

INSTITUTO TECNOLÓGICO DE MEXICO

CAMPUS ORIZABA

NOMBRE DE LA ASIGNATURA:
ESTRUCTURA DE DATOS

TEMA:
REPORTE DE PRACTICA TEMA 4 ESTRUCTURAS NO LINEALES

NOMBRE DE LOS INTEGRANTES:
VELAZQUEZ SARMIENTO CELESTE - 21010223
ESPINDOLA OLIVERA PALOMA – 21010186

CARRERA:
INGENIERIA INFORMATICA

GRUPO:
3a3A

Contenido

INTRODUCCION	3
MARCO TEÓRICO	3
COMPETENCIA(S) ESPECÍFICA(S):	5
RECURSOS, MATERIALES Y EQUIPO.....	5
DESARROLLO DE PRÁCTICAS	6

INTRODUCCION

En este reporte de la unidad 4 se explicarán todos los códigos/programas que en este caso abarcan los temas de doble liga, estructuras colas, estructuras no lineales, etc. Este reporte tiene el propósito de que incluyamos y documentemos lo visto durante esta unidad. Además, nos beneficiará en la parte de que repasaremos todo lo visto en las unidades anteriores y recalcar nuestros conocimientos prácticos para resolver problemas por nuestra propia cuenta basándonos en apuntes y temas/documentos previos para saber si los resultados son los esperados al finalizar no solo cada unidad sino la materia de estructura de datos en el presente semestre. El presente proyecto académico está dirigido a estudiar y comprender la forma en cómo se trabaja con nodos en listas simplemente enlazadas.

En esta unidad se debe conocer, identificar y aplicar las estructuras no lineales en la solución de problemas del mundo real. Para poder aprender de ello se necesita consultar en las fuentes bibliográficas la terminología sobre árboles como también practicar ejercicios y buscar información ayudara para su mayor comprensión. Utilizando un lenguaje de programación podemos implementar las operaciones básicas (insertar, eliminar, buscar) en un árbol binario de búsqueda, así como los recorridos en PreOrden, InOrden y PostOrden. En si podemos decir que aprenderemos conceptos y aplicaciones de por ejemplo: Concepto de árbol y su clasificación de árboles al igual las operaciones básicas sobre árboles binarios que sin duda son herramientas que sirven y servirán en nuestra vida escolar.

MARCO TEÓRICO

Listas Simples Enlazadas

Las listas simplemente enlazadas son estructuras de datos semejantes a las estructuras array salvo que el acceso a un elemento no se hace mediante un índice, sino mediante un puntero. La asignación de memoria se la realiza durante la ejecución.

En una lista simplemente enlazada, los elementos son contiguos en lo que concierne al enlazado.

En una lista simplemente enlazada los elementos están dispersos.

Es decir, que cada elemento se almacena en un lugar de memoria que le asigna el ordenador de forma aleatoria.

Esta asignación como ya se mencionó, se la realiza durante la ejecución.

Cada dirección de memoria designada a una lista, estará ocupada por nodos.

Nodos

Una lista se compone de nodos, que son estructuras de datos que nos permiten registrar datos de interés. Para que estos nodos se conviertan en una lista, debe existir un enlace entre ellos, que en términos más propios se los conoce como apuntadores o punteros.

Funciones

En listas simplemente enlazadas, existen algunas funciones principales o elementales:

- Agregar. - Una de la característica de las listas simplemente enlazadas, y de cualquier otro tipo de estructura similar, es que es dinámico, es por eso que una lista no es fija, más al contrario, podemos agregarle nodos al principio, al final, o en cualquier posición de la lista.
- Eliminar. - Así como podemos ir agregando nuevos nodos a nuestra lista, también podemos eliminar nodos de la misma; una de las razones principales, para liberar espacio en memoria ya que es posible el ya no estar utilizando ese nodo.
- Buscar. - Podemos obtener, mediante algoritmos de búsqueda, los datos o direcciones de uno o varios nodos de la lista.

COMPETENCIA(S) ESPECÍFICA(S):

Comprende y aplica estructuras no lineales para la solución de problemas.

Genéricas:

- Habilidad para buscar y analizar información proveniente de fuentes diversas.
- La comprensión y manipulación de ideas y pensamientos.
- Metodologías para solución de problemas, organización del tiempo y para el aprendizaje.
- Habilidad en el manejo de equipo de cómputo
- Capacidad para trabajar en equipo.
- Capacidad de aplicar los conocimientos en la práctica.
- Elaborar un cuadro sinóptico o esquema con la clasificación de los árboles y sus aplicaciones.
- Implementar las operaciones básicas de inserción, eliminación y búsqueda en un árbol binario.

RECURSOS, MATERIALES Y EQUIPO

- Computadora
- Java
- Lectura de los materiales de apoyo del tema 4
- Notas de clase (problemas resueltos en clase y materiales de trabajo de la profesora).

DESARROLLO DE PRÁCTICAS

DATOS DESORDENADOS DOBLE LIGA

```
toolsList.java  busquedas.java  TesArbol.java  ArbolBin.java  ArbolBin.java  DatosDesordenadosDobleLiga.java  x
1 package DobleLiga;
2
3 import MemoriaDinamica.Nodo;
4
5
6
7 public class DatosDesordenadosDobleLiga<T> implements OperaTDA<T>{
8
9     private Nodito puntero;
10    private Nodito f;
11
12    public DatosDesordenadosDobleLiga() {
13        puntero = null;
14    }
15
16    @Override
17    public void insertarFrente(T dato) {
18        Nodito p = new Nodito(dato);
19        if(isListaVacia()) {
20            puntero=p;
21            f=p;
22        }else {
23            p.der=puntero;
24            puntero.izq=p;
25            puntero=p;
26        }
27    }
28
29    @Override
30    public void insertarFinal(T dato) {
31        Nodito p= new Nodito(dato);
32        if(isListaVacia()) {
33            puntero=p;
34        }else {
35            f.der=p;
36            p.izq=f;
37        }
38        f=p;
39    }
40
41    @Override
```

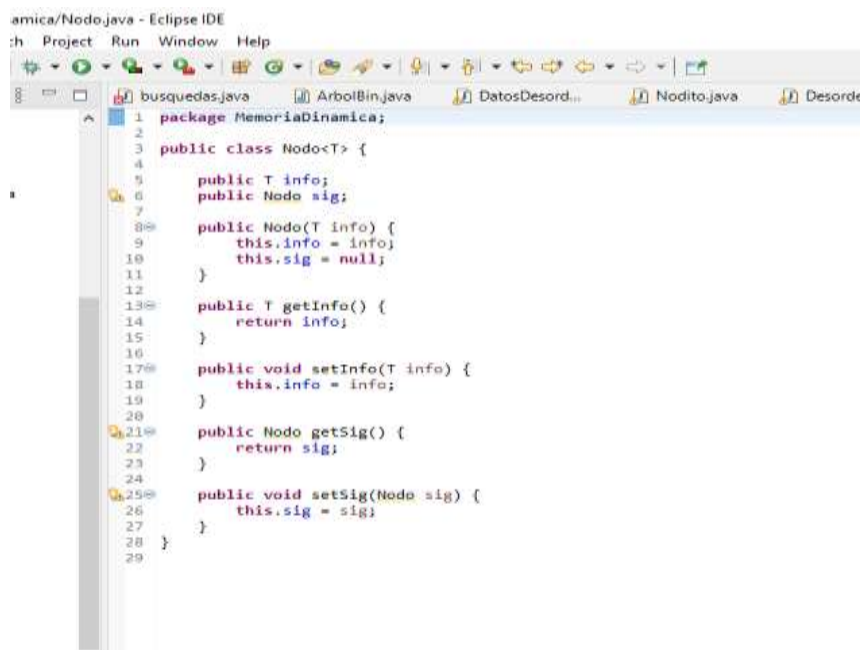
En esta clase tenemos lo que son varios métodos, con los cuales podemos manipular datos con ciertas operaciones en este caso tenemos las de insertar enfrente, final, eliminar, buscar lista y todo esto se realiza con datos desordenados además de que esta clase utiliza una implementación de una interfaz en este caso es `OperaTDA` donde vienen los métodos ya antes mencionados.

NODITO

```
package DobleLiga;
public class Nodito <T>{
    public T info;
    public Nodito izq;
    public Nodito der;
    public Nodito(T dato) {
        this.info=dato;
        this.izq=null;
        this.der=null;
    }
    public T getInfo() {
        return info;
    }
    public void setInfo(T info) {
        this.info = info;
    }
    public Nodito getIzq() {
        return izq;
    }
    public void setIzq(Nodito izq) {
        this.izq = izq;
    }
    public Nodito getDer() {
        return der;
    }
    public void setDer(Nodito der) {
        this.der = der;
    }
}
```

En esta clase están las funciones para manipular las listas enlazadas.

Creamos nuestra clase Nodo la cual será genérica (<T>) la cual contendrá las variables que utilizaremos para gestionar la lista. Creamos una variable de tipo genérico(<T>) la cual llamaremos info, esta se encargara de obtener he insertar información en la lista. Creamos una variable de tipo Nodo la cual llamaremos sig esta se encargará de obtener las direcciones del siguiente nodo en la lista. Creamos nuestro constructor el cual se encargará de iniciar nuestra lista, iniciando por el dato que le mandemos y a siguiente le pondrá null.. Después tendremos los Set y Get necesarios para obtener y añadir nodos a nuestra lista.



```
amica/Nodo.java - Eclipse IDE
:h Project Run Window Help

package MemoriaDinamica;

public class Nodo<T> {

    public T info;
    public Nodo sig;

    public Nodo(T info) {
        this.info = info;
        this.sig = null;
    }

    public T getInfo() {
        return info;
    }

    public void setInfo(T info) {
        this.info = info;
    }

    public Nodo getSig() {
        return sig;
    }

    public void setSig(Nodo sig) {
        this.sig = sig;
    }
}
```

Creamos una interfaz genérica(<T>) la cual contendrá todos los métodos que ocuparemos para manipular la lista enlazada.

ARBOLES

Para esta clase ocuparemos la clase Nodito.

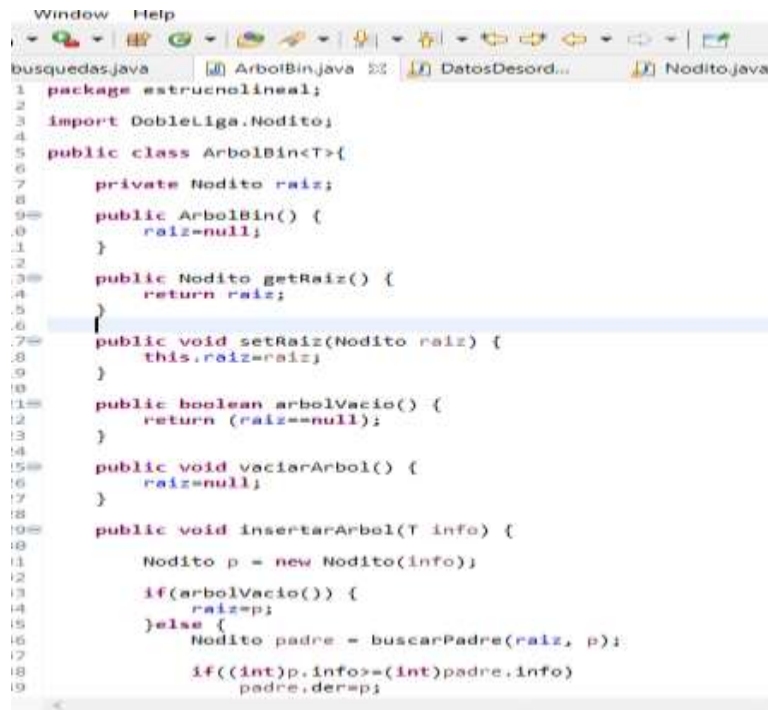
Crearemos una clase genérica (<T>) la cual se encargara de administrar nuestro árbol.

Empezamos por crear nuestro árbol, creamos una variable la cual llamaremos raíz de tipo Nodito, esta variable se encargara de tener la dirección donde comienza nuestro árbol.

Creamos nuestro constructor el cual iniciara nuestra raíz como null.

Creamos el Get y el Set los cuales se encargaran de obtener he insertar los datos de la raíz.

Creamos un método que devolverá una variable booleana, si la raíz es igual con null quiere decir que no tenemos ningún dato en el árbol así que devolverá un true y en caso contrario devolverá un false.



```
Window Help
busquedas.java  ArbolBin.java  DatosDesord...  Nodito.java
1 package estructural;
2
3 import DobleLiga.Nodito;
4
5 public class ArbolBin<T>{
6
7     private Nodito raiz;
8
9     public ArbolBin() {
10         raiz=null;
11     }
12
13     public Nodito getRaiz() {
14         return raiz;
15     }
16
17     public void setRaiz(Nodito raiz) {
18         this.raiz=raiz;
19     }
20
21     public boolean arbolVacio() {
22         return (raiz==null);
23     }
24
25     public void vaciarArbol() {
26         raiz=null;
27     }
28
29     public void insertarArbol(T info) {
30
31         Nodito p = new Nodito(info);
32
33         if(arbolVacio()) {
34             raiz=p;
35         }else {
36             Nodito padre = buscarPadre(raiz, p);
37             if((int)p.info>=(int)padre.info)
38                 padre.der=p;
39         }
40     }
41 }
```

Creamos un método que se encargara de vaciar el arreglo, igualando la Raíz con null así perdiendo todas las direcciones del árbol.

Creamos un método que se encargara de insertar los nuevos datos al árbol, residimos el dato que queremos ingresar en los parámetros, creamos un nuevo nodo y le insertamos el dato, si el árbol esta vacío insertamos el nodo en la raíz, en caso contrario buscaremos el que será el padre del nuevo nodo y lo insertamos debajo de la dirección que regreso buscar padre dependiendo si es mayor o menor.

Creamos un método que se encargara de buscar el padre del nuevo nodo que queremos ingresar al árbol, este método recibirá como parámetros la raíz y el dato que queremos ingresar, buscare la dirección dependiendo si el dato que queremos ingresar es más grande igual o más pequeño que el nodo donde se encuentra actual he ira recorriendo el árbol hasta encontrar

Un null, cuando lo encuentre regresara la última dirección donde estuvo.


```

1         padre.izq=p;
2     }
3 }
4
5 }
6
7 public Nodito buscarPadre(Nodito actual, Nodito p) {
8     Nodito padre = null;
9     while(actual!=null) {
10         padre = actual;
11         if((int)p.info>=(int)padre.info) {
12             actual = padre.der;
13         }else {
14             actual = padre.izq;
15         }
16     }
17     return padre;
18 }
19
20 public String preorden(Nodito r) {
21     if(r!=null) {
22         return r.getInfo()+" - "+preorden(r.getIzq())+" - "+preorden(r.getDer());
23     }
24     else return "";
25 }
26
27 public String inorden(Nodito r) {
28     if(r!=null) {
29         return inorden(r.getIzq())+" - "+r.getInfo()+" - "+inorden(r.getDer());
30     }
31     else return "";
32 }
33
34 public String inorden2(Nodito r) {
35     if(r!=null) {
36         return inorden2(r.getDer())+" - "+r.getInfo()+" - "+inorden2(r.getIzq());
37     }
38     else return "";
39 }

```

Activar W
Ve a Config

Writable Smart Insert 16:5:202

Creamos un método que imprimirá el árbol en pre orden, este método recibirá como parámetro el árbol, recorrerá este árbol de forma recursiva primero imprimiendo la información del nodo donde se encuentra y

Recorrerá el árbol primero por la izquierda, una vez que termine con la izquierda pasara con la derecha y así sucesivamente, hasta llegar al último nodo.

Crearemos un método que imprimirá el árbol en la forma inorden, de forma

Recursiva recorrerá el árbol primero por la izquierda, después imprimirá el

Nodo donde se encuentra y al final recorrerá el árbol por su lado derecho,

Esto se repetirá hasta que llegue al último nodo.

```
ArbolIn.java 22 DatosDesordi... Nodito.java Desordenado... ColaTDA.java E Nodo.java
public String inorden2(Nodito r) {
    if(r != null) {
        return inorden2(r.getDer()) + " " + r.getInfo() + " " + inorden2(r.getIzq());
    }
    else return "";
}

public String posorden(Nodito r) {
    if(r != null) {
        return posorden(r.getIzq()) + " " + posorden(r.getDer()) + " " + r.getInfo();
    }
    else return "";
}

public Nodito buscarDato(Nodito r, T dato) {
    while(r != null) {
        if(r.getInfo() == dato) {
            return r;
        }
        else {
            if((int) dato < (int) r.getInfo()) {
                r = r.getIzq();
            }
            else {
                r = r.getDer();
            }
        }
    }
    return null;
}

public static void graficarArbol(Nodito nodo, int nivel) {
    if (nodo != null) {
        graficarArbol(nodo.der, nivel + 1);
        System.out.println(" ".repeat(nivel) + nodo.getInfo());
        graficarArbol(nodo.izq, nivel + 1);
    }
}
```

Crearemos un método que imprimirá el árbol en la forma inorden pero al revés, de forma recursiva recorrerá el árbol primero por la derecha, después imprimirá el nodo donde se encuentra y al final recorrerá el árbol por su lado izquierdo, esto se repetirá hasta que llegue al último nodo. Creamos un método que recorrerá el árbol de forma recursiva, iniciando por recorrer el árbol en su lado izquierdo, cuando termine parase al lado derecho y al final imprimirá el nodo donde se encuentra.

Crearemos un método que busque cualquier dato que queramos en el

Árbol, para esto recibiremos dos parámetros los cuales serán, el árbol y el dato a buscar, para esto usaremos un while para recorrer el árbol dependiendo si en el nodo donde nos encontramos es más grande que el dato que queremos nos iremos a la derecha del árbol en caso contrario nos iremos a la izquierda, repitiendo este proceso encontraremos el dato que buscamos. Para este método lamentablemente no pudimos imprimir el árbol verticalmente pero si horizontalmente, primero tenemos que verificar que el árbol

No este vacío, una vez verificado empezaremos a recorrer el árbol, primero por la derecha para que salgan arriba los nodos de la derecha una vez llegando al final se empezarán a imprimir para esto daremos los espacios necesarios para que tome la forma del árbol para hacer esto mientras vamos recorriendo el árbol incrementaremos el nivel, una vez llegada hasta la raíz se empezaran a imprimir los de la izquierda cada vez más abajo. Creamos un método que nos dará la altura del árbol, el cual recorrerá todo el árbol he ira guardando cuantas veces recorrió el árbol en las variables altura izquierda y altura derecha, al final ocuparemos la función Math.max para sacar la altura del árbol y le aumentaremos uno para contar la raíz

Como 1.

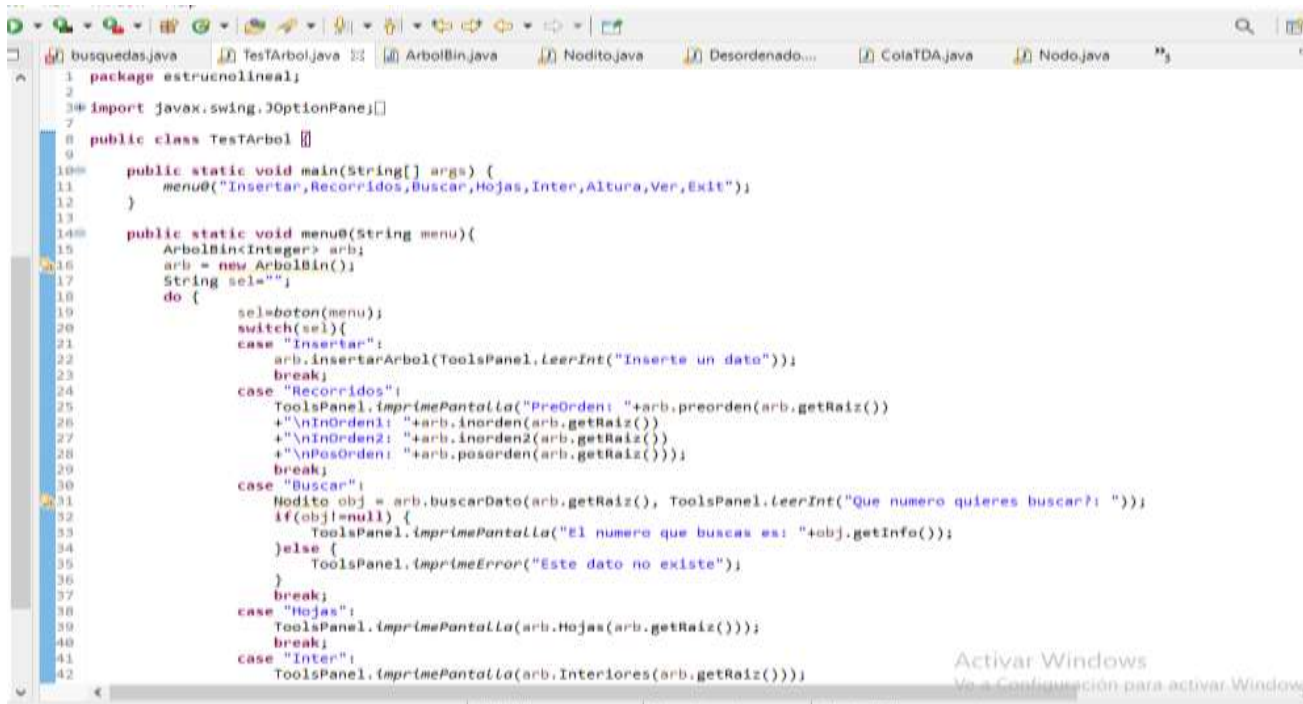
Creamos un método que nos dará las hojas del árbol ósea los nodos de

Los extremos, para esto recorreremos el árbol hasta que derecha he

Izquierda sean nulos así sabremos que son las hojas del árbol.

Para este método compararemos cada nodo si tiene dos nodos hijos eso

Significa que es un nodo interior así que lo agregamos a la cadena.



```
1 package estructural;
2
3 import javax.swing.JOptionPane;
4
5 public class TestArbol {
6
7     public static void main(String[] args) {
8         menu0("Insertar,Recorridos,Buscar,Hojas,Inter,Altura,Ver,Exit");
9     }
10
11     public static void menu0(String menu){
12         ArbolBin<Integer> arb;
13         arb = new ArbolBin();
14         String sel="";
15         do {
16             sel=boton(menu);
17             switch(sel){
18                 case "Insertar":
19                     arb.insertarArbol(ToolsPanel.LeerInt("Inserte un dato"));
20                     break;
21                 case "Recorridos":
22                     ToolsPanel.imprimePantalla("PreOrden: "+arb.preorden(arb.getRaiz())
23                     +"\nInOrden1: "+arb.inorden(arb.getRaiz())
24                     +"\nInOrden2: "+arb.inorden2(arb.getRaiz())
25                     +"\nPosOrden: "+arb.posorden(arb.getRaiz()));
26                     break;
27                 case "Buscar":
28                     Nodito obj = arb.buscarDato(arb.getRaiz(), ToolsPanel.LeerInt("Que numero quieres buscar?: "));
29                     if(obj!=null) {
30                         ToolsPanel.imprimePantalla("El numero que buscas es: "+obj.getInfo());
31                     }else {
32                         ToolsPanel.imprimeError("Este dato no existe");
33                     }
34                     break;
35                 case "Hojas":
36                     ToolsPanel.imprimePantalla(arb.Hojas(arb.getRaiz()));
37                     break;
38                 case "Inter":
39                     ToolsPanel.imprimePantalla(arb.Interiores(arb.getRaiz()));
40             }
41         } while (sel != "Exit");
42     }
43 }
```

Finalmente crearemos un menú para manipular nuestro árbol.

```

26     *"\nInOrden: " + arb.inorden(arb.getRaiz())
27     *"\nInOrden2: " + arb.inorden2(arb.getRaiz())
28     *"\nPosOrden: " + arb.posorden(arb.getRaiz());
29     break;
30     case "Buscar":
31         Nodo obj = arb.buscarDato(arb.getRaiz(), ToolsPanel.leerInt("Que numero quieres buscar?: "));
32         if(obj != null) {
33             ToolsPanel.imprimePantalla("El numero que buscas es: " + obj.getInfo());
34         } else {
35             ToolsPanel.imprimeError("Este dato no existe");
36         }
37         break;
38     case "Hojas":
39         ToolsPanel.imprimePantalla(arb.Hojas(arb.getRaiz()));
40         break;
41     case "Inter":
42         ToolsPanel.imprimePantalla(arb.Interiores(arb.getRaiz()));
43         break;
44     case "Altura":
45         ToolsPanel.imprimePantalla(arb.Altura(arb.getRaiz()) + "");
46         break;
47     case "Ver":
48         arb.graficarArbol(arb.getRaiz(), 0);
49         break;
50     case "Exit":
51         break;
52     }
53     } //switch
54     } while(!sel.equals(ignoreCase("Exit")));
55 }
56
57 public static String boton(String menu) {
58     String valores[] = menu.split(",");
59     int n;
60     n = JOptionPane.showOptionDialog(null, "SELECCIONA DANDO CLICK ", " M E N U ", JOptionPane.NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, valores, valores[0]);
61     return (valores[n]);
62 }
63
64 }

```

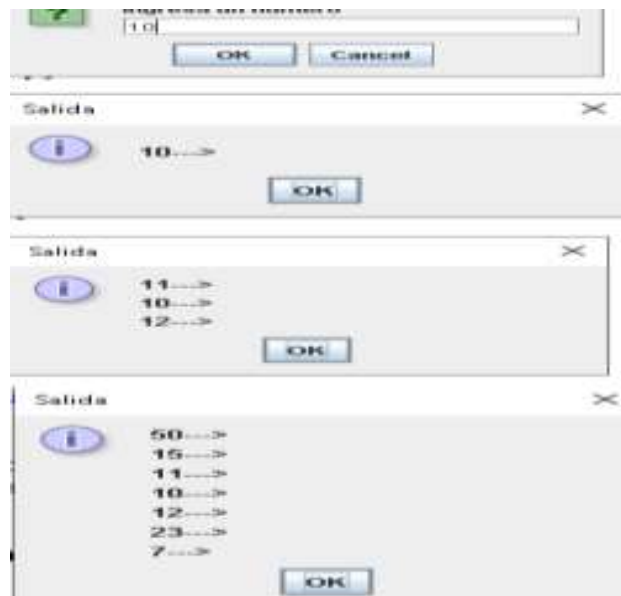
RESULTADOS

Listas enlazadas desordenadas

Menú



Insertar frente, insertar final



Modificar



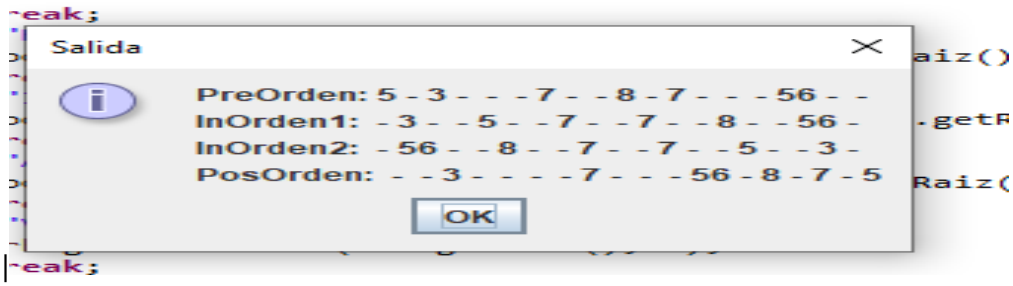
Eliminar



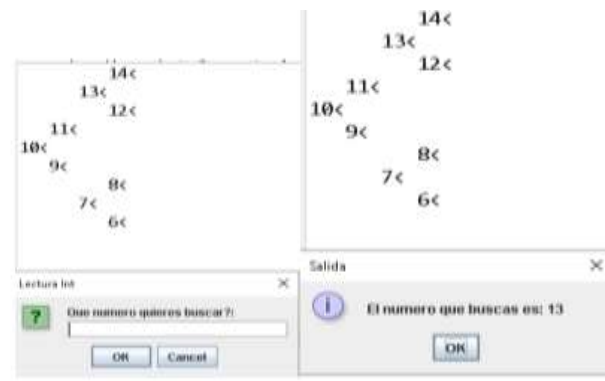
Arboles
Menú



Recorridos



Impresión del arbol y busqueda



CONCLUSION

De acuerdo a la forma de trabajo, y lo visto en este documento académico, podemos concluir que:

Las listas simplemente enlazadas, son posiblemente las estructuras de datos más fáciles, rápidas y sencillas de estudiar, aprender y entender.

La resolución de una función relacionada con listas simplemente enlazadas, es fácil y rápida, en especial porque no se requieren demasiadas líneas de código, por ende la solución es inmediata.

La implementación de pilas y colas mediante listas enlazadas posibilita la representación eficiente de los datos en situaciones donde es necesario indicar el orden de procesamiento de los mismos y no es posible prever la cantidad de elementos a procesar por cuanto este tipo de representación permite crear y destruir variables dinámicamente.

Conocimos, identificamos y aplicamos las estructuras no lineales en la solución de problemas del mundo real. Al igual conocimos un poco acerca de las teorías de esta unidad así como el aprendizaje utilizando un lenguaje de programación podemos implementar las operaciones básicas (insertar, eliminar, buscar) en un árbol binario de búsqueda, así como los recorridos en PreOrden, InOrden y Postorden.

BIBLIOGRAFIAS

Mariaca, J. (2013, 6 diciembre). Proyecto Programación: Listas Enlazadas. Monografias.com.

<https://www.monografias.com/trabajos99/proyecto-programacion-listas-enlazadas/proyecto-programacion-listas-enlazadas>

Tonkseverus. (s. f.). Estructuras lineales y no lineales.

<http://estructuradedatos10111248.blogspot.com/2015/07/estructuras-lineales-y-no-lineales.html>

Estructuras De Datos Básicas":

<http://users.dcc.uchile.cl>,

"Listas Enlazadas Simples, Y Árboles Binarios", 2002:

<http://tutorialms-dos.bligoo.com>

De Madrid, U. C. I. (s. f.). Listas Enlazadas, Pilas y Colas. http://www.it.uc3m.es/java/2012-13/units/pilas-colas/guides/4/guide_es_solution.html