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

Estructura de Datos y Análisis de Algoritmos

Laboratorio 4 : Traductor

Bastián Gonzalo Vera Palacios

Profesor: Jacqueline Köhler

Ayudantes: Nicole Henríquez

Sebastián Vallejos

Javiera Torres

Santiago - Chile
2-2017

Tabla de Contenido

Índice de Figuras

Índice de Tablas

CAPÍTULO 1. INTRODUCCIÓN

Durante épocas pasadas, la comunicación entre distintas zonas del mundo se había tornado en un problema, ya que en muy pocas oportunidades se podía entender lo que alguien que hablara en un idioma distinto al propio pudiera expresar. A medida que avanzó el tiempo y a la vez con los avances tecnológicos, el hecho de poder buscar y saber como expresarte con otra persona que no hable en la misma lengua que tú se volvió un tema trivial y bastante simple gracias a la creación de un sistema encargado de comprender el significado de un texto para producir un significado equivalente en otro idioma, la traducción. Actualmente hay una gran variedad de traductores dentro de la red las cuales logran resultados bastante certeros respecto a nuestras consultas e inquietudes.

En este laboratorio se le solicitó a los estudiantes de ingeniería informática de la Universidad de Santiago de Chile realizar mediante una serie de algoritmos en lenguaje C y la implementación de un árbol AVL, un programa que reciba un archivo y que éste sea capaz de entregar la traducción de una palabra, su definición y un archivo generado con los valores de sus punteros para la verificación de los hijos de cada nodo con el fin de poder medir las habilidades adquiridas durante el desarrollo de éste curso.

Los métodos esperados para el desarrollo de los algoritmos del programa en lenguaje C, serán principalmente TDA árbol AVL, el cual debe ser implementado con punteros y memoria dinámica para el trabajo con los nodos y el uso de manejo de archivos para poder trabajar con la información entregada.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1 MARCO TEÓRICO

2.2.1 Librerías

Entre las herramientas utilizadas para el desarrollo de este laboratorio, se encuentran las librerías de:

- stdio.h: Para la entrada, salida de datos y manejo de archivos.
- stdlib.h: Utilizada para el manejo de la memoria dinámica.
- ctype.h: Utilizada para trabajar los strings en minúscula para las revisiones.
- string.h: Utilizada para el manejo de concatenaciones de caracteres.

2.2.2 Punteros

Los punteros son elementos bastante utilizados durante el desarrollo del código debido a que permiten trabajar directamente a las direcciones de memoria de las estructuras de datos que tenemos. Se utilizan tanto como para lectura del archivo como para poder acceder a los hijos, posee cada nodo tanto en la palabra como en su traducción.

2.2.3 Lectura de Archivos

Para poder realizar la lectura de archivos anexos a los ingresados manualmente por terminal en nuestro código se utilizan funciones específicas llamadas por la librería stdio.h las cuales nos facilitan el acceso a éstos.

2.2.4 Memoria Dinámica

Para un óptimo avance en este código se utilizó memoria dinámica durante la inicialización de los nodos con el fin de que se pueda trabajar con los valores leídos en el archivo de manera más eficiente. Para esto se utilizó la función *calloc()* para poder asignar a la vez con el valor 0 cada vez que se pedía memoria y la función *free()* para ir liberando la memoria una vez que esta ya ha sido utilizada.

2.2.5 Estructuras

Las estructuras son un tipo de dato utilizado para poder agrupar distintos tipos de información y manejar datos que sería muy complicado de realizar de manera más primitiva. En el desarrollo de este código se utilizan para la implementación de los nodos del árbol ya que así se puede acceder de manera inmediata a la información contenida en estos.

2.2 HERRAMIENTAS Y TÉCNICAS

El traductor consiste en un algoritmo que sea capaz de poder entregar tanto la traducción de la palabra en otro idioma como el significado de esta. Con esto se solicita organizar la información con el fin de poder realizar las búsqueda de manera más eficiente y práctica posible. Una de las mejores opciones para lograr esto es utilizar como estructura un árbol AVL la cual favorece al momento de realizar las búsquedas de las palabras debido a que dentro de las propiedades de un árbol AVL se encuentre el reordenamiento de las hojas lo cual optimiza el proceso de búsqueda reduciendo los tiempos en cada una de las llamadas al algoritmo. Para esto se implementó el TDA del árbol AVL con el fin que al momento de realizar la lectura del archivo con las palabras, éstas se almacenen en primera instancia de manera ordenada y listas para trabajar.

Dentro de las operaciones que se utilizaran durante el TDA se solicita que cada nodo tenga punteros distintos para la palabra en español como en la palabra traducida, por ende se tendrá que realizar el TDA árbol para cada operación dentro de las palabras ingresadas de manera independiente el uno del otro, ya que el fin es poder generar dentro de un mismo árbol dos ordenamientos de AVL, uno correspondiente a la palabra en español y otro a la palabra traducida, ambas pertenecientes al mismo nodo.

Una vez que la lectura de los datos se haya realizado con éxito se procederá a la búsqueda de las palabras dependiendo de la solicitud del usuario. Para esto se le entregará al usuario un menú el cual posee las opciones que pueda realizar, tales como la traducción de una palabra, el significado de esta o simplemente mostrar en un archivo de salida todos los nodos del árbol junto con sus respectivos hijos. Posterior a esto el programa procederá a mostrar los resultados y a realizar una nueva consulta en el menú respecto a la acción que se desee realizar. En caso de no querer realizar ninguna acción más, se dará la opción de salir del programa.

2.3 Algoritmos y Estructuras de Datos

2.3.1 Estructuras Utilizadas

2.3.1.1 Nodo

Estructura utilizada como nodo del árbol el cual contiene almacenada la palabra en español, su traducción, su significado y los punteros respectivos hacia los padres de cada nodo y los hijos izquierdo y derecho para la palabra en español y la traducción.

2.3.1.2 AVL

Estructura que posee los punteros hacia el nodo raíz en español y en su traducción.

```
typedef struct Nodo {
    char * palabraEsp;
    char * palabraOtro;
    char * defEsp;
    char * defOtro;
    struct Nodo * padreEsp;
    struct Nodo * padreOtro;
    struct Nodo * espL;
    struct Nodo * espR;
    struct Nodo * otroL;
    struct Nodo * otroR;
} Nodo ;

typedef struct AVL {
    struct Nodo *rootEsp;
    struct Nodo *rootOtros;
} AVL;
```

Figura 1 : Estructuras del programa

2.3.2 Algoritmos Más Importantes

2.3.2.1 buscarPalabraEsp( )

Entradas: - nodo *Nodo - char * palabrasIn - char * palabrasOut

Salidas: - Sin salidas.

```
void buscarPalabraEsp(Nodo *nodo,char * palabraIn,char * palabraOut) {
    if(nodo != NULL){
        if (strcmp(convertirTolower(nodo -> palabraEsp),convertirTolower(palabraIn))==0) {
            strcpy(palabraOut,nodo -> palabraOtro);
        }
        buscarPalabraEsp(nodo -> espR,palabraIn,palabraOut);
        buscarPalabraEsp(nodo -> espL,palabraIn,palabraOut);
    }
}
```

Figura 2 : Función búsquedaPalabraEsp

Función recursiva encargada de realizar la búsqueda de la palabra solicitada y entregar su respectiva traducción. Existe a la vez su función opuesta con la misma estructura y funcionamiento pero encargada de obtener la palabra a partir de una traducción en otro idioma.

$T(n) = \mathbf{2}n$

2.3.2.2 buscarDefinicion( )

Entradas: - nodo *Nodo - char * palabrasIn - char * palabrasOut

Salidas: - Sin salidas.

```
void buscarDefinicion(Nodo *nodo,char * palabraIn,char * palabraOut) {
    if(nodo != NULL){
        if (strcmp(convertirTolower(nodo -> palabraEsp),convertirTolower(palabraIn))==0) {
            strcpy(palabraOut,nodo -> defEsp);
        }
        buscarDefinicion(nodo -> espL,palabraIn,palabraOut);
        buscarDefinicion(nodo -> espL,palabraIn,palabraOut);
    }
}
```

Figura 3: Función buscarDefinicion

Función recursiva con el mismo funcionamiento de la función anterior pero esta entrega el significado de la palabra solicitada.

$T(n) = 2n$

2.3.2.3 insertarNodoEsp( )

Entradas: - AVL * árbol - char * palabra - char * otra - char * significado

Salidas: - Sin salidas.

```
void insertarNodoEsp(AVL *arbol, char *palabra, char *otra, char *significado ) {

    Nodo *nodo      = NULL;
    Nodo *siguiente = NULL;
    Nodo *ultimo    = NULL;

    if ( arbol->rootEsp == NULL ) {

        nodo = inicializarNodo();
        strcpy(nodo  -> palabraEsp,palabra);
        strcpy(nodo  -> palabraOtro,otra);
        strcpy(nodo  -> defEsp,significado);
        arbol -> rootEsp = nodo;


    } else {
        siguiente = arbol->rootEsp;
        ultimo = NULL;
        insertarEsp (nodo, siguiente,ultimo,palabra,otra,significado);
    }
}
```

Figura 4: Función insertarNodoEsp

Función encargada de realizar la primera inserción de un nodo en el árbol avl y en el caso que ya existan nodos en el árbol este llama a la función insertarEsp para que esta logra la inserción en la posición correcta. A la vez en esta función se realiza la inserción de los datos contenidos en el nodo, como la palabra, su traducción y su significado.

$T(n) = 2n + 2$

2.3.2.4 equilibrarEsp( )

Entradas: - nodo *Nodo - AVL * árbol

Salidas: - Sin salidas.

```
void equilibrarEsp(AVL *arbol, Nodo *nodo) {

    int diferencia1, diferencia2;
    diferencia1 = diferenciaAlturasEsp(nodo);
    if (diferencia1 > 1) {
        diferencia2 = diferenciaAlturasEsp(nodo->espL);
        if (diferencia2 < 0) {
            rotarIzquierdaEsp(arbol,nodo->espL);
        }
        rotarDerechaEsp(arbol,nodo);
    }
    if (diferencia1 < -1) {
        diferencia2 = diferenciaAlturasEsp(nodo->espR);
        if (diferencia2 > 0) {
            rotarDerechaEsp(arbol,nodo->espR);
        }
        rotarIzquierdaEsp(arbol,nodo);
    }

}
```

Figura 5: Función equilibrarEsp

Función encargada de realizar las rotaciones dentro del árbol avl a medida que cada nodo es insertado. Este a la vez realiza las rotaciones dobles en caso de ser necesarias verificando el sentido de las diferencias de altura en cada subÁrbol.

$T(n) = 8n$

2.3.2.5 insertarEsp( )

Entradas: - Nodo * nodo - Nodo * siguiente - Nodo * último - char * palabra

- char * otra - char * significado

Salidas: - Sin salidas.

```
void insertarEsp(Nodo *nodo, Nodo *siguiente, Nodo *ultimo, char *palabra, char *otra, char *significado) {

    int flag = 0;
    while( siguiente != NULL ) {
        ultimo = siguiente;
        int valor = strcmp(siguiente->palabraEsp,palabra);
        if (strcmp(siguiente->palabraEsp,palabra) < 0) {
            siguiente = siguiente->espR;
            flag = 1;

        } else if (strcmp(siguiente->palabraEsp,palabra) > 0) {
            siguiente = siguiente->espL;
            flag = 0;
        } else {
            return;
        }
    }
```

Figura 6: Función insertarEsp

Función encargada de realizar la comprobación de hacia qué lado de cada nodo va la inserción, ya sea hijo izquierdo o hijo derecho. Utiliza la función strcmp( ) para realizar la comparación según el valor ASCII referente a la palabra.

$T(n) = \mathbf{4n + 10}$

2.3.2.6 rotarIzquierdaEsp( )

Entradas: - Nodo * nodo - AVL * árbol

Salidas: - Sin salidas.

```
void rotarIzquierdaEsp(AVL *arbol, Nodo *nodo) {

    Nodo *raiz;
    raiz = nodo -> espR;
    if (nodo -> padreEsp == NULL) {
        arbol -> rootEsp = raiz;
    } else {
        if (nodo == nodo -> padreEsp -> espL) {
            nodo-> padreEsp -> espL = raiz;
        } else {
            nodo-> padreEsp -> espR = raiz;
        }
    }
    nodo -> espR = raiz -> espL;
    raiz -> espL = nodo;
    raiz -> padreEsp = nodo -> padreEsp;
    raiz -> espL -> padreEsp = raiz;
    if (raiz -> espR != NULL) {
        raiz -> espR -> padreEsp = raiz;
    }
}
```

Figura 7: Función rotarIzquierdaEsp

Función encargada de realizar la rotación del árbol gracias a un ajuste en los punteros de éste. Se toma el nodo padre de donde se requiere hacer la rotación y desde ahí empieza a modificar los punteros. Ésta función funciona de igual manera para la rotación en otro idioma como en la rotación derecha.

$T(n) = \mathbf{8n}$

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

Estos son los T(n) calculados del código:

Tabla 1: Tabla de tiempos de ejecución

Funciones	T(n)
leerArchivo ( )	$18n + 8$
crearAVL ( )	3
inicializarNodo ( )	8
insertarNodoEsp( ) - insertarNodoOtros( )	$4n + 10$
insertarEsp( ) - insertarOtro( )	$2n + 2$
inOrdenEsp( ) - inOrdenOtro( )	n
alturaEsp( ) - alturaOtro( )	$4n + 4$
diferenciaAlturasEsp( ) - diferenciaAlturasOtros( )	N
equilibrarEsp( ) - equilibrarOtros( )	$5n$
rotarDerechaEsp( ) - rotarDerechaOtros( )	2
rotarIzquierdaEsp( ) - rotarIzquierdaOtros( )	$8n$
buscarPalabraEsp ( ) - buscarPalabraOtro( )	$2n$
buscarDefinicion( )	$2n$
convertirTolower( )	$2n + 1$
menu( )	$27n+26$
main( )	$27n + 28$

El orden de complejidad de este código es O(n) = n

Durante el desarrollo de este código se logró un uso correcto de los punteros a los hijos de cada nodo para la formulación del TDA árbol, obteniendo así un O(n) óptimo donde por lo general el peor de los casos se representa como n debido que hay que realizar recorridos de todas las hojas del árbol. A su vez , el uso de memoria dinámica y la lectura de archivos funcionan de manera correcta. Respecto a las falencias en la optimización de este código se encuentra el uso de algoritmos repetidos impiden que se pudiera utilizar como un solo algoritmo como función genérica durante el código. Debido a lo anterior el código posee una extensión de aproximadamente el doble de lo que se pudo haber logrado respecto a usar un algoritmo genérico para ambas opciones, la palabra y su traducción. Otro punto que se pudo haber optimizado es la búsqueda en la cual se implementó usando un recorrido inOrden logrando que para buscar un elemento realice una búsqueda semejante a la de una lista ordenada. Para que se lograra un mejor resultado en la búsqueda, la mejor opción hubiera sido implementar búsqueda binaria desde el nodo raíz del árbol, logrando así que el recorrido de la búsqueda se reduzca a la mitad por el hecho de ir generando el descarte de un subárbol por cada vez que se baja un nivel durante la búsqueda binaria.

CAPÍTULO 4. CONCLUSIONES

Gracias a el desarrollo de este laboratorio se puede apreciar de mejor manera el manejo de los conocimientos ya adquiridos durante el desarrollo de los laboratorios anteriores, ya que en cada uno de éstos, se profundizaron temas que ahora nos facilitan el manejo de técnicas tales como la utilización de punteros, la lectura/escritura de archivos, el manejo con memoria dinámica en el lenguaje de Programación C el uso adecuado de un TDA y las facilidades que se logran debido a la implementación de éstos. Referente a los algoritmos utilizados estos trabajan de manera correcta, gracias a una correcta implementación del TDA árbol AVL, el cual tenía las mayores complicaciones en este código al momento de tener que trabajar las rotaciones con movimientos de punteros de una manera correcta, aunque existen situaciones las cuales podrían mejorar la efectividad del código tales como las mencionadas anteriormente referentes a una mejor búsqueda y una mejor implementación. Respecto a objetivos planteados en este laboratorio, el programa funciona de manera correcta tanto en Windows como en Linux/OSX entregando los resultados deseados sin mayor complicaciones.

CAPÍTULO 5. REFERENCIAS

Traducción.(2017,16) de abril. Wikipedia, La enciclopedia libre. Fecha consulta: 22:41, noviembre 23, 2017 from https://es.wikipedia.org/wiki/Traducción