UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Análisis de algoritmos y estructura de datos

Laboratorio 3 – Veterinaria

Alberto Rodríguez Z.

Profesor:	Pablo Schwarzenberg Riveros
Ayudantes:	Javiera Torres
	Javiera Sáez
	Diego Opazo

Santiago - Chile

1-2018

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

Uno de los contenidos dentro del curso Algoritmos y Estructuras de datos son los árboles binarios, una estructura de datos que derivan de los grafos, específicamente los grafos acíclicos. Su jerarquía nos ayuda a organizar colecciones de elementos de forma más óptima que las estructuras que ya conocemos como las listas enlazadas.

Este laboratorio cuenta con una veterinaria llamada el "Bulto Feliz", que guarda la información de las mascotas de sus pacientes en registros y estos son almacenados en un archivo de texto. Se tiene que implementar un programa que permita buscar y modificar información de una mascota en específico, como también agregar y eliminar registro de mascotas en el sistema. Además, se ingresa un archivo de texto llamado "Bultos.in" y el programa debe ser capaz de cargar su información. Finalmente se crea un archivo llamada "Bultos.out" donde contiene todos los registros con las mascotas que tienen que ser atendidas en el mes determinado.

Para lograrlo se utiliza la estructura antes mencionada, en particular un AVL, el cual es un tipo de árbol binario ordenado creado por Adelson-Velskii y Landis. Este tipo de árbol es de búsqueda binaria que cumplen con la propiedad de equilibrio.

Para lograrlo se exige construir el programa en base al lenguaje de programación C, específicamente ANSI C (estándar de lenguaje publicado por el Instituto Nacional Estadounidense de Estándares). Los algoritmos a usar tendrán que ser los mencionados anteriormente de manera que estos sean los óptimos para el problema a resolver.

Objetivos

1.1 Generales:

El propósito central de este proyecto es la implementación de un árbol AVL a fin de comprender las estructuras y algoritmos que lo abordan.

1.2 Específicos:

1. Aplicación de conceptos clave de árboles AVL aprendidos en clases, para poder trabajarlos con los archivos de la veterinaria.
2. Estudiar los algoritmos desde variados aspectos, desde la representación a través de los tipos de datos abstractos hasta los tiempos de ejecución de aquellos algoritmos.
3. Elaborar un programa que respete los estándares ANSI C con el fin de crear un programa eficiente, óptimo y estable.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

A continuación, se explicarán las herramientas, técnicas y algoritmos empleados para la elaboración de una solución además del material investigado el cual sirvió como base para resolver el problema.

2.1 Marco teórico:

Dentro de los conceptos clave de este informe se encuentran los tipos de datos abstractos, los árboles, árboles binarios y su anexo, específicamente AVL (nombre por sus creadores Adelson-Velskii y Landis). Para entenderlos se desarrollará brevemente la definición.

Un Tipo de dato abstracto (abreviado como TDA) es un conjunto de datos u elementos al cual se le asocian operaciones. El TDA provee de una interfaz con la cual es posible realizar dichas operaciones, de esta manera es posible acceder a los datos a través de las operaciones provistas por aquella interfaz.

ABO o árbol binario ordenado es un tipo especial de grafo, llamados grafos acíclicos. Son estructuras con una jerarquía definida que permiten la organización de objetos o elementos. Dentro de sus características son árboles cuyos nodos tienen a lo más dos hijos y estos se encuentran ordenados, es decir, para cualquier nodo dentro de un árbol, si este se encuentra en el sub-árbol izquierdo del padre, este será menor, así mismo, si el nodo se encuentra en el sub-árbol derecho del padre, su valor es mayor.

Un árbol AVL contiene todas las propiedades de un ABO, pero además cumplen con la propiedad de equilibrio. Esta propiedad señala que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es a lo más 1.

Es a través de estos árboles AVL que nos permitirá desarrollar un diccionario debido a que son óptimos para realizar búsquedas otorgando tiempos de ejecución mínimos O(logn) y O(N) en un peor caso).

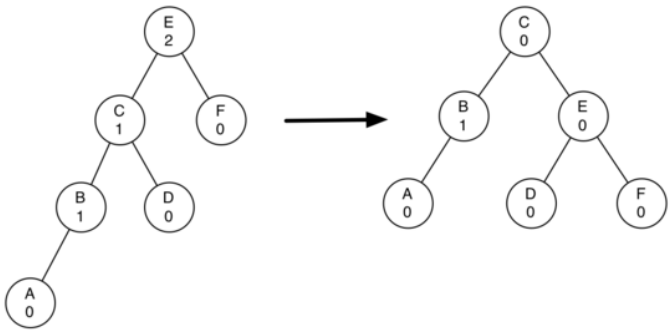

Fig 2.1 Transformación de un ABO a AVL.

2.2 Herramientas y Técnicas:

Como se implementó un TDA de AVL, fue necesario implementar todas sus operaciones correspondientes como saber si un nodo esta equilibrado o no, diferencias de nodos e inserción de nodos, entre estas operaciones una de las más importantes para un AVL son las rotaciones. Estas rotaciones, pueden ser simples o dobles y cumplen las funciones de mantener balanceado el árbol AVL, esta técnica consiste en el intercambio de nodos como lo ilustra la figura 2.2.1 para la rotación simple y la figura 2.2.2 para una doble.

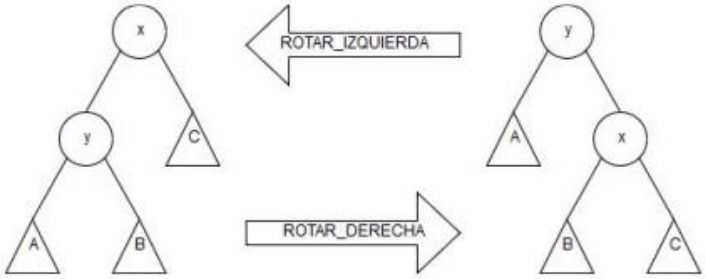

Fig 2.2.1 Rotación simple.

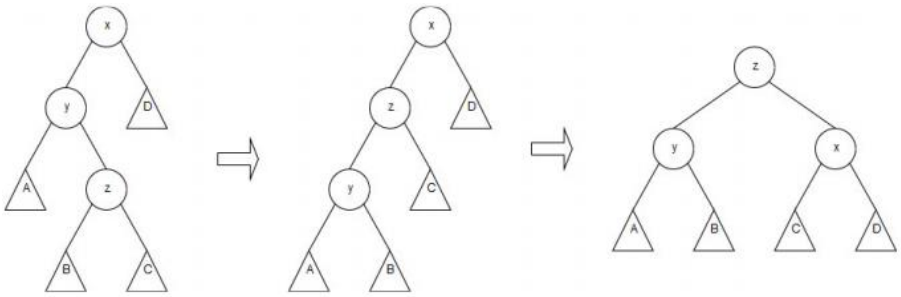

Fig 2.2.2 Rotación doble hacia la derecha.

Otra técnica importante fue como trabajar con el tipo de dato string dinámico. En el lenguaje C trabajar con string dinámicos tiene sus ciertas funciones que operan sobre los strings, algunas de las funciones utilizadas en el programa fueron:

-char *strcpy(char *cad1,char *cad2): copia el contenido de cad2 en cad, devolviendo cad1

-char *strtok(char *cad, “dato”): guarda en otro string los caracteres de un string hasta el dato indicado, osea si el string es strtok(“Hola Mundo”, “ “) guardará hasta el espacio

-char * strcmp(char *cad1,char *cad2): compara dos strings, de ser iguales devolverá un 0.

2.3 Algoritmos y estructuras de datos:

Estructura:

Gracias a los punteros, cada nodo podría considerarse un sub-árbol de un padre (teniendo claro que este solo posee los punteros, no los nodos en si de sus hijos), es por esto que se implementó una estructura única representando simultáneamente un árbol como un nodo.

Esta estructura con tiene strings con las características de cada ficha, tres enteros donde uno es el mes de la ficha, otro es el numero de la ficha y el ultimo el equilibrio de ese nodo (si esta equilibrado debiera ser -1,0 o 1), además de 3 punteros, 2 hijos (hijo izquierdo y hijo derecho) y uno que apunta al padre del nodo.

```
typedef struct nodo{
    char nombre[31],apellidop[31],apellidom[31],nomMascota[21],especie[21],edad[4],telefono[21],atenciones[4],vacunas[4],fecha[11];
    int numeroFicha;
    int mes;
    int equilibrio; //diferencia entre la altura del subarbol izquierdo y derecha de un nodo
    struct nodo *derecho;
    struct nodo *izquierdo;
    struct nodo *padre;
}Ficha;
```

Fig 2.3.1 Estructura nodo.

Algoritmos:

- *void leerTexto( )*

Función que lee el archivo ingresado por el usuario en este caso “Bultos.in”. La función para guardar la información de una línea crea un nodo, y va creando un nodo para cada línea.

Tiempo de ejecución: T(n) = cn+ c
Orden de complejidad: O(n).

- *void agregar(Ficha **raiz,Ficha *fich,int numero)*

Función que agrega un nodo al árbol, primero verifica cual es la posición correcta en donde va, para eso verifica el número de ficha de los nodos de árbol y el número de ficha del nodo a ingresar. Ya ingresado el nodo va a la función *equilibrar* para ver si el árbol esta balanceado.

Tiempo de ejecución T(n) = cn+c.
Orden de complejidad: O(n).

- *void eliminar(Ficha **raiz,int numero)*

Función que elimina algún nodo del árbol, este nodo es indicado por el usuario. Ya eliminado el nodo la función reordena el árbol, para eso cambia los hijos del nodo eliminado a otro nodo, para después ir a la función *equilibrar* para equilibrar el árbol.

Tiempo de ejecución: T(n) = n^2 + cn + c.
Orden de complejidad: O(n^2).

- *void modificar(Ficha **arbol)*

Función que modifica algún dato del nodo seleccionado por el usuario, retorna el arbol AVL con el dato del nodo modificado.

Tiempo de ejecución: T(n) = 2(n+c) + c
Orden de complejidad: O(n).

- *void buscar(Ficha *inicio)*

Función que busca el string ingresado por el usuario en él árbol. Solo puede buscar en los nodos: el nombre del cliente, nombre de la mascota y la fecha del próximo control. Imprime por pantalla el registro si se encuentra dentro de él lo solicitado.

Tiempo de ejecución: T(n) = cn + c
Orden de complejidad: O(n).

- *void equilibrar(Ficha **raíz,Ficha *nodo,int rama,int operación)*

Función que verifica si el árbol esta equilibrado o no, si este no está equilibrado lo envía a una de las 4 funciones de rotación según su caso de desequilibrio.

Tiempo de ejecución: T(n) = cn +c

Orden de complejidad: O(n).

- *void rotacionSimpleIzquierda(Ficha **raíz,Ficha *nodo)*

Función que realiza rotación hacia la izquierda cuando la raíz tiene equilibrio -2, donde el hijo derecho del parámetro *nodo* pasa a ser la raíz (no necesariamente del árbol, puede ser de un sub-árbol), y equilibra los nodos.

Tiempo de ejecución: T(n) = c

Orden de complejidad: O(1).

- *void rotacionSimpleDerecha(Ficha **raíz,Ficha *nodo)*

Función que realiza rotación hacia la derecha cuando la raíz tiene equilibrio a 2, realiza lo mismo que la función anterior.

Tiempo de ejecución: T(n) = c

Orden de complejidad: O(1).

- *void rotacionDobleIzquierda(Ficha **raíz,Ficha *nodo)*

Función que realiza doble rotación a la izquierda, esta función solo se efectuará cuando el equilibrio de la raíz sea -2 y el equilibrio del hijo derecho de la raíz sea 1. El hijo derecho del hijo izquierdo de la raíz pasará a ser la nueva raíz.

Tiempo de ejecución: T(n) = c

Orden de complejidad: O(1).

- *void rotacionDobleDerecha(Ficha **raiz,Ficha *nodo)*

Función que realiza doble rotación a la derecha, esta función solo se efectuará cuando el equilibrio de la raíz sea 2 y el equilibrio del hijo izquierdo de la raíz sea -1. El hijo izquierdo del hijo derecho de la raíz pasará a ser la nueva raíz.

Tiempo de ejecución: T(n) = c

Orden de complejidad: O(1).

- *void ImprimirArchivo(Ficha *inicio)*

Función que crea el archivo “Bultos.out” donde ingresa los datos de cada nodo según el mes que le corresponda.

Tiempo de ejecución: T(n) = c
Orden de complejidad: O(1).

2.4 ANÁLISIS DE LOS RESULTADOS

Al trabajar en el lenguaje de programación C, una de sus características es el manejo de memoria, la cual para propósitos de este laboratorio nos permite realizar un programa que aparte de resolver el problema, sea óptimo en cuanto a tiempos de ejecución, para eso se analizó a través de fórmulas que permiten calcular cuánto se demora cada ciclo. Estas fórmulas nos dan un parámetro para poder determinar la eficiencia de nuestro programa.

Una de las funciones más importantes del programa es la de leer texto, que donde mas se demora el programa en funcionar, al tener que leer cada línea y guardar cada dato en la estructura. Al ingresar un archivo de gran tamaño para este laboratorio el cual contiene 2 millones de registros se calculó que el tiempo de demora en cada programa es de 50 minutos.

Por lo cual al compararlo con el laboratorio 1, el cual tiene el mismo objetivo, pero se implementa con estructuras de datos diferentes (listas y arreglos). Se puede apreciar que implementar este objetivo con árbol AVL es menos eficiente. Esto se debe a que el algoritmo de ordenamiento de los árboles AVL requieren más tiempo de ejecución, caso contrario a las listas enlazas. Pero en caso de búsqueda, los algoritmos de búsqueda los arboles son más eficiente que los de listas, porque su complejidad es de log(n), mientras que los otros tiene complejidad n exponencial.

Después de leer el texto las otras funciones se demoran un tiempo razonable en realizar las funciones no se demorar más de 2 minutos.

Retomando los objetivos, podemos ver un logro satisfactorio, implementando correctamente un programa que aborde lo que se pide el enunciado, ya que con las pruebas realizadas el programa es capaz de agregar, eliminar, modificar y buscar registros en los dos algoritmos.

Dentro de los objetivos específicos podemos ver como logramos estudiar los conceptos clave y aplicar aquellos de manera que se construyó un programa que solucione el problema correctamente. Este programa también cumple nuestro objetivo de respetar los estándares ANSI C compilando en entornos de Windows como en Linux, siendo un programa optimo y estable para recibir al usuario y permitirle el uso de las herramientas creadas.

CAPÍTULO 3. CONCLUSIÓN

Como se mencionó en los análisis, el proyecto tuvo un grado de logro bastante alto ya que se logró estudiar en profundidad y con tiempo el TDA de los árboles AVL, para así poder plantear los algoritmos necesarios para su operación y de manera exitosa crear un programa que resuelve todos los problemas planteados en el enunciado tal como se pidieron.

Gracias al presente laboratorio fue posible aprender la implementación de una estructura en árboles AVL, como trabajar en sus TDA, balancearlos y sus operaciones respectivas. Con respecto al tiempo de ejecución podemos analizar que es más eficiente que implementando con otros tipos de estructuras, el programa es capaz realizar todas las instrucciones que se le piden si riesgo de errores o fallos en el mismo.

A futuro se espera poder realizar un manual de usuario más completo ya que si bien se aborda todo lo especificado, siempre podría detallarse mejor las instrucciones de compilado y uso del programa junto a sus limitaciones.

REFERENCIAS

Héctor Canales Molina (2002). *Árbol AVL en C*. (España) Recuperado de: *http://www.academia.edu/9750509/Arbol_AVL_en_C*

Jacqueline Köhler Casasempere (2017). *Apuntes de la asignatura Análisis de Algoritmos y Estructuras de Datos*. Chile, Departamento de ingeniería informática (DINF), Usach.