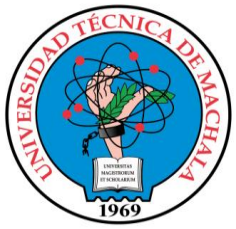


UTMACH

UNIDAD ACADÉMICA DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

TEMA:

IMPLEMENTACIÓN DE UN TAD QUE GESTIONE UNA LISTA ENLAZADA DE NODOS DE TIPO ÁRBOL AVL Y APLICACIÓN PRÁCTICA

TRABAJO PRÁCTICO DEL EXAMEN COMPLEXIVO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SISTEMAS

AUTOR:

ALVAREZ MORALES ALEXANDER MIGUEL

MACHALA - EL ORO

CESIÓN DE DERECHOS DE AUTOR

Yo, ALVAREZ MORALES ALEXANDER MIGUEL, con C.I. 0704776509, estudiante de la carrera de INGENIERÍA DE SISTEMAS de la UNIDAD ACADÉMICA DE INGENIERÍA CIVIL de la UNIVERSIDAD TÉCNICA DE MACHALA, en calidad de Autor del siguiente trabajo de titulación IMPLEMENTACIÓN DE UN TAD QUE GESTIONE UNA LISTA ENLAZADA DE NODOS DE TIPO ÁRBOL AVL Y APLICACIÓN PRÁCTICA

- Declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional. En consecuencia, asumo la responsabilidad de la originalidad del mismo y el cuidado al remitirme a las fuentes bibliográficas respectivas para fundamentar el contenido expuesto, asumiendo la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera EXCLUSIVA.

- Cedo a la UNIVERSIDAD TÉCNICA DE MACHALA de forma NO EXCLUSIVA con referencia a la obra en formato digital los derechos de:

a. Incorporar la mencionada obra al repositorio digital institucional para su democratización a nivel mundial, respetando lo establecido por la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0), la Ley de Propiedad Intelectual del Estado Ecuatoriano y el Reglamento Institucional.

b. Adecuarla a cualquier formato o tecnología de uso en internet, así como incorporar cualquier sistema de seguridad para documentos electrónicos, correspondiéndome como Autor(a) la responsabilidad de velar por dichas adaptaciones con la finalidad de que no se desnaturalice el contenido o sentido de la misma.

Machala, 18 de noviembre de 2015

1. INTRODUCCION

La gran mayoría de las aplicaciones software que se emplean están fundamentadas en el uso de Tipos Abstractos de Datos, que son efectuados por programadores de software o sencillamente reutilizados por terceras personas. La elección del Tipo Abstracto de Datos es de manera muy sencilla, ya que se la concluye de forma muy natural por las propias características del problema. (Sanchez Barreiro, 2011).

Por este motivo el presente informe busca implementar un TAD personalizado que gestione una lista enlazada de nodos de tipo árbol AVL, el mismo que sea verificable con una aplicación práctica donde se puedan asociar los nombres de los habitantes con las diferentes parroquias de la Ciudad de Machala. Para lograr con éxito esta implementación se debe conocer y sobre todo utilizar de manera correcta las estructuras de datos para escribir programas que utilicen eficientemente los recursos de la computadora. Este informe contiene diferentes temáticas, las cuales están estructuradas de la siguiente manera:

En la Introducción, se refiere a la contextualización, los indicadores del problema, el objetivo general y la ventaja competitiva del proyecto. El desarrollo, comprende la fundamentación de la respuesta, que se refiere al proceso de diseño de estructuras y algoritmos empleados, además de la aplicación práctica y resultados obtenidos.

1.1. MARCO CONTEXTUAL

En la actualidad los programadores definen el Tipo Abstracto de Datos a utilizarse para la solución de un problema, ya que en el mundo real un objeto es algo más complejo que la representación de números o caracteres que se han visto en los tipos de datos simples.

La abstracción de datos es la habilidad para innovar nuevos tipos de datos de manera más eficientes para una aplicación, teniendo como resultado programas más cortos, legibles y flexibles. Entre algunos ejemplos de la aplicación práctica del TAD tenemos, la implementación de conjuntos con sus operaciones básicas, Árboles de elementos, de los algoritmos LIFO Y FIFO y de grafos.

Se han almacenado los habitantes de n parroquias de la Ciudad de Machala en un archivo, donde cada registro tiene el nombre de la parroquia y el número de habitantes. Por lo que se desea asociar los nombres de cada habitante a cada parroquia y así mismo guardar en un archivo dicha información de manera detallada.

Ante la necesidad presentada se va a desarrollar un proyecto, donde se implementará un Tipo de Dato Abstracto (TAD), con la finalidad que gestione una lista enlazada de nodos de tipo Árbol AVL y su aplicación práctica.

1.2. PROBLEMA

1.2.1. Planteamiento

Los datos son los valores que manejamos en la resolución de un problema, tanto los valores de entrada, como los de proceso y los de salida. Es decir, los datos son información y por lo tanto, para manejarlos se requieren varios tipos de datos.

Un tipo de dato se puede definir como un conjunto de valores y un conjunto de operaciones definidas por esos valores. Clasificar los datos en distintos tipos aporta muchas ventajas, como por ejemplo indicarle al compilador la cantidad de memoria que debe reservar para cada instancia dependiendo del tipo de dato al que pertenezca. (Ochoa, 2014)

Es por esto que se tiene la necesidad de construir un TAD personalizado que gestione árboles AVL de cualquier tipo de objeto con la ayuda de una estructura de datos llamada listas simplemente ligadas que nos permita almacenar datos.

1.3. OBJETIVO GENERAL

Implementar una estructura algebraica TAD (Tipo Abstracto de Datos), con en el uso de listas simplemente enlazadas, para la gestión de árboles AVL o binarios.

2. DESARROLLO

2.1. MARCO TEÓRICO

2.1.1. Listas Enlazadas

Este tipo de estructura es de forma lineal y permite almacenar elementos comúnmente llamados nodos, en donde cada uno de ellos puede almacenar datos y un puntero hacia los otros nodos. También son estructuras dinámicas que pueden almacenar datos que cambian con frecuencia. Las listas enlazadas se propagan y se retraen para que sean más sencillas al añadir o eliminar información. Permiten almacenar información en diferentes posiciones de memoria; y se almacena en los nodos. Estos nodos tienen dos campos uno para almacenar el valor del elemento y otro para el enlace de la posición del siguiente elemento. (*Ver Figura 1*).

2.1.1.1. Listas Enlazadas Simples

Es un conjunto de nodos con una sola dirección y forman una estructura de datos lineal. Cada nodo es un objeto que almacena un elemento (dato) y una dirección a otro nodo.

La dirección que guarda un nodo de otro nodo se llama puntero y el salto que los une, se llama salto de puntero. El primer nodo de una lista es la cabecera y el último se denomina final.

Un nodo puede decretar quien se encuentra después de él pero no puede saber quién se encuentra antes. (*Ver Figura 2*)

Para dirigirnos a un nodo se usa el puntero anterior con excepción del puntero primero, al que se accede a través de un puntero externo a la lista, siendo así usado el puntero INICIO que es el que apunta constantemente al primero y el puntero ultimo apunta continuamente al último. (*Ver Figura 3*)

Crea_Inicio: Permite crear una lista, añadiendo cada nuevo nodo al inicio de la lista. P y Q son variables de tipo puntero. Los campos del nodo son INFO, y LIGA es de tipo apuntador, P apunta al inicio de la lista. RES es una variable de tipo entero

- 1. Crear (P) {Se crea un primer nodo de la lista simplemente ligada}
- 2. Leer P^.INFO
- 3. Hacer P^.LIGA(NILL
- 4. Escribir "¿Desea ingresar más nodos a la lista? SI:1 , NO : 0"
- 5. Leer RES
- 6. Mientras (RES=1) Repetir

Crear (Q)

Leer Q^.INFO

Hacer Q^.LIGA (P y P (Q

Escribir “¿Desea ingresar más nodos a la lista? SI: 1, NO: 0"

Leer RES

- 7. Fin- mientras
- 8. Fin

(Ver Figura 4)

2.1.1.2. Operaciones Básicas de Listas Enlazadas Simples

La **operación de recorrido,** radica en pasar por cada nodo que integran la lista. Para recorrer estos nodos se inicia con el primero tomando su valor del campo Liga y se avanza al segundo, y así repetidamente hasta llegar al final de los nodos, teniendo el campo LIGA un valor NILL.

La **operación de inserción**, consiste en insertar un nuevo nodo a la lista, dependiendo de la posición en la que se va a insertar el nodo, pueden existir diferentes casos:

Insertar un nodo al inicio de la lista.

- Insertar un nodo al final de la lista.
- Insertar un nodo antes que otro cuya información es X.
- Insertar un nodo después de otra cuya información es X.

a) Inserción al inicio de una lista simplemente ligada

En este caso el nodo se coloca al principio de la lista, convirtiéndose en la primera de ellas. Este paso es simple como se puede apreciar en el siguiente algoritmo.

Inserta_inicio (PP, DATO): Este algoritmo inserta un nodo al inicio de una lista. P es el apuntador al primer nodo, y DAO es la información que se guarda en el nuevo nodo. Q es una variable de tipo apuntador, INFO y LIGA son los campos de cada nodo de la lista.

1. Crear (Q)

2. Hacer Q^.INFO (DATO.Q^.LIGA (P y P (Q (*Ver Figura 5*)

- b) Inserción al final de una lista simplemente Ligada

En este caso el nuevo nodo se coloca al final de la lista, convirtiéndose en el último.

Inserta_final (P; DATO): Este algoritmo incrusta un nodo al final de una lista. P es el apuntador al primer nodo, y DATO es la información que se guardara en el nuevo nodo, Q y t son variables de tipo puntero. INFO y LIGA son los campos de los nodos de la lista.

- 1) Hacer T(P
- 2) Mientras (T^.LIGANIL) repetir

Recorre la lista hasta llegar al último elemento

Hacer T (T^.LIGA

- 3) Fin-Mientras
- 4) Crear {Q}
- 5) Hacer Q^.INFO (DATOQ^.LIGA (NIL y T^.LIGA(Q

(*Ver Figura 6*)

- c) Inserción de un nodo antes que otro en la lista simplemente ligada

En esta clase de inserción, el nuevo nodo se sitúa antes de otro nodo.

Inserte_antes_X (P, DATO, X): Este algoritmo inserta un nodo dado como referencia en una lista simplemente ligada. P es el apuntador al primer nodo de la lista, DATO es la información se guarda en el nuevo nodo, y X representa la información del nodo dado. Q, X y T son variables de tipo apuntador, INFO y LIGA son los campos de los nodos. BAND es una variable de tipo entero

- 1) Hacer Q(P y BAND (1
- 2) Mientras ((Q^.INFO X) y (BAND =1) repetir

Si (Q^.LIGANIL) Entonces

Hacer T (Q y Q^.LIGA

Sino

Hacer BAND (0

Fin - Si

Fin - Mientras

- 3) Si (BAND=1) Entonces

Crear(X)

Hacer X^.INFO (DATO

Si (P=Q) Entonces {El nodo dado como referencia es el primero}

Hacer X^.LIGA (P y P (X

Si no

Hacer T^.LIGA(X y X^.LIGA (Q

Fin - Si

Si no

Escribir "el nodo dado como referencia no se encuentra en la lista”

Fin – Si

- d) Inserción de un nodo después de otro en una lista completamente ligada

En este tipo de inserción, el nuevo nodo se debe situar después de otro nodo.

Inserta_despues_X (P; DATO, X): Este algoritmo inserta un nodo después de otro nodo. P es el apuntador al primer nodo de la lista, DATO indica la información que se guarda el nuevo nodo, y X representa a la información del nodo. Q y T son variables de tipo apuntador, INFO y LIGA son los campos del nodo de la lista y BAND es una variable de tipo entero.

- Hacer Q(P y BAND (1
- Mientras ((Q^.INNFOX)y(BAND=1)) Repetir

Si Q ^.LIGANIL Entonces

Hacer Q (Q^.LIGA

Si no

Hacer BAND (0

- Fin-Si
- Fin - Mientras
- Si (BAND=1) Entonces

Crear (T)

Hacer T^.INFO (DATO, T^.LIGA (Q^.LIGA (T

Si no

Escribir "El nodo dado como referencia no se encuentra en la lista"

- Fin Si

(*Ver Figura 7*)

- **Eliminación** en listas simplemente enlazadas

Esta operación radica en eliminar un número de la lista y liberar el espacio de memoria. De acuerdo a la posición en la que se encuentre, se pueden encontrar diferentes casos:

- Eliminar el primer nodo
- Eliminar el ultimo nodo
- Eliminar un nodo con información X
- Eliminar el nodo anterior al nodo con información X
- Eliminar el nodo posterior al nodo con información X

- a) Eliminar el primer nodo de una lista simplemente ligada

Se debe determinar el apuntador al inicio de la lista. Si quedara vacía, entonces apuntaría a NILL.

Elimina_Inicio (P): Permite eliminar el primer elemento de una lista. P es el apuntador al primer nodo de la lista

- Hacer Q (P {Si la lista tuviera solo un elemento a P se le asigna NIL, que es el valor de Q^.LIGA. En caso contrario, queda con la dirección del siguiente nodo}
- Hacer P (Q^.LIGA. {redefine el puntero al inicio de la lista}
- Quitar (Q)

(*Ver Figura 8*)

- b) Eliminación del ultimo nodo de la lista simplemente ligada

Aquí se elimina el último nodo de una lista. Es significativo prestar atención que para obtener el último nodo, se debe recorrer toda la lista, menos si se usara un apuntador que indique su final.

Elimina_ultimo (P): Permite eliminar el último nodo de una lista. P es el apuntador al primer nodo de la lista. Q y T son variables de tipo apuntador. INFO y LIGA son los campos de los nodos de la lista

- Hacer Q(P
- Si (P^.LIGA = NIL) {Se verifica si la lista tiene solo un nodo} entonces

Hacer P (NIL

Sino

Mientras (Q^.LIGA!=NIL) Repetir

Hacer T (Q y Q (Q^.LIGA

Fin-Mientras

Hacer T^.LIGA (NIL

- Fin-Si (*Ver Figura 9*)

    - c) Eliminar un nodo con información X de una lista simplemente ligada

Elimina_X (P, X): Elimina un nodo con información X de una lista. P es el apuntador al primer nodo de la lista. Q y T son variables de tipo apuntador. BAND es una variable de tipo entero, INFO y LIGA son los campos de los nodos de la lista

- Hacer Q(P y BAND(1
- Mientras ((Q^.INFO X) y (band = 1)) Repetir

Si Q^. LIGA NIL entonces

Hacer T (Q y Q (Q^.LIGA

Sino

Hacer BAND (0

FIN SI

FIN - Mientras

- Si (BAND=0) entonces

Escribir "El elemento con información X no se encuentra en la lista"

Si (P= Q) {se verifica si el elemento a eliminar es el primero} entonces

Hacer P (Q^.LIGA

Sino

Hacer T^.LIGA (Q^.LIGA

FIN-SI

(*Ver Figura 10*)

- d) Eliminar el nodo anterior al nodo con información X en una lista simplemente ligada

elimina_ante_X (P, X): Elimina el nodo anterior al nodo que contiene la información X en una lista. P es el apuntador al primer nodo de la lista. Q, T y R son variables de tipo apuntador. BAND es una variable de tipo entero, INFO y LIGA son los campos de los nodos de la lista

2.1.2. Árboles AVL

Estos árboles están siempre equilibrados por lo que la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha. Su factor de equilibrio puede ser guardado en cada nodo.

Para obtener esta propiedad de equilibrio, la inserción y la eliminación de los nodos se realizan de manera específica. Si en el momento de efectuar una operación de inserción o eliminación se rompe la condición de equilibrio, se tiene que ejecutar una serie de rotaciones en los nodos. (*Ver Figura 11 y 12*)

2.1.2.1. Operaciones

2.1.2.2. Rotación simple a la derecha

Un árbol que tiene como raíz (r) y los hijos: izquierdo (i) y derecho (d), lo que crearemos será un nuevo árbol donde la raíz será la raíz del hijo izquierdo. Como nuestro hijo izquierdo pondremos al hijo izquierdo de i y como hijo derecho fundaremos un nuevo árbol que tendrá como raíz la raíz del árbol (r), el hijo derecho de i, será el hijo izquierdo y el hijo derecho será el hijo derecho del árbol (d). (*Ver Figura 13*)

2.1.2.3. Rotación simple a la izquierda

Un árbol que tiene como raíz (r) y los hijos: izquierdo (i) y derecho (d), lo que crearemos será un nuevo árbol cuya raíz será la raíz del hijo derecho. Como nuestro hijo derecho pondremos al hijo derecho de d y como hijo izquierdo fundaremos un nuevo árbol que tendrá como raíz la raíz del árbol (r), el hijo izquierdo de d, será el hijo derecho y el hijo izquierdo será el hijo izquierdo del árbol (i). (*Ver Figura 14*)

2.1.2.4. Rotación doble a la derecha

La Rotación doble a la Derecha son dos rotaciones simples, primero la rotación simple a la izquierda y luego la rotación simple a la derecha. A continuación en ejemplo: (*Ver Figura 15*)

2.1.2.5. Rotación doble a la izquierda

La Rotación doble a la Izquierda son dos rotaciones simples, primero rotación simple derecha y luego rotación simple izquierda. (*Ver Figura 16*)

2.1.2.6. Inserción

La inserción puede ser ejecutada ingresando un valor dado al árbol como si este se tratara de un árbol de búsqueda binario desequilibrado. Proceso de inserción:

1. Buscar para encontrar la posición de inserción o modificación
2. Insertar el nuevo nodo teniendo en cuenta que el factor de equilibrio este “equilibrado”
3. Reanudar el camino de búsqueda, comprobando el equilibrio de los nodos, y re-equilibrando si fuese necesario. (*Ver Figura 17*)

2.1.2.7. Extracción

Una extracción acarrea una deducción de la altura de la rama de donde se eliminó y habrá como consecuencia un cambio en el factor de equilibrio del nodo padre, pudiendo necesitarse una rotación. (*Ver Figura 18*)

2.1.2.8. Recorrido de Árboles

2.1.2.9. Pre Orden

Para realizar un recorrido en pre orden, se debe ejecutar los siguientes pasos recursivamente en cada nodo:

1. Se comienza visitando la raíz
2. Luego se pasa por el sub-árbol izquierdo
3. Y finalmente se recorre el sub-árbol derecho

2.1.2.10. In Orden

Para realizar un recorrido en in orden (simétrico), se debe de ejecutar los siguientes pasos recursivamente en cada nodo:

1. Se comienza visitando el sub-árbol izquierdo
2. Luego recorremos la raíz
3. Y finalmente se pasa por el sub-árbol derecho

2.1.2.11. Post Orden

Para realizar un recorrido en post orden, se debe de ejecutar los siguientes pasos recursivamente en cada nodo:

1. Se comienza visitando el sub-árbol izquierdo
2. Luego recorremos el sub-árbol derecho
3. Y finalmente se pasa por la raíz

La diferencia existente entre Pre Orden, In Orden y Post Orden se debe al recorrido de la raíz, ya que en las tres formas lo primero en recorrer son los sub-árbol izquierdo y derecho.

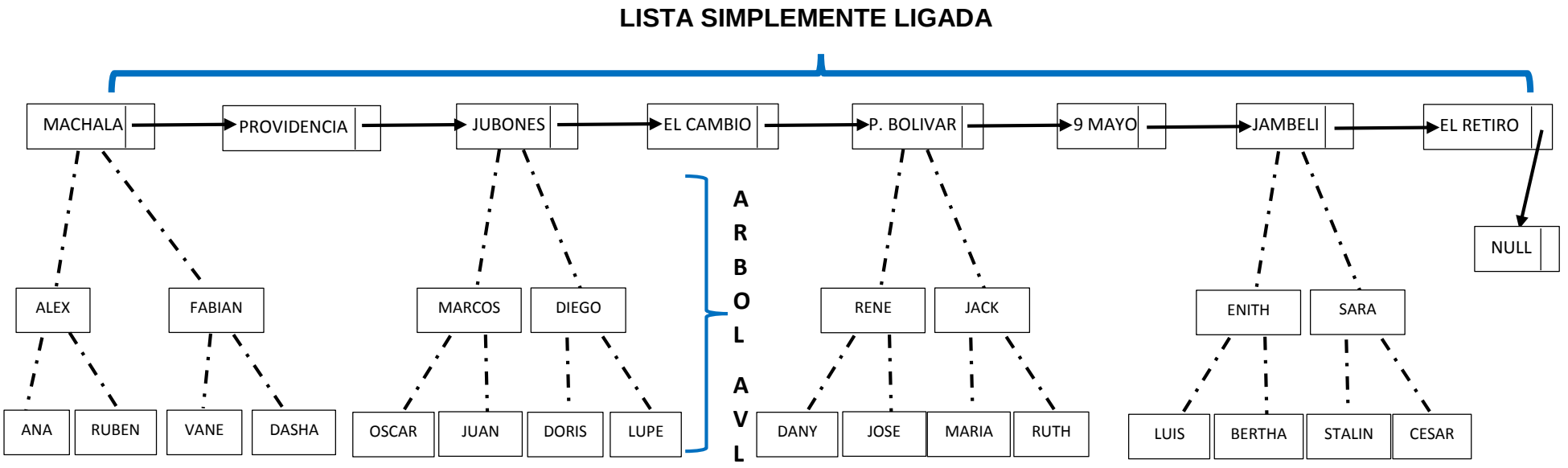
2.2. MARCO METODOLÓGICO

Un tipo de dato abstracto es una estructura algebraica, que consiste en el dominio de objetos y de operaciones que se pueden efectuar con esos objetos. Los pasos que se siguen radican en especificar el nuevo tipo de dato en términos de código y en algún lenguaje de programación. Para construir tipo de dato abstracto es preciso dos pasos: definir los tipos de datos e implementar dicho tipo de dato abstracto como una clase.

La construcción del presente proyecto se realiza con el lenguaje de programación Java, un lenguaje orientado a objetos. Para la definición del tipo de datos se ha escogido el tipo OBJECT. A continuación se detalla el diagrama del diseño de la implementación del TAD basada en una aplicación práctica donde se asocian los nombres de los habitantes con las parroquias de la ciudad de Machala.

Comenzando con la selección del tipo de estructura de datos, que en este proyecto son las listas simplemente enlazadas o ligadas con el fin de guardar en ellas los nombres de las parroquias de la ciudad de Machala. Las parroquias son las siguientes: Machala, El Cambio, La Providencia, El Retiro, 9 de Mayo, Jubones, Jambelí y Puerto Bolívar, cada una de estas parroquias vendrían a ser la raíz de un nodo de tipo Árbol AVL o binario el mismo que estaría ordenado por la forma InOrden para de esta manera poder optimizar la búsqueda de la información contenida dentro de un hijo (nodo).

Figura 19 Representación de la estructura de un TAD gestionando un árbol AVL



El siguiente código fuente permite la creación de la lista simplemente ligada en este caso la clase se llama **ListaSimple**, la cual contiene un método **add** que agrega un objeto a nuestra lista simplemente ligada.

Figura 20 Código Fuente de la Clase ListaSimple

```
/**
 *Lista Enlazada Simple
 */
public class ListaSimple {

    private NodoSimple primero;

    /**
     * Método para agregar un objeto a nuestra lista enlazada simple
     * @param obj
     */
    public void add(Object obj) {
        NodoSimple aux = null;
        if (primero == null) {
            primero = new NodoSimple(obj);
        } else {
            aux = primero;
            while (aux.getSiguiente() != null) {
                aux = aux.getSiguiente();
            }
            aux.setSiguiente(new NodoSimple(obj));
        }
    }
```

El siguiente código fuente permite la creación del nodo del árbol AVL, llamándose su clase **NodoSimple.**

Figura 21 Código Fuente de la creación del árbol AVL

```
package lista;

import avlnuevo.ArbolAVL;

public class NodoSimple {

    private Object informacion;
    private NodoSimple siguiente;

    public NodoSimple() {
        this.informacion = new Object();
        this.siguiente = null;
    }

    public NodoSimple(Object informacion) {
        this.informacion = informacion;
        this.siguiente = null;
    }

    public Object getInformacion() {
        return informacion;
    }

    public void setInformacion(Object informacion) {
        this.informacion = informacion;
    }

    public NodoSimple getSiguiente() {
        return siguiente;
    }

    /**
     * @param siguiente the siguiente to set
     */
    public void setSiguiente(NodoSimple siguiente) {
        this.siguiente = siguiente;
    }
```

Código fuente que permite la creación del nodo del árbol AVL, clase llamada NodoAvl.

Figura 22 Código Fuente de la creación del nodo AVL

```
package avlnuevo;

public class NodoAVL{

    public int factorBalance;
    private NodoAVL hizq;
    private NodoAVL hder;
    private Object informacion;

    // Constructor
    public NodoAVL() {
        factorBalance = 0;
        hizq = hder = null;
    }

    // Constructor
    public NodoAVL(Object per) {
        factorBalance = 0;
        hizq = hder = null;
        informacion=per;
    }

    // Metodos get y set
    //-----------------------------------
    // Establece el nodo izquierdo
    public void setHizq(NodoAVL Hizq) {
        hizq = Hizq;
    }


    // Devuelve el nodo izquierdo
    public NodoAVL getHizq() {
        return hizq;
    }

    //-----------------------------------------
    // Establece el nodo derecho
    public void setHder(NodoAVL Hder) {
        hder = Hder;
    }

    // Devuelve el nodo derecho
    public NodoAVL getHder() {
        return hder;
    }

    // Establece el factor balance
    public void setFactorBalance(int FacBal) {
        factorBalance = FacBal;
    }

    // devuelve el factor balance
    public int getFactorBalance() {
        return factorBalance;
    }

    /**
     * @return the informacion
     */
    public Object getInformacion() {
        return informacion;
    }

    /**
     * @param informacion the informacion to set
     */
    public void setInformacion(Object informacion) {
        this.informacion = informacion;
    }
```

2.3. RESULTADOS

A continuación se presenta las imágenes de las principales funciones de la implementación de un TAD que gestiona una lista enlazada de nodos de tipo árbol AVL y su comprobación práctica a través de un sistema informático realizado en el lenguaje Java.

Figura 23 Selección y carga del archivo Parroquias.txt

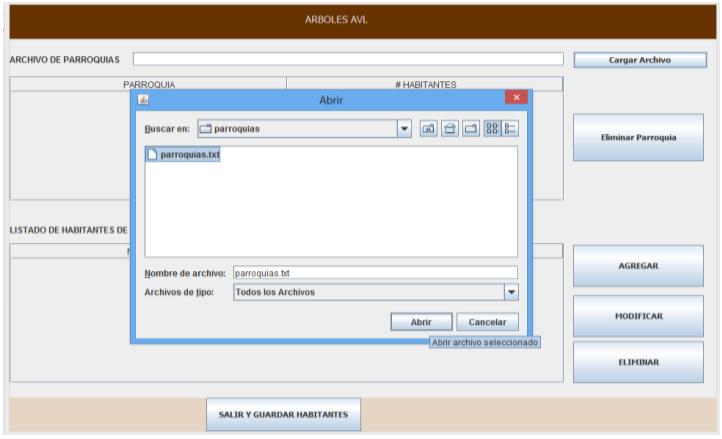


Figura 24 Listado del contenido del archivo parroquias .txt

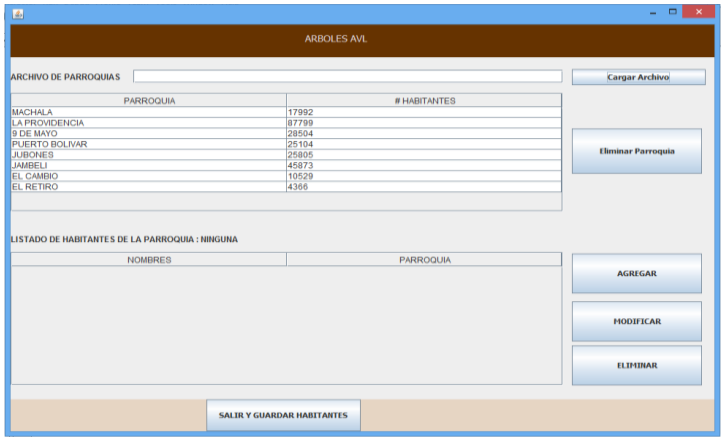


Figura 25 Creación del nombre de un nuevo habitante

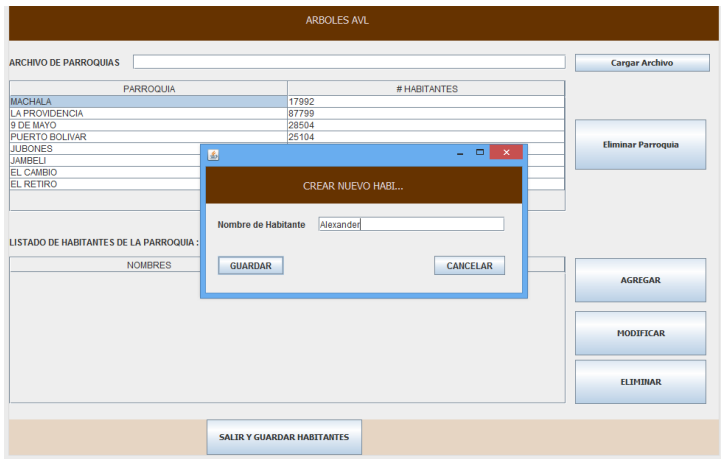


Figura 26 Listado de los nombres de habitantes creados en la parroquia Machala

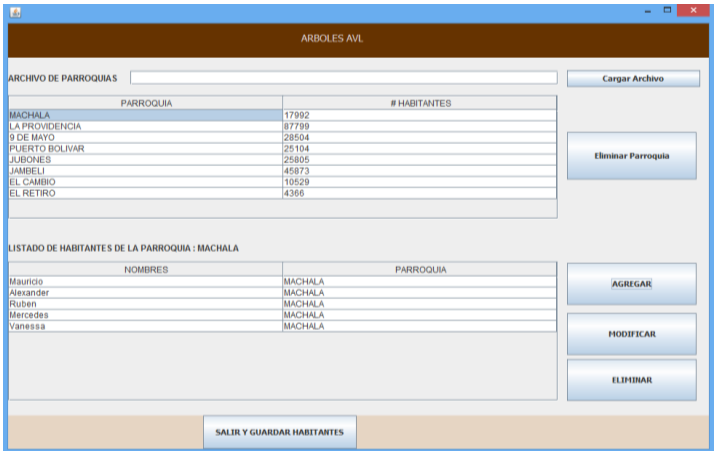


Figura 27 Modificación del nombre del habitante Alexander perteneciente a la parroquia 9 de Mayo por el habitante Fabián

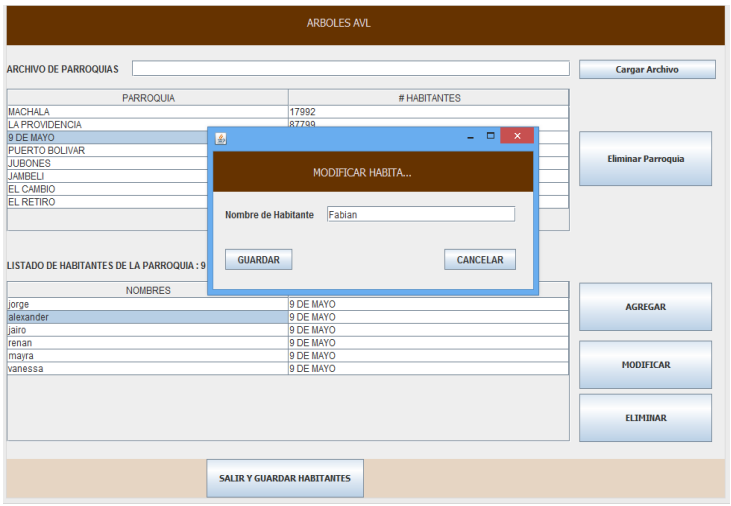


Figura 28 Listado de los habitantes de la parroquia 9 de Mayo con la modificación realizada.

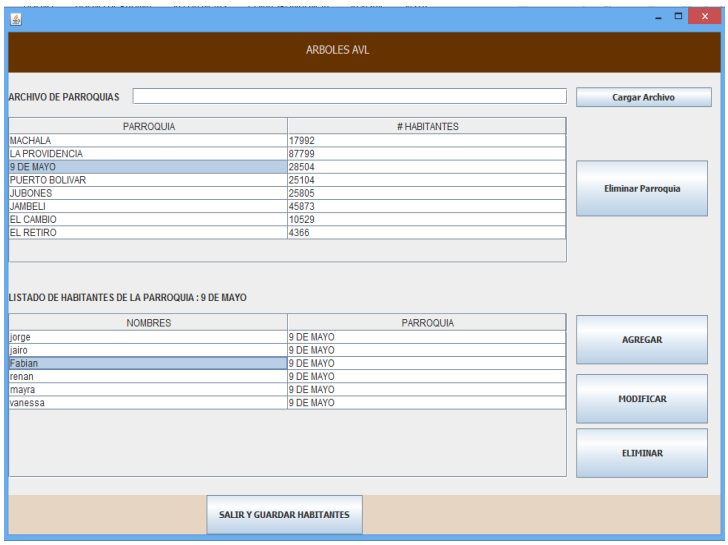


Figura 29 Eliminación del habitante Renán de la parroquia 9 de Mayo

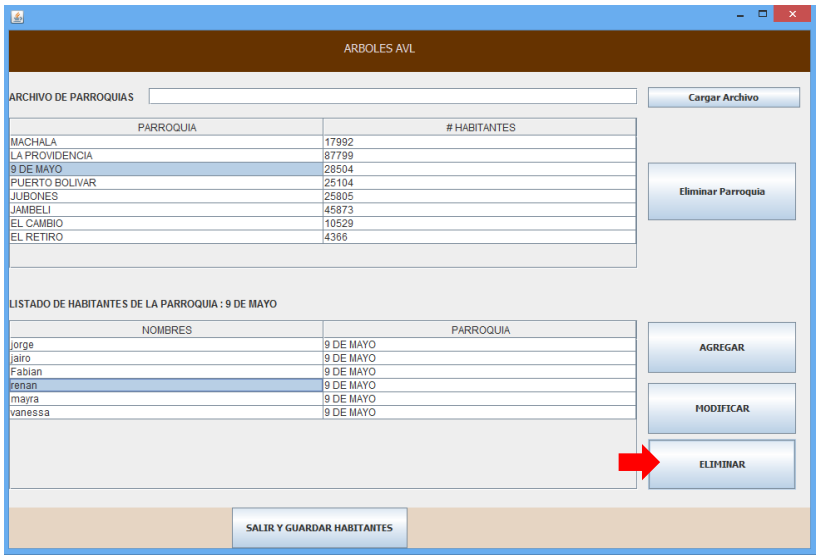


Figura 30 Listado de la parroquia 9 de Mayo sin el habitante Renán

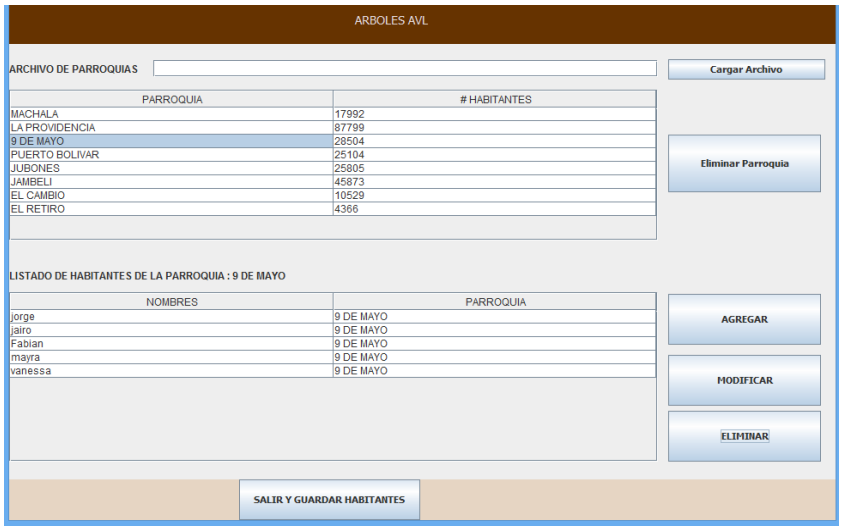


Figura 31 Eliminación de la parroquia 9 de Mayo

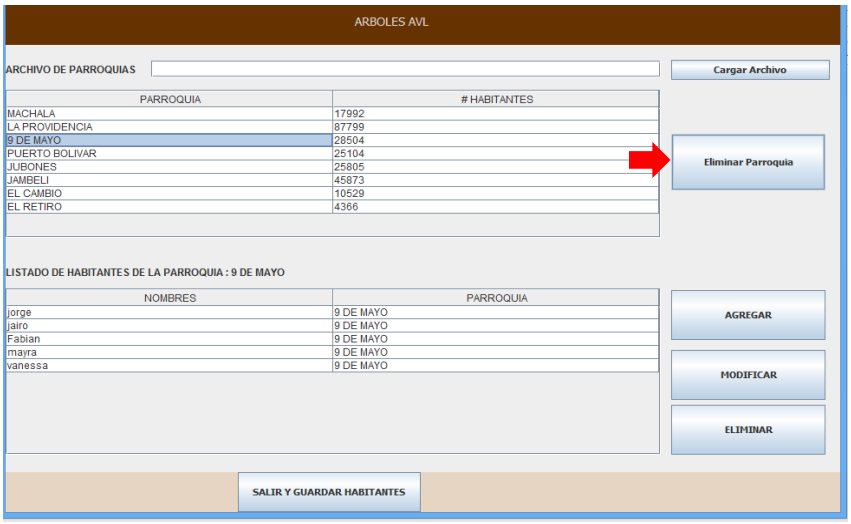


Figura 32 Listado de los nombres de los habitantes que pertenecían a la parroquia 9 de Mayo y fueron agregados a la parroquia Machala

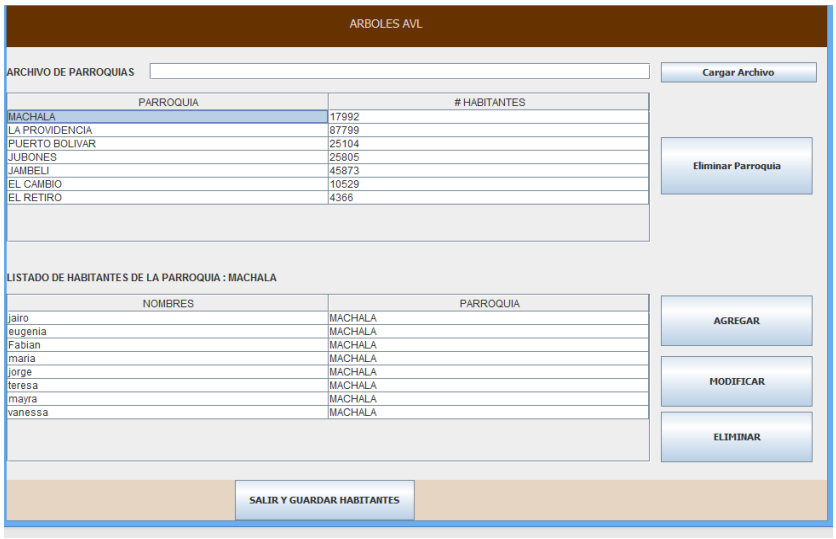


Figura 33 Selección del sitio donde se va a guardar la información que contiene cada parroquia

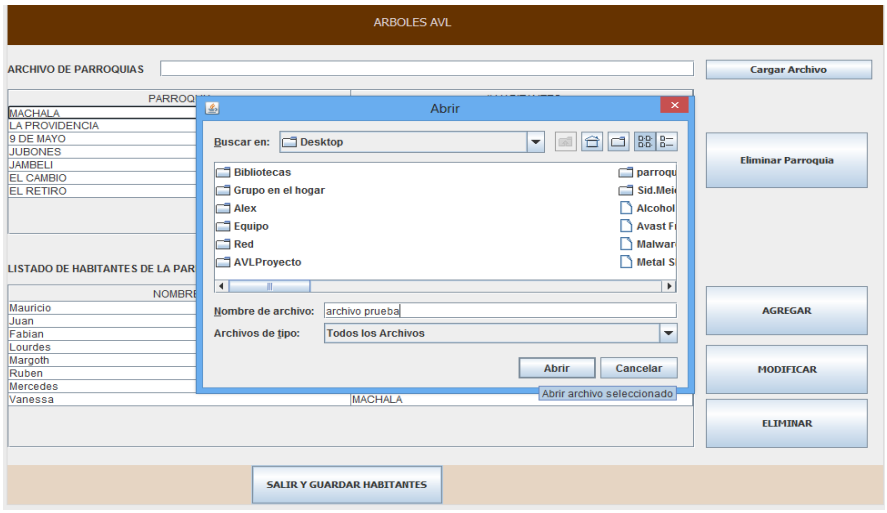
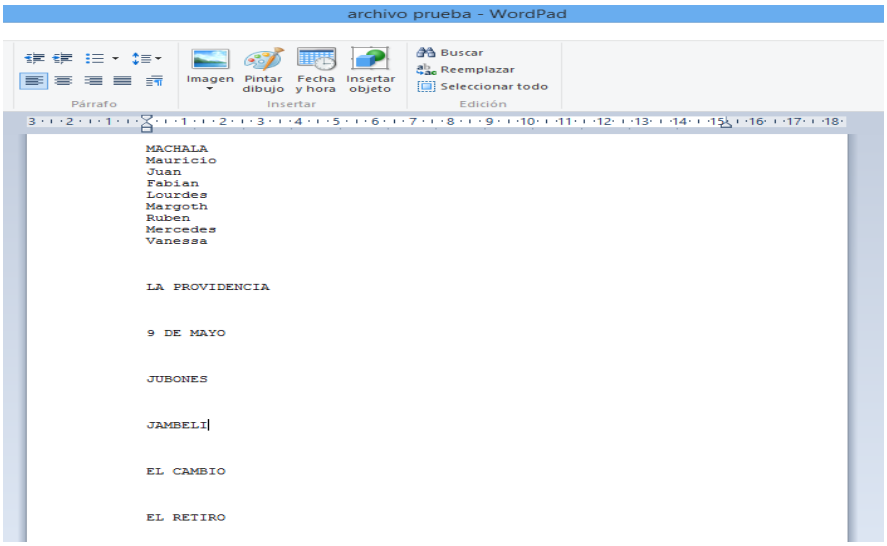


Figura 34 Información guardada en el archivo llamado prueba



3. CONCLUSIONES

- Un árbol utilizado como estructura de datos permite almacenar una cantidad significativa de datos de forma ordenada, y se representa con un conjunto de nodos entrelazados entre sí por medio de ramas, siendo así un nodo base único, denominado raíz. En dicho árbol un padre puede tener varios hijos pero un hijo solo puede tener un padre, ya que desde la raíz se puede llegar a cualquier nodo avanzando por las ramas y atravesando los sucesivos niveles estableciendo así un camino.

- El recorrido del árbol afecta a la totalidad de los nodos, obligando al árbol saber dónde estamos ubicados, y así cuando se extienda el algoritmo que se ha creado sea más fácil de ubicarnos. Debido a que si nos ubicamos mal en un nodo del sistema realizado en este proyecto puede crear un mal funcionamiento del algoritmo, borrar información equivocada o que ocurra un desbordamiento de datos.

- Una aplicación que está fundamentada en listas simples ligadas, va a permitir tener un fácil desarrollo, siempre y cuando se trabaje con lenguajes de programación como *Java*, ya que este lenguaje brinda muchos beneficios al trabajar con estructuras.

- Usar árboles AVL o árboles binarios de búsqueda permiten hallar el elemento deseado sin perder mucho tiempo ya que cada uno de sus nodos cumplen una condición, donde todos los nodos del subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz.

Reporte de Similitud URKUND

U R K U N D

Urkund Analysis Result

Analysed Document: trabajo de titulacion.docx (D16346007)
Submitted: 2015-11-23 21:26:00
Submitted By: amam_13_87@hotmail.com
Significance: 9 %

Sources included in the report:

INFORME EXPOSICION.docx (D16291588)
PERFIL PROYECTO INTEGRADOR 9(1).doc (D9420695)
Barcia_Miranda.docx (D14122807)
http://garciagregorio.webcindario.com/ed/arboles_clasificacion.pdf
https://es.wikipedia.org/wiki/%C3%81rbol_AVL https://prezi.com/njksrbj3umi7/copy-of-listas/
http://www.ammanu.edu.jo/wiki1/es/articles/%C3%A1/r/b/%C3%81rbol_AVL_2a9b.html
https://prezi.com/w3dr1sm85bfj/arbol-avl/
http://www.utim.edu.mx/~svalero/docs/resumen4B.doc
https://prezi.com/akkogh68xw8m/arboles-avl/

Instances where selected sources appear:

21