol AVL que estamos representando, cada vez que insertamos o eliminamos un nodo al árbol, tenemos que actualizar el balance del árbol y determinar si es necesario realizar rotaciones, y para ello necesitamos actualizar y saber la altura de cada nodo del árbol.

- Introducir las rotaciones y armar un pseudocódigo para intervalos:

Uno de nuestros problemas fue a la hora de pensar las rotaciones y su introducción en el código, para solucionar esto nos nutrimos de una fuente externa (1). Luego armamos un pseudocódigo que se adapte a lo que necesitamos. Este constaba de:

1. Guardar: Se guarda en una variable la información del nodo izquierdo (rotación derecha) o derecho (rotación izquierda), y además el nodo derecho (rotación derecha) o izquierdo (rotación izquierda) de este mismo.
2. Intercambiar nodos: Se intercambia la posición de los nodos correspondientes para así lograr un árbol balanceado
3. Actualizar altura: Se actualiza la altura de los nodos intercambiados
4. Reestablecer máximo extremo derecho: se establece el máximo derecho de cada nodo para cumplir con la implementación.

- Problema a la hora de eliminar un nodo y actualizar el máximo extremo derecha

Nos surgió un problema a la hora de eliminar un nodo y actualizar el máximo extremo derecha de los nodos que se ven afectados por dicha eliminación. Como a la hora de insertar no había problema, lo contrario paso cuando teníamos que eliminar un nodo y actualizar el máximo extremo derecha de sus ascendientes. Por ejemplo, en el caso de que el nodo fuese una hoja y a la vez sea el máximo extremo derecha de su padre, a la

hora del llamado recursivo de la función eliminar, no teníamos un caso contemplado que actualizara el nodo cuando ocurriera esto y nuestra función actualizar_max (2) no realizaba la operación que nosotros esperábamos, por eso decidimos colocar el caso de que si el padre queda como una nueva hoja, su máximo extremo derecha será el propio extremo derecha del intervalo. Luego la recursión se encargaría de actualizar a los demás nodos afectados.

- Ingresos por consola, en que formato es el ingreso

A la hora de leer el ejemplo por consola del trabajo practico para poder crear las funciones correspondientes, vimos que en el ejemplo toma intervalos de la forma “[a, b]” y luego toma un comando de la forma “[a,b]”. Nosotros decidimos tomar intervalos de la primera manera, además realizamos una consulta por tal motivo, y nos respondieron que fue un error de tipeo. Por lo cual la única forma de ingresar intervalos a través del interprete es si son de la forma [double, double], en caso contrario se imprimirá por pantalla el error correspondiente.

- Forma de Recorrido DFS

La forma en la que decidimos implementar la funcion itree_recorrer_dfs fue abstrayendo la forma de recorrer el árbol a una única, representada por funcionRecorrido, este recorrido será de la forma in order, el cual lo elegimos ya que los intervalos quedan ordenados según lo estipulado en la consigna.

- Función BFS, implementación sin utilizar pilas

Para nuestra función itree_recorrer_bfs decidimos implementarla sin la utilización de una cola, para eso nos disponemos de hacer uso de la altura de los nodos y de una función extra recorrer_niveles la cual aplica la función visitante a cada nodo de una determinada altura antes de seguir con la siguiente altura. La implementación de este algoritmo se debe a la comodidad de no estar importando la estructura de datos de colas.

- Formato para imprimir los intervalos

Nuestra implementación cuenta con una función visitante para imprimir el árbol por profundidad y por anchura, dicha función visitante imprime los números con dos decimales (estos se redondean). Esta decisión fue por motivo de que, si manteníamos el formato original de un float, se volvía muy tedioso la interpretación de los intervalos cuando estos se imprimían por consola, por lo cual utilizamos el formato ‘%.2f’ para imprimir los extremos de dichos intervalos.

- Finalización del Trabajo Practico

Luego de estos apartados antes mencionados no surgieron problemas a la hora de encontrar una solución a la consigna dada, pudimos adaptar lo dado en los slides de teoría más la bibliografía que buscamos para poder diseñar y aplicar los algoritmos que hacían falta para realizar el trabajo.

- Makefile

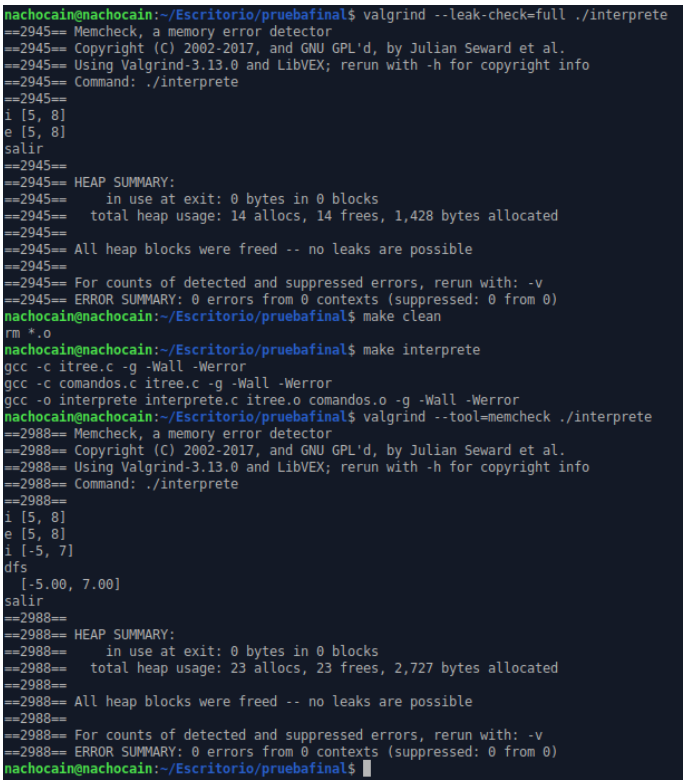
El archivo makefile adjuntado se encarga de la compilación del proyecto, dentro de el se encuentran las instrucciones para la compilación de cada uno de los archivos con sus respectivas banderas. Para utilizar este archivo y compilar el proyecto entero se debe ingresar el comando '$ make interprete' y para limpiar los archivos objeto (.o) se debe ingresar '$ make clean'.

- Debugging, Valgrind y problemas de memoria luego de terminar de hacer código

Una vez finalizado todo el código necesario para la finalización del trabajo, pasamos a realizar el debugging de nuestro proyecto, utilizando Valgrind nos dimos cuenta de ciertos errores de memoria que habíamos cometido:

  - Error en realloc para el array del comando ingresado, redimensionábamos según la longitud del string y no guardábamos espacio para el carácter '\0'.
  - No liberar memoria de un intervalo cuando lo eliminábamos, la función itree_eliminar hacia free del nodo, pero no del intervalo que contenia, con lo que agregamos ese free faltante para que se pueda liberar la memoria
  - No realizábamos el free del comando luego de salir del interprete, este lo ubicamos luego del switch, para que sea lo ultimo que se ejecute en el programa

A continuación, una imagen luego de lograr de debuggear el programa, primero con la herramienta leak-check y luego con memcheck, utilizando algunas funciones de la implementación.



El debugging se realizo en Ubuntu 18.04.01, y la compilación y ejecución se realizaron tanto en Ubuntu 18.04.01 como en Ubuntu 20.04 (este último en consola virtual desde Windows).

Bibliografía:

(1) https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm
(2) Introduction to Algorithms 3rd Edition, Thomas Cormen, Sección 14.3. https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf