UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Estructura de Datos y Análisis de Algoritmos

Laboratorio 4:
Traductor

Catalina Araya Ilabaca

Profesor: Alejandro Cisterna
Ayudantes: Javiera Torres
Sebastián Garay
Nicolas Romero

Santiago - Chile
2-2018

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

CAPÍTULO 1. INTRODUCCIÓN

La traducción ha sido fundamental para el desarrollo de las sociedades, pues es lo que permite el flujo de información entre personas de países de distintas lenguas maternas , en este caso en particular, el poder de la traducción ayudará a los informáticos dueños del casino Little C's a librarse de la nube de mala suerte que siempre los acompaña no importa el tiempo que pase, y lo hará a base de un diccionario de palabras inglés-español y español-inglés , el cual permitirá a esas pobres almas en pena, comunicarse con sus clientes en las Vegas , que es donde mudarán el casino ,para poder desentenderse de los grandes impuestos chilenos y aceptar y disfrutar de los bajos impuestos y placeres de las Vegas.

Por ellos , los dueños del casino Little C's han recurrido a nosotros los alumnos de analisis de algoritmo y estructura de datos para que llevemos a cabo el diccionario inglés-español y español-inlges a través de un algoritmo que cumpla con tal rol implementado en el lenguaje de programación C.

El diccionario debe destacar su contenido , es decir, la palabra en el idioma raíz y la palabra traducida al idioma contrario, existen dos tipos de estos diccionarios. Lo que se busca finalmente con esta simulación es un programa al cual se le entrega una palabra y devuelve su traducción.

A lo largo del siguiente informe se detallarán los distintos métodos y algoritmos a usar para llegar a una solución exitosa del problema mencionado anteriormente. Se partirá mencionando que a través de este documento no debe quedar ninguna de duda de cómo se realizó el programa y cada una de sus funcionalidades principales.

Para iniciar se mencionan los objetivos generales y específicos que se proponen en este informe, posterior a eso se detalla cómo se llegó a la solución donde se mencionan conocimientos teóricos y las herramientas conocidas para llevar a cabo la implementación tales conocimientos, luego se mencionan los algoritmos y estructuras de datos usados donde se detallarán las funciones más importantes, su(s) entrada(s), su(s) salida(s) , su tiempo de ejecución y orde, además del rol que cumplen dentro del programa. Se analizaron los tiempos de ejecución y orden para poder finalmente concluir y proponer mejoras, además de mencionar los errores o imprevistos que se encontraron a lo largo del desarrollo e implementación.

1.1 Objetivos

1.1.1 Generales :

i. El objetivo principal de este laboratorio es comprender y aplicar el concepto de árboles binarios ordenados , abreviado como ABO , para poder implementarlo como solución para el laboratorio.

1.1.2 Específicos :

i. Aplicar conceptos aprendidos en clase sobre el TDA de tipo ABO, para poder simular el diccionario mencionado anteriormente.

ii. Aplicar conocimientos adquiridos en clase para el análisis de tiempos de ejecución, logrando así una conclusión sobre si el programa puede ser mejorado.

iii. Crear una programa que solucione la problemática planteada anteriormente. En el lenguaje de programación C y utilizando la biblioteca estándar ANSI C.

iv. Detallar las principales funcionalidades del programa en cuestión.

CAPÍTULO 2. SOLUCIÓN

2.1 DESCRIPCIÓN DE LA SOLUCIÓN

Para solucionar el problema planteado se debe crear una simulación de diccionarios , inglés-español y español-inglés ,en el cual el usuario deberá decidir en cuál de los dos diccionarios antes mencionadas buscar la palabra , luego ingresar la palabra cuya traducción se desea saber y mostrar por pantalla el resultado final que es la traducción de esa palabra , ya sea en ingles o español dependiendo del diccionario ocupado. Para esto se debe crear un árbol que contenga información de los nodos por ende de las palabras tanto en español como en inglés .

La solución será un algoritmo que resuelva el problema antes planteado , para ello se utiliza el lenguaje de programación C, el cual será la herramienta para implementar el programa. A continuación se nombran las principales funciones que componen el programa: leerArchivo, crearNodo, agregarEspañol, agregarIngles, construiArbol , buscar ,palabraSolicitada, imprimirHoja, borrarArreglo y borrarArbol. De estas funciones depende el correcto funcionamiento, además de otras funciones de apoyo y complementarias como: existente, menuDiccionario y comparar.

Las funciones anteriores están creadas sobre la base de funciones propias del lenguaje programacion C, tales como tales como malloc y realloc las que sirven para reservar memoria y ampliar o reducir la memoria antes reservada en el caso de arreglos dinámicos ,cómo se usará en este caso, para crear el arreglo que contiene las palabras , y para reservar memoria para las hojas (o nodos) que contienen las palabras , en el arreglo creado para ello. Además se utilizó la función free para liberar la memoria que había sido reservada con malloc.

Por otro lado se utilizan también las funciones fopen, fclose y fgetc, las que sirven para abrir un archivo , cerrar un archivo y obtener lo que está dentro del archivo carácter por carácter respectivamente.

Adicionalmente se hace uso las siguientes funciones: strncmp, toupper y scanf, la primera de ellas sirve para comparar dos string ,la segunda tiene como función convertir a mayúscula el carácter que se le entregue y la tercera permite que el usuario ingrese lo que el programa requiera por pantalla, en este caso , el diccionario que utilizara y la palabra que desea buscar.

Finalmente cabe destacar que existen dos estructuras creadas para lograr el resultado final: palabras y arbol las cuales se detallaran más adelante en el informe.

2.2 MARCO TEÓRICO

Para llevar a cabo la solución se utiliza el tipo de dato árboles , que son un tipo especial de grafos, pues son acíclicos .Son estructuras jerárquicas para organizar colecciones de elementos. Se tiene un nodo llamado raíz que es la base (o nodo superior depende de la perspectiva con que se mire) para

formar un árbol , este a su vez tiene más nodos conectados (llamados nodos hijo) , que se alejan cada vez más de las raíz.

El árbol tiene distintos niveles , dependiendo de la distancia con el nodo raíz, mientras más alejado más grande el nivel en el que se encuentra el nodo hijo. Todos los nodos hijos tienen un padre exceptuando al nodo raíz. Se considera nodo hermano a un nodo con el que se comparta el mismo padre. A continuación se muestra un ejemplo árbol, y algunos de los conceptos mencionados.

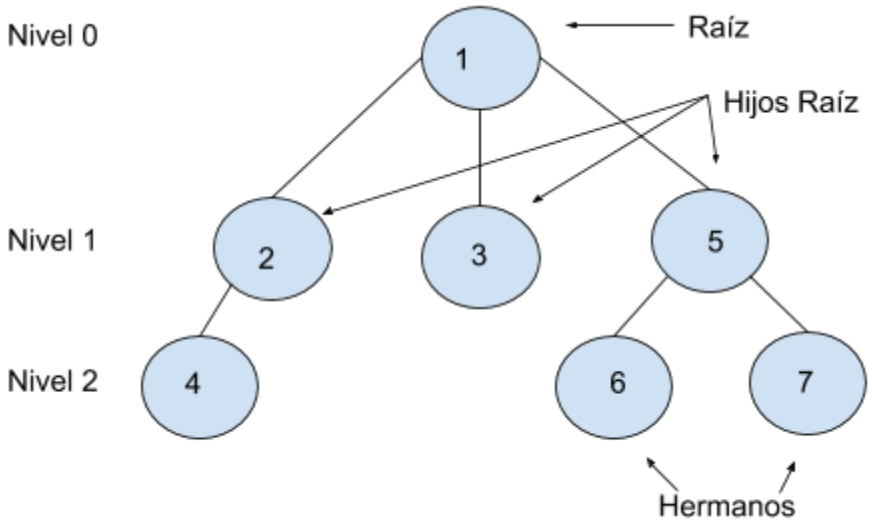

Figura 2-1: Ejemplo árbol y algunos de sus conceptos

Los árboles se pueden recorrer de tres forma distinta preorden,inorden y postorden que se explican a continuación :

En el recorrido preorden la raíz del árbol siempre irá al inicio,por ende se revisa la raíz, luego el hijo izquierdo luego el derecho y los nodos que sigan a este.

En el recorrido inorden la raíz del árbol siempre irá siempre al medio, primero se revisa el hijo izquierdo , luego la raíz y finalmente los nodos que estén a la derecha .

En el recorrido postorden la raíz del árbol siempre irá al final, primero se revisa el nodo izquierdo , luego el derecho y los nodos hermanos de este y finalmente la raíz.

A continuación se mostrará un ejemplo de estos recorridos.

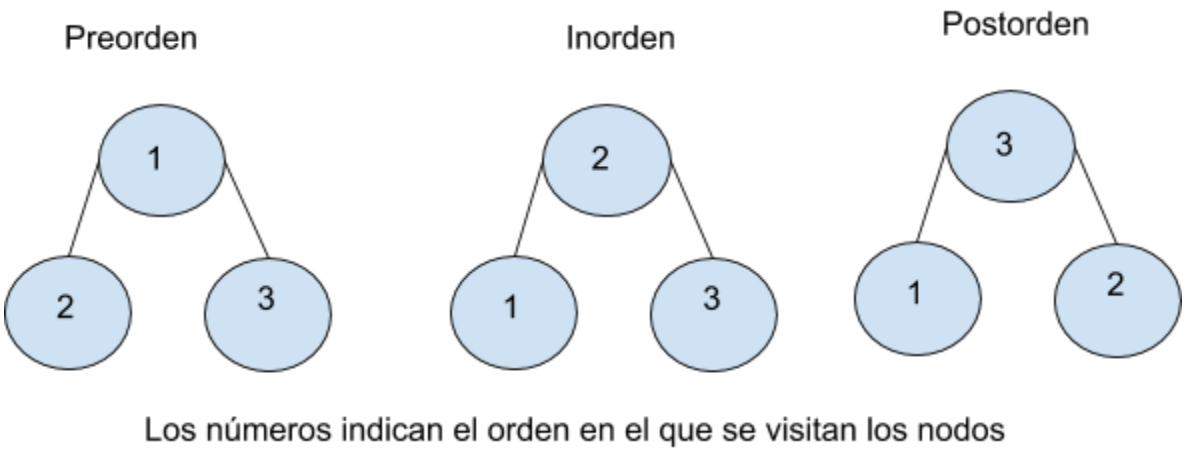

Figura 2-2: Ejemplo de recorridos de árboles

Existen distintos tipos de árboles, uno de ellos son los árboles binarios, en su forma abreviada AB, estos consisten en árboles que cómo máximo cada nodo puede tener dos nodos hijos y además se debe hacer la distinción si el nodo será hijo izquierdo o derecho.

Otro tipo de árboles son los llamados árboles binario ordenados, en su forma abreviada ABO, o árboles de búsqueda binaria el cual cumple con la condición de árbol binario al tener como máximo 2 hijos , pero se le agrega la siguiente condición: si el nodo que se quiere insertar es mayor o igual se debe insertar en el nodo recho y si es menor en el nodo izquierdo. Debido a esta condición si el árbol se recorre inorden se deberán mostrar de forma ordenada. A continuación se muestra un ejemplo con la diferencia entre un AB y un ABO.

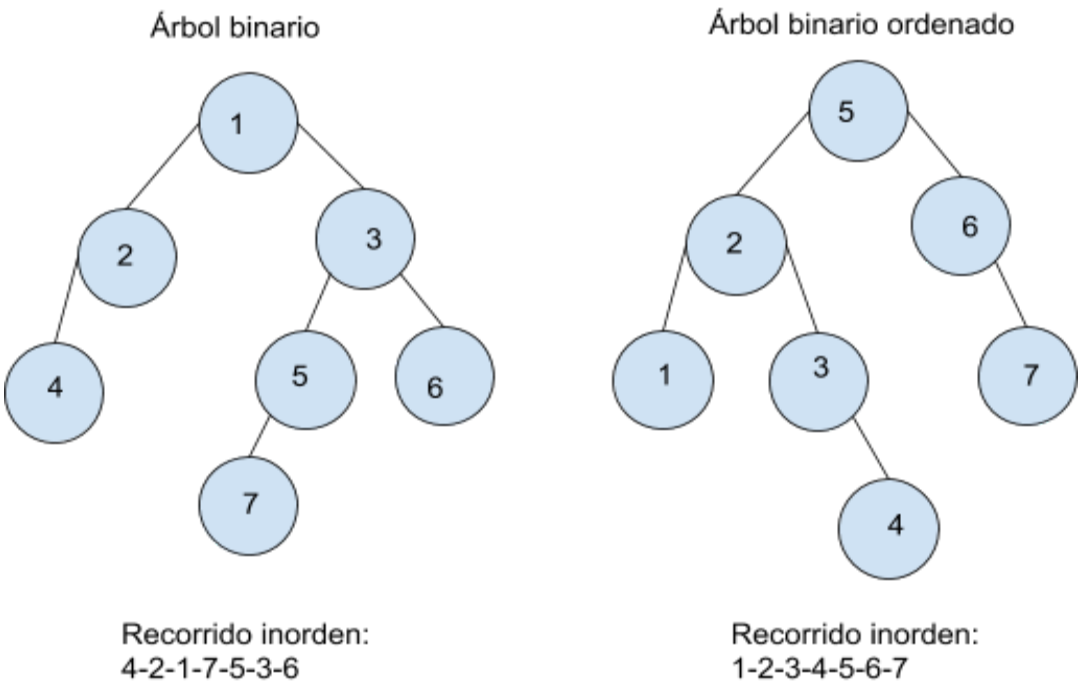

Figura 2-3: Diferencia entre un árbol AB y un árbol ABO

2.3 Herramientas y técnicas

Para llevar a cabo la implementación se hizo uso del TDA del árbol binario ordenado aprendido en clase, el cual se adaptó para poder resolver el problema bajo las condiciones pedidas en este caso. Pues se utiliza la búsqueda binaria para buscar la palabra ingresada por el usuario.

Además de la utilización del método de programación de división en subproblemas. Por lo que se decide separar en tres partes el programa: obtener información del archivo, armar el árbol binario ordenado y buscar la palabra que quiere el usuario.A Cada parte está enunciada en el orden en el que se utilizará a lo largo del programa.

Debido a que en el lenguaje de programación C , la letra ñ da problemas ,por ello la palabra español se cambio por espanol a lo largo de todo el programe en las funciones que lo requerían.

2.4 Algoritmos y estructuras de datos

Primero se analizaron las estructuras utilizadas a los largo del programa cada una de ellas por separado. Posterior a eso se continua con la explicación de las distintas funciones utilizadas a lo largo del programa las cuales se explicaran por los tramos en lo que se separaron al momento de aplicar la división en subproblemas.

2.4.1 Estructuras

estructuraPalabras: Estructura de datos implementada para guardar dos palabras (o strings) una en español y otra en inglés, cada palabra es la traducción de la otra en su idioma correspondiente. Se usó “palabras” como identificador para esta estructura.

palabras: estructura

palabraEspanol : arreglo de caracteres (string)

palabraIngles : arreglo de caracteres (string)

estructuraNodoArbol: Estructura de datos implementada para guardar los datos para poder crear un nodo que pertenece a un árbol. Dentro de sus características destacan que posee 6 punteros los cuales apuntarán a los distintos nodos hijos izquierda o derecha , español o inglés , y también a los nodos padres en ambos idiomas. Esto permite saber donde agregar el siguiente nodo pues se sabe que existe en el nodo hijo izquierdo y en el nodo hijo derecho por lo cual el insertar un nuevo nodo al árbol será simple. Su función principal es guardar las palabras en español e inglés y los punteros hacia los otros nodos para conocer a los padre e hijos que le corresponden dependiendo del idioma en el que se revise (tipo de diccionario). Se utiliza la palabra “Arbol” como identificador de esta estructura.

arbol: estructura

palabraEspanol: arreglo de caracteres (string)
palabraIngles: arreglo de caracteres (string)
padreEspanol: puntero a estructura arbol
padreIngles: puntero a estructura arbol
hijoIzquerdoEspanol: puntero a estructura arbol
hijoDerechoEspanol: puntero a estructura arbol
hijoIzquerdoIngles: puntero a estructura arbol
hijoDerechoIngles: puntero a estructura arbol

2.4.2 Funciones

Tal y como se mencionó anteriormente se mostrarán las funciones del programa por partes.

1. Obtener información del archivo

Nombre Función (algoritmo)	Entrada (input) - Salida (output)	Funcionamiento (descripción)	T(n) - O(n)
leerArchivo	E: No posee entrada ---------- S: Arreglo dinámico de palabras, y puntero a un entero	Función que rescata las palabras del archivo "traductor.in" y las guarda en un arreglo dinámico del largo de la cantidad de pares del tipo TDA palabras. En caso de que el archivo esté nulo entrega un '1' como resultado del arreglo de palabras. También a través de paso por referencia entrega la cantidad de pares de palabras presentes en el archivo "traductor.in". Cabo mencionar que sólo acepta archivos donde la primera letra de ambas palabras este en mayuscula.	T(n)= n+c ---------- O(n)=n
Existente	E: No posee entrada ---------- S: Entero	Funcion que comprueba a existencia del archivo traductor.in en la carpeta donde esta el programa.	T(n)=c ---------- O(n)=1

Tabla 2-1 : Tabla de funciones de la parte "obtener información del archivo" del programa

2. Armar el árbol binario ordenado

Nombre Función (algoritmo)	Entrada (input) - Salida (output)	Funcionamiento (descripción)	T(n) - O(n)
crearNodo	E: Una estructura palabras. ---------- S: Puntero a estructura arbol.	Función que crea una hoja o nodo del árbol ,y reserva memoria, también inicializan los seis punteros de la estructura arbol.	T(n)=c ---------- O(n)=1
comparar	E: Dos cadenas de caracteres (string).	Función que retorna un entero con el valor (negativo, 0 o positivo) que usa	T(n)=c ----------

	---------- S: Un entero.	la función strncmp propia de lenguaje c para comprar los dos string y decir a qué lado según el alfabeto debería ir el segundo string en comparación al primero ingresado.	O(n)=1
agregarEspanol	E: dos punteros a una estructura arbol. ---------- s: Puntero a estructura arbol.	Función recursiva que agrega las hojas al arbol principal , por la rama del idioma español, es decir, altera sólo los punteros del idioma español. Siendo el primer puntero ingresado (hoja) el que se desea agregar al segundo puntero(arbol principal). Si el puntero que ingresa es una estructura nula, se devuelve el mismo puntero que ingresa, sino es asi se hace uso de la función comparacion para saber si el nodo debe ingresarse a la derecha o a la izquierda y esto se hace hasta encontrar un nodo que esté vacío y esa es la ubicación donde se insertará el primer puntero(hoja).	$T(n) = T(n-1) + O(n)$ ---------- $O(n)= O(n^2)$
agregarIngles	E: dos punteros a una estructura arbol. ---------- s: puntero a estructura arbol.	Exactamente el mismo funcionamiento que que agregarEspanol sólo que esta vez se recorre por el idioma inglśs. Función que agrega las hojas al arbol principal , por la rama del idioma inglés , es decir, solo altera los punteros del idioma inglés.	$T(n) = T(n-1) + O(n)$ ---------- $O(n)= O(n^2)$
construirArbol	E: Arreglo de estructura palabras, un entero. ---------- S: Puntero a estructura arbol.	Función que construye el arbol completo es decir entrega el arbol con todos los elementos , tanto en inglés como en español, y para ello usa las funciones antes mencionadas. Para ello se recorre el arreglo de estructura de palabras y por cada par de palabras se va insertando en el arbol tanto en español como en ingles.	$T(n)=n*2(T(n-1) + O(n))$ ---------- $O(n)=n^3$

Tabla 2-2 : Tabla de funciones de la parte "Armar el árbol binario ordenado" del programa

3. Buscar la palabra que quiere el usuario

Nombre Función (algoritmo)	Entrada (input) - Salida (output)	Funcionamiento (descripción)	T(n) - O(n)
buscar	E: Cadena de caracteres, puntero a estructura arbol, entero, estructura palabras	Función recursiva que busca una palabra dada por el usuario , en inglés (tipo 0) o en español (tipo 1) y entrega la información de ese nodo , en caso de no encontrar la palabras crea un nodo vacío que e indicará al programa que la palabra no existe en el diccionario indicado por el usuario.	$T(n) = T(n-1) + O(n)$ ---------- $O(n)= O(n^2)$
palabraSolicitada	E: Puntero a cadena de caracteres. ---------- S: Puntero a cadena de caracteres por referencia.	Función que guarda la palabra que será buscada por el usuario . Esta función hace uso de la función scanf para guardar el resultado de lo ingresado por el usuario , tiene una limitación de 15 caracteres . El resultado de esta función es el arreglo de caracteres que ingresa en la función buscar.	T(n)=n+c ---------- O(n)=n+c
imprimirHoja	E: Puntero a estructura arbol y un entero. ---------- S: No posee salidas.	Función que muestra el contenido de la hoja ,es decir,todos los elementos presentes en una estructura del tipo arbol (dos arreglos de caracteres y los arreglos de caracteres pertenecientes a las estructuras donde apuntan) y los imprime para que el usuario pueda verlos. En caso de no existir tal estructura se le mostrar la palabra NULL al usuario indicando que tal estructura no posee el puntero que se indica.	T(n)=c ---------- O(n)=1
menuDiccionario	E: No posee entradas. ---------- S: Un carácter.	Función que muestra el menú de diccionario y da lugar a dos opciones, elegir español-inglés o inglés-español. Del resultado de esta	T(n)=c ---------- O(n)=1

		función depende el tipo de enter que ingresa a la función buscar.	

Tabla 2-3 : Tabla de funciones de la parte "Buscar la palabra que quiere el usuario" del programa

Por otro lado y externas a la solución del problema se encuentras las funciones de limpieza de memoria , pues en el lenguaje de programación los programadores son los encargados de limpiar la memoria que se pidió a través de la función malloc , para ellos se crearon dos funciones: borrarArreglo y borrarArbol las que se explican a continuación.

Nombre Función (algoritmo)	Entrada (input) - Salida (output)	Funcionamiento (descripción)	T(n) - O(n)
borrarArreglo	E: Arreglo del tipo palabras. ---------- S: Retorna el arreglo vacío.	Función encargada de liberar memoria del arreglo de TDA palabras usado en el inicio del programa. Para ello se hace uso de la función free.	T(n)=n ---------- O(n)=n
borrarArbol	E: Puntero a estructura de tipo arbol. ---------- S: Puntero a estructura de tipo árbol vacía.	Función que libera la memoria de la estructura arbol una vez finalizado utilizando la recursión. Sólo se hace uso una vez , pues al memoria sólo fue pedida una vez.	$T(n) = T(n-1) + O(n)$ ---------- $O(n)= O(n^2)$

Tabla 2-4 : Tabla de las funciones encargadas de limpiar la memoria

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

La mayoría de las funciones dependen de la cantidad de pares existentes en el archivo "traductor.in". Se puede observar que la mayoría de los algoritmo posee un T(n) constante, sin embargo los siguientes tres destacan por su alto tiempo de ejecución y por su complejidad.

Nombre función	T(n)	O(n)
agregarEspanol	$T(n-1) + O(n)$	$O(n)=n^2$
agregarIngles	$T(n-1) + O(n)$	$O(n)=n^2$
construirArbol	$T(n)=n*2(T(n-1) + O(n))$	$O(n)=n^3$

Tabla 3-1 : Tabla de funciones, tiempos de ejecución y orden de aquellas con mayor orden

En la tabla podemos notar que la función con mayor tiempo de ejecución es construirArbol , por ende esta será la función que haga que el programa tenga al menos un tiempo de ejecución igual al de esa función.

Al hacer uso de la técnica de ABO, se corre el riesgo de que la primera palabra en el diccionario fuese por ejemplo una que empieza por la letra "A" , por ello el árbol quedaría cargado hacia un lado pues todas las palabras siguientes se hubiraciana la derecha de esta,y si fuese el caso que el usuario busque una palabra que empiece por "Z", la búsqueda tomaría un tiempo considerable.

Por otro lado si se hubiera utilizado una técnica distinta cómo el uso de árboles AVL , el tiempo de ejecución hubiese sido más pequeño debido a que estos tiene una complejidad completamente menor , por lo tanto , al ser AVL están por definición equilibrados toma aproximadamente el mismo tiempo buscar cualquier nodo u hoja.

CAPÍTULO 4. CONCLUSIÓN

A modo de conclusión se tiene que el programa es capaz resolver el problema propuesto, y se complieron con los objetivos planteados en la introducción de este documento, puesto que se logró aplicar la materia de ABO vista en clases, también se aprendió del diverso uso que se le puede dar a los ABO al ser utilizado en algo fuera de lo abstracto como es el uso dado normalmente en cátedra. Por otro lado se detallaron las funciones que componen el programa , sus tiempos de ejecución y su complejidad. Se utiliza la biblioteca estándar ANSI C.

CAPÍTULO 5. REFERENCIAS

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