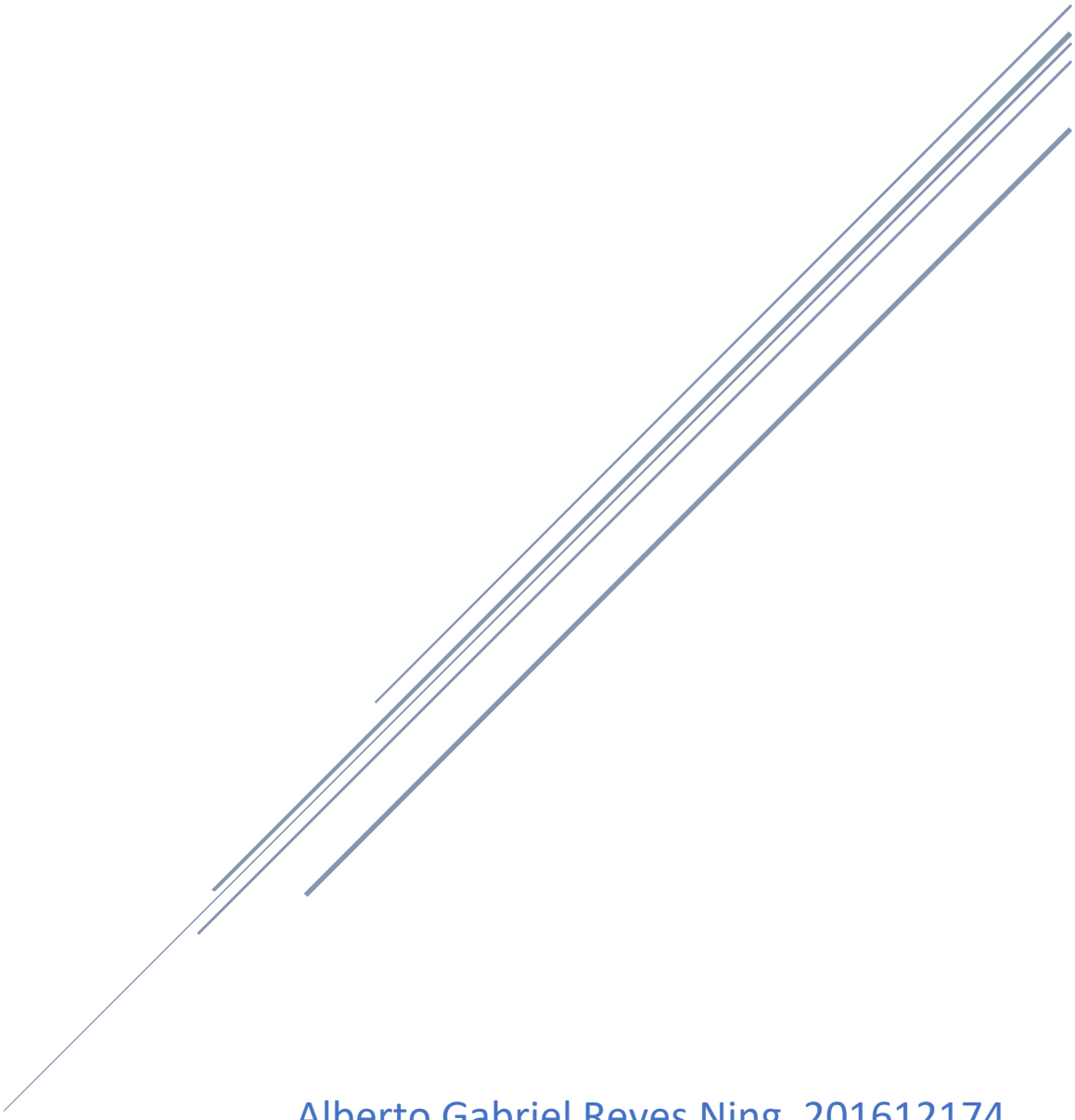
MANUAL TECNICO

EDD PROYECTO FASE 2



Alberto Gabriel Reyes Ning, 201612174
Universidad de San Carlos de Guatemala

Contents

Objetivos del Sistema

Desarrollar una aplicación utilizando las estructuras de datos y sus algoritmos explicados en clase, de tal forma que pueda simular los diferentes procesos que se dan en la empresa. Dicha aplicación deberá ser capaz de representar las estructuras de forma visual, mediante la utilización de bibliotecas soportadas. Desea realizar una aplicación de generar imágenes utilizando diferentes estructuras y sus diferentes recorridos.

Especificación Técnica

Requisitos de Hardware

- Memoria RAM: 4 GBs o Superior
- Procesador: Intel Celeron o Superior

Requisitos de Software

Sistema Operativo

- Windows 8 o Superior

Lenguajes de Programación e IDE

- Java Versión 8 Update 271
- NetBeans 8.2

Librerías Usados

- JsonSimple 1.1
- GraphViz

Lógica del Programa

Clases:

Estructuras Utilizados:

- Matriz Dispersa
- Árbol Binaria de Búsqueda
- Árbol AVL
- Árbol B
- Lista Doblemente Enlazada

Nodos:

- Nodo_MD
- Nodo_ABB
- Nodo_AVL
- NodoB
- RamaB
- Nodo_DE

Interfaz:

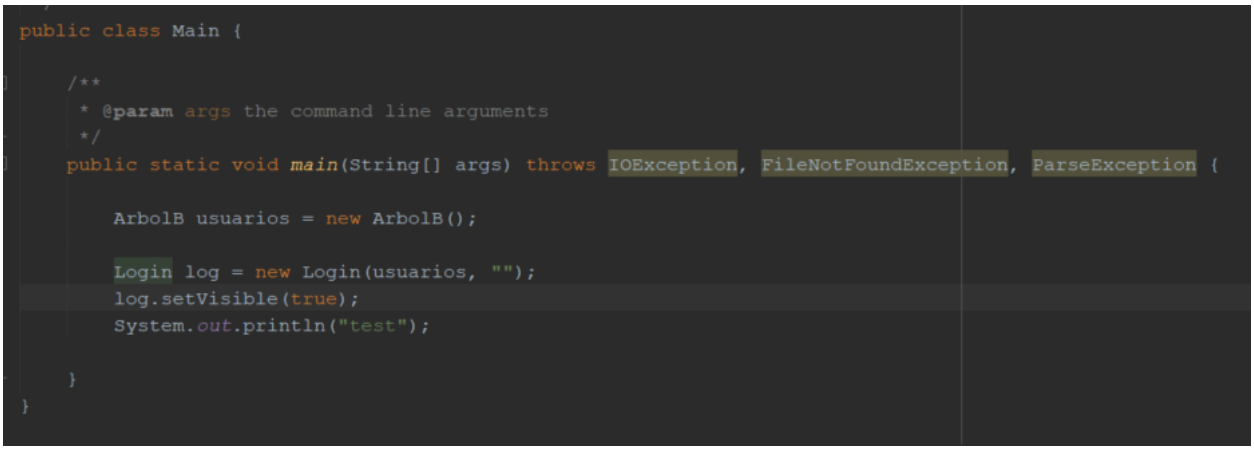
- Login
- Administrador
- Usuario

Main:

- Main
- Carga_Masiva
- GraphViz

Clase Main

Contiene una llamada al interfaz de la aplicación, en este caso, el login. También inicializa la estructura donde guardan los usuarios.



```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException, FileNotFoundException, ParseException {

        ArbolB usuarios = new ArbolB();

        Login log = new Login(usuarios, "");
        log.setVisible(true);
        System.out.println("test");

    }
}
```

Clase Carga Masiva

Esta clase contiene cuatro funciones para cargar diferentes tipos de archivos. También se guardan los archivos creados en carpetas creadas por estas funciones.

```
public static void cargaMasivaUsuarios(File f, ArbolB usuarios) throws FileNotFoundException, IOException, ParseException {
    JSONParser jsonParser = new JSONParser();
    JSONArray jsonArray = (JSONArray) jsonParser.parse(new FileReader(f));
    Iterator<JSONObject> iterator = jsonArray.iterator();
    while (iterator.hasNext()) {
        JSONObject image = iterator.next();
        String id = (String) image.get("dpi");
        String nombreC = (String) image.get("nombre_cliente");
        String pass = (String) image.get("password");
        usuarios.insertar(id, nombreC, pass);
    }
    usuarios.graph("Usuarios");
}
```

```java
public static void cargaMasivaImages(File f, String usuarioID, ArbolB usuarios) throws FileNotFoundException, IOException, ParseException {
    String path = "src\\Salidas\\" + usuarioID + "\\Images";
    String[] top = new String[5];
    int[] top5 = new int[5];
    int count = 0;
    JSONParser jsonParser = new JSONParser();
    JSONArray jsonArray = (JSONArray) jsonParser.parse(new FileReader(f));
    Iterator<JSONObject> iterator = jsonArray.iterator();
    AVL avl = new AVL();
    NodoB usuario = usuarios.find(usuarioID);
    ABB random = usuario.node_Capas;
    while (iterator.hasNext()) {

        JSONObject image = iterator.next();
        long id = (long) image.get("id");
        JSONArray capas = (JSONArray) image.get("capas");

        ABB arbol = new ABB();
        arbol.raiz.data = id;
        avl.raiz = avl.insert(avl.raiz, arbol);
        Long t = (Long) id;
        Matriz_Dispersa MD = new Matriz_Dispersa();
        MD.add_Node(0, 0, t.toString());

        capas.forEach(x -> {
            Long idCapa = (Long) x;
            System.out.println("Node: " + idCapa);
            arbol.insert(idCapa);
            Matriz_Dispersa MD_ = random.search(idCapa);
            MD.add_Matrix(MD_);
        });
        MD.graph2(usuarioID, id);
        avl.graficar(avl.raiz);
    }
    usuario.node_Img = avl;
    System.out.println("pause");
}
```

```java
public static ABB carga_Masiva_Capa(File f, String usuarioID) throws FileNotFoundException, IOException, ParseException {
    JSONParser jsonParser = new JSONParser();
    JSONArray jsonArray = (JSONArray) jsonParser.parse(new FileReader(f));
    Iterator<JSONObject> iterator = jsonArray.iterator();
    Matriz_Dispersa MD_Main = new Matriz_Dispersa();
    MD_Main.add_Node(0, 0, "TEST");
    ABB arbol = new ABB();
    while (iterator.hasNext()) {
        JSONObject capa = iterator.next();
        Long id = (Long) capa.get("id_capa");
        String id_ = id.toString();
        JSONArray pixeles = (JSONArray) capa.get("pixeles");
        Iterator<JSONObject> iterator2 = pixeles.iterator();
        Matriz_Dispersa MD = new Matriz_Dispersa();
        MD.add_Node(0, 0, id_);
        while (iterator2.hasNext()) {
            JSONObject obj = iterator2.next();
            long row = (long) obj.get("fila");
            int row_ = (int) row;
            long col = (long) obj.get("columna");
            int col_ = (int) col;
            String color = (String) obj.get("color");
            MD.add_Node(row_, col_, color);
            MD_Main.add_Node(row_, col_, color);
        }
        System.out.println("NEW MD: " + MD.inicio.data);
        arbol.insert(MD);
        MD.graph(usuarioID, id);
    }
    MD_Main.graph(usuarioID, 123L);
    arbol.graficar(arbol.raiz, "1234567890101", 25L);
    return arbol;
}
```

```
public static void cargaMasivaAlbums(String usuarioID, ArbolB usuarios, File f) throws FileNotFoundException, IOException, ParseException {
    JSONParser jsonParser = new JSONParser();
    JSONArray jsonArray = (JSONArray) jsonParser.parse(new FileReader(f));
    Iterator<JSONObject> iterator = jsonArray.iterator();
    while (iterator.hasNext()) {
        JSONObject capa = iterator.next();
        String nombre_Album = (String) capa.get("nombre_album");
        JSONArray imgs = (JSONArray) capa.get("imgs");
        Lista_DE alb = new Lista_DE();
        alb.add(nombre_Album);
        String path = "src\\Salidas\\" + usuarioID + "\\" + nombre_Album;
        File album = new File(path);
        album.mkdir();
        NodoB usuario = usuarios.find(usuarioID);
        imgs.forEach(t -> {
            Long idImagen = (Long) t;
            ABB x = usuario.node_Img.search(idImagen);
            alb.add_Images(nombre_Album, x);
            String path1 = "src\\Salidas\\" + usuarioID + "\\Images\\" + idImagen + ".png";
            String path2 = "src\\Salidas\\" + usuarioID + "\\" + nombre_Album + "\\" + idImagen + ".png";
            File src = new File(path1);
            File dest = new File(path2);
            try {
                copyFile(path1, path2);
            } catch (IOException ex) {
                Logger.getLogger(Carga_Masiva.class.getName()).log(Level.SEVERE, null, ex);
            }
        });

        usuario.node_Album = alb;
    }
}
```

```
public static void copyFile(String from, String to) throws IOException {
    Path pathIn = (Path) Paths.get(from);
    Path pathOut = (Path) Paths.get(to);
    System.out.println("Path of target file: "
            + pathIn.toString());

    System.out.println("Path of source file: "
            + pathOut.toString());
    try {
        Files.copy(pathIn, pathOut);
    } // Catch block to handle the exceptions
    catch (IOException e) {
    }
}
```

Estructuras:

ABB

Esta estructura contiene un constructor para crear un nuevo ABB, dos funciones insert, una para agregar Matriz Dispersas y otro para agregar datos Long. También tiene una función search para regresar un matriz dispersa por razones de cálculos. Hay tres funciones que recorren las capas en diferentes recorridos para generar las capas recorridas. También tiene dos funciones uno para invocar el graficar y el otro para escribir el parte dot.

```java
public Node_ABB raiz;

public ABB() {
    raiz = new Node_ABB();
}

public void insert(Matriz_Dispersa MD) {
    Long data = Long.parseLong(MD.inicio.data);
    System.out.println("TEST::::" + data);
    Node_ABB temp = this.raiz;
    Node_ABB rand = new Node_ABB(MD);
    Boolean status = false;
    while (status == false) {
        if (temp.data == -1) {
            temp.data = data;
            this.raiz.node_Access = MD;
            status = true;
            System.out.println(data + " has been inserted as the root");
        } else {
            if (data == temp.data) {
                System.out.println("Error: Valor Duplicado");
                break;
            } else if (data > temp.data) {
                if (temp.right == null) {
                    temp.right = rand;
                    status = true;
                    System.out.println(data + " has been inserted on the right");
                } else {
                    temp = temp.right;
                }
            } else if (data < temp.data) {
                if (temp.left == null) {
                    status = true;
                    temp.left = rand;
                    System.out.println(data + " has been inserted on the left");
                } else {
                    temp = temp.left;
                }
            }
        }
```

```java
public void insert(Long data) {
    Node_ABB temp = this.raiz;
    Node_ABB rand = new Node_ABB(data);
    Boolean status = false;
    while (status == false) {
        if (temp.data == -1) {
            temp.data = data;
            status = true;
            System.out.println(data + " has been inserted as the root");
        } else {
            if (data == temp.data) {
                System.out.println("Error: Valor Duplicado");
                break;
            } else if (data > temp.data) {
                if (temp.right == null) {
                    temp.right = rand;
                    status = true;
                    System.out.println(data + " has been inserted on the right");
                } else {
                    temp = temp.right;
                }
            } else if (data < temp.data) {
                if (temp.left == null) {
                    status = true;
                    temp.left = rand;
                    System.out.println(data + " has been inserted on the left");
                } else {
                    temp = temp.left;
                }
            }
        }
    }
}
```

```java
public Matriz_Dispersa search(Long data){
    Node_ABB temp = this.raiz;
    while(temp != null){
        if(Objects.equals(data, temp.data)){
            return temp.node_Access;
        }
        if(data > temp.data){
            temp = temp.right;
        }
        if(data < temp.data){
            temp = temp.left;
        }
    }
    return temp.node_Access;
}

public void displayIO(Node_ABB temp, Matriz_Dispersa MD, int go) { //EnOrden
    if (temp == null) {
        return;
    }
    if(MD.count < go){
        MD.add_Matrix(temp.node_Access);
        MD.count++;
    }
    displayIO(temp.left, MD, go);
    System.out.print(temp.data + " ");
    displayIO(temp.right, MD, go);
}

public void displayPreO(Node_ABB temp, Matriz_Dispersa MD, int go) {
    if (temp == null) {
        return;
    }
    if(MD.count < go){
        MD.add_Matrix(temp.node_Access);
        MD.count++;
    }
    System.out.print(temp.data + " ");

    displayPreO(temp.left, MD, go);
    displayPreO(temp.right, MD, go);
}
```

```java
public void displayPostO(Node_ABB temp, Matriz_Dispersa MD, int go) {

    if (temp == null) {
        return;
    }
    if(MD.count < go){
        MD.add_Matrix(temp.node_Access);
        MD.count++;
    }
    displayPostO(temp.left, MD, go);
    displayPostO(temp.right, MD, go);
    System.out.print(temp.data + " ");

}
//Graph ABB
public void graficar(Node_ABB node, String usuarioID, Long id) {
    try {
        String path = "src\\Salidas\\" + usuarioID + "\\Images\\" + id +".txt";
        FileWriter myWriter = new FileWriter(path);
        myWriter.write("digraph G\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\n");
        myWriter.write(graficadora(node));
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir(path, usuarioID, id);
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```java
public String graficadora(Node_ABB node) {
    String cadena = "";
    if ((node.left == null) && (node.right == null)) {
        cadena = "nodo" + node.data + "[ label =\"" + node.data + "\"]; \n";
    } else {
        cadena = "nodo" + node.data + " [ label =\"" + node.data + "\"];\n";
    }
    if (node.left != null) {//:C0
        cadena = cadena + graficadora(node.left) + "nodo" + node.data + "->nodo" + node.left.data + "\n";
    }
    if (node.right != null) {//:C1
        cadena = cadena + graficadora(node.right) + "nodo" + node.data + "->nodo" + node.right.data + "\n";
    }
    return cadena;
}
```

AVL

La estructura AVL tiene los mismos funciones del ABB con la diferencia que toma en cuenta la altura del Árbol. Cuando el árbol tiene una diferencia de 2 o más en altura, se balancea solo.

```
public class AVL {
    Long id;
    Node_AVL raiz;

    public AVL() {
        raiz = null;
    }

    public int height(Node_AVL x) {
        if (x == null) {
            return 0;
        } else {
            return x.height;
        }
        //Math.max(3, 5);
    }

    public Node_AVL rightRotate(Node_AVL x) {
        Node_AVL temp = x.left;
        Node_AVL temp_ = temp.right;
        temp.right = x;
        x.left = temp_;
        x.height = Math.max(height(x.left), height(x.right)) + 1;
        temp.height = Math.max(height(temp.left), height(temp.right)) + 1;
        return temp;
    }

    public Node_AVL leftRotate(Node_AVL x) {
        Node_AVL temp = x.right;
        Node_AVL temp_ = temp.left;
        temp.left = x;
        x.right = temp_;
        x.height = Math.max(height(x.left), height(x.right)) + 1;
        temp.height = Math.max(height(temp.left), height(temp.right)) + 1;

        return temp;
    }
```

```java
public int getBalance(Node_AVL x) {
    if (x == null) {
        return 0;
    } else {
        return height(x.left) - height(x.right);
    }
}

public Node_AVL insert(Node_AVL x, ABB newNode){
    Long data = newNode.raiz.data;
    if(x == null){
        return (new Node_AVL(newNode));
    }

    if(data < x.data){
        x.left = insert(x.left, newNode);
    } else if (data > x.data){
        x.right = insert(x.right, newNode);
    } else {
        return x;
    }

    x.height = 1 + Math.max(height(x.left), height(x.right));

    int balance = getBalance(x);

    if( balance > 1 && data < x.left.data){
        return rightRotate(x);
    }
    if(balance < -1 && data > x.right.data){
        return leftRotate(x);
    }
    if(balance > 1 && data > x.left.data){
        x.left = leftRotate(x.left);
        return rightRotate(x);
    }
    if(balance < -1 && data < x.right.data){
        x.right = rightRotate(x.right);
        return leftRotate(x);
    }
    return x;
}
```

```java
public Node_AVL insert(Node_AVL x, Long data){
    if(x == null){
        return (new Node_AVL(data));
    }

    if(data < x.data){
        x.left = insert(x.left, data);
    } else if (data > x.data){
        x.right = insert(x.right, data);
    } else {
        return x;
    }

    x.height = 1 + Math.max(height(x.left), height(x.right));

    int balance = getBalance(x);

    if( balance > 1 && data < x.left.data){
        return rightRotate(x);
    }

    if(balance < -1 && data > x.right.data){
        return leftRotate(x);
    }

    if(balance > 1 && data > x.left.data){
        x.left = leftRotate(x.left);
        return rightRotate(x);
    }

    if(balance < -1 && data < x.right.data){
        x.right = rightRotate(x.right);
        return leftRotate(x);
    }

    return x;
}
```

```java
public void preOrder(Node_AVL x){
    if(x != null){
        System.out.print(x.data + " ");
        preOrder(x.left);
        preOrder(x.right);
    }
}

public void postOrder(Node_AVL x){
    if(x != null){
        postOrder(x.left);
        postOrder(x.right);
        System.out.print(x.data + " ");
    }
}

public void enOrden(Node_AVL x){
    if(x != null){
        enOrden(x.left);
        System.out.print(x.data + " ");
        enOrden(x.right);
    }
}

public ABB search(Long data){
    Node_AVL temp = this.raiz;
    while(temp != null){
        if(Objects.equals(data, temp.data)){
            return temp.node_Access;
        }
        if(data > temp.data){
            temp = temp.right;
        }
        if(data < temp.data){
            temp = temp.left;
        }
    }
    return temp.node_Access;
}
```

```java
public void graficar(Node_AVL node) {
    try {
        FileWriter myWriter = new FileWriter("src\\Salidas\\reporte1.txt");
        myWriter.write("digraph G\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\n");
        myWriter.write(graficadora(node));
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir("reporte1");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

public String graficadora(Node_AVL node) {
    String cadena = "";
    if ((node.left == null) && (node.right == null)) {
        cadena = "nodo" + node.data + "[ label =\"" + node.data + "\"]; \n";
    } else {
        cadena = "nodo" + node.data + " [ label =\"" + node.data + "\"];\n";
    }
    if (node.left != null) {//:C0
        cadena = cadena + graficadora(node.left) + "nodo" + node.data + "->nodo" + node.left.data + "\n";
    }
    if (node.right != null) {//:C1
        cadena = cadena + graficadora(node.right) + "nodo" + node.data + "->nodo" + node.right.data + "\n";
    }
    return cadena;
}
```

Árbol B:

Los árboles-B son árboles de búsqueda. Este estructura contiene funciones para dividir y agregar en diferentes ramas del árbol según el cantidad de nodos en cada rama.

```java
public class ArbolB {
    int orden_arbol = 5;
    int h = 0;
    int nodo = 0;
    int r = -1;
    String cadena = "";
    RamaB raiz;

    public ArbolB() {
        this.insertar("1234567890101", "Admin", "EDD1S2022");
    }
```

```java
public void insertar(String id, String nombreC, String pass) {
    NodoB nodo = new NodoB(id, nombreC, pass);
    String path = "src\\Salidas\\" + nodo.id;
    File f1 = new File(path);
    boolean bool = f1.mkdir();
    if (bool) {
        System.out.println("Folder is created successfully");
        String path1 = path + "\\Images";
        path += "\\Capas";

        System.out.println(path);
        f1 = new File(path);
        f1.mkdir();

        f1 = new File(path1);
        f1.mkdir();
    } else {
        System.out.println("Error: Possible Duplicate");
    }

    if (raiz == null) {
        raiz = new RamaB();
        raiz.insertar(nodo);
    } else {
        NodoB obj = insertar_en_rama(nodo, raiz);
        if (obj != null) {
            //si devuelve algo el metodo de insertar en rama quiere decir que creo una nueva rama, y se debe insertar en el arbol
            raiz = new RamaB();
            raiz.insertar(obj);
            raiz.hoja = false;
        }
    }
}
```

```java
private NodoB insertar_en_rama(NodoB nodo, RamaB rama) {
    if (rama.hoja) {
        rama.insertar(nodo);
        if (rama.contador == orden_arbol) {
            //si ya se insertaron todos los elementos posibles se debe dividir la rama
            return dividir(rama);
        } else {
            return null;
        }
    } else {
        NodoB temp = rama.primero;
        do {
            if (nodo.id == temp.id) {
                return null;
            } else if (nodo.id.compareTo(temp.id) < 0) {
                NodoB obj = insertar_en_rama(nodo, temp.izquierda);
                if (obj instanceof NodoB) {
                    rama.insertar((NodoB) obj);
                    if (rama.contador == orden_arbol) {
                        return dividir(rama);
                    }
                }
                return null;
            } else if (temp.siguiente == null) {
                NodoB obj = insertar_en_rama(nodo, temp.derecha);
                if (obj instanceof NodoB) {
                    rama.insertar((NodoB) obj);
                    if (rama.contador == orden_arbol) {
                        return dividir(rama);
                    }
                }
                return null;
            }
            temp = (NodoB) temp.siguiente;
        } while (temp != null);
    }
    return null;
}
```

```java
private NodoB dividir(RamaB rama) {
    String val = "-999";
    String nombre = "";
    String pass = "";
    NodoB temp, Nuevito;
    NodoB aux = rama.primero;
    RamaB rderecha = new RamaB();
    RamaB rizquierda = new RamaB();

    int cont = 0;
    while (aux != null) {
        cont++;
        //implementacion para dividir unicamente ramas de 4 nodos
        if (cont < 3) {
            temp = new NodoB(aux.id, aux.nombre_Cliente, aux.password);
            temp.izquierda = aux.izquierda;
            if (cont == 2) {
                temp.derecha = aux.siguiente.izquierda;
            } else {
                temp.derecha = aux.derecha;
            }
            //si la rama posee ramas deja de ser hoja
            if (temp.derecha != null && temp.izquierda != null) {
                rizquierda.hoja = false;
            }

            rizquierda.insertar(temp);

        } else if (cont == 3) {
            val = aux.id;
            nombre = aux.nombre_Cliente;
            pass = aux.password;
        } else {
            temp = new NodoB(aux.id, aux.nombre_Cliente, aux.password);
            temp.izquierda = aux.izquierda;
            temp.derecha = aux.derecha;
            //si la rama posee ramas deja de ser hoja
            if (temp.derecha != null && temp.izquierda != null) {
                rderecha.hoja = false;
            }
            rderecha.insertar(temp);
        }
        aux = aux.siguiente;
    }
    Nuevito = new NodoB(val, nombre, pass);
```

```java
public void graph(String fileName) {
    try {
        FileWriter myWriter = new FileWriter("src\\Salidas\\" + fileName + ".txt");
        myWriter.write("digraph G\n{\nrankdir=\"TB\"\nnode[shape = record, width = 0.5]\nlabel=\"Carnet: 201612174\"\n");
        myWriter.write(graficadora());
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir(fileName);
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

public String graficadora() {
    String cadena = "";
    nodo = 0;
    h = 0;
    cadena = Show();
    System.out.println("TEST\n" + cadena);
    return cadena;
}

public String Show() {
    cadena = Show(raiz, 0);
    return cadena;
}
```

```
private String Show(RamaB x, int pos) {
    assert (x == null);
    NodoB temp = x.primero;
    if (nodo != 0) {
        cadena += "Nodo" + pos + " -> Nodo" + nodo + "\n";
    }
    cadena += "Nodo" + nodo + " [label=\"";
    for (int i = 0; i < x.contador; i++) {
        cadena += "<r" + i + ">|" + temp.nombre_Cliente + "|";
        //System.out.print(temp.id + " ");
        temp = temp.siguiente;
    }
    cadena += "\"];\n";
    System.out.println(cadena);
    int tmp = nodo;
    temp = x.primero;
    while (temp != null) {
        if (temp.izquierda != null) {
            h++;
            nodo++;
            Show(temp.izquierda, tmp);
            h--;
        }
        if (temp.derecha != null) {
            h++;
            nodo++;
            Show(temp.derecha, tmp);
            h--;
        }
        temp = temp.siguiente;
    }
    return cadena;
}
```

```
public void Search(String id, ABB capas) {
    Search(raiz, 0, id, capas);
}

// Display
private void Search(RamaB x, int pos, String id, ABB capas) {
    assert (x == null);
    NodoB temp = x.primero;
    if (nodo != 0) {
        if (temp.id.compareTo(id) == 0) {
            temp.node_Capas = capas;
        }
    }
    for (int i = 0; i < x.contador; i++) {
        if (temp.id.compareTo(id) == 0) {
            temp.node_Capas = capas;
        }
        temp = temp.siguiente;
    }
    int tmp = nodo;
    temp = x.primero;
    while (temp != null) {
        if (temp.izquierda != null) {
            Search(temp.izquierda, tmp, id, capas);
        }
        if (temp.derecha != null) {
            Search(temp.derecha, tmp, id, capas);
        }
        temp = temp.siguiente;
    }
}
```

```
public NodoB find(String usuarioID) {
    RamaB x = this.raiz;
    NodoB temp = x.primero;

    if (temp.id.compareTo(usuarioID) == 0) {
        return temp;
    }

    while (temp != null) {
        if (temp.id.compareTo(usuarioID) == 0) {
            return temp;
        }
        if (temp.siguiente == null && usuarioID.compareTo(temp.id) > 0) {
            temp = temp.derecha.primero;
        } else if (usuarioID.compareTo(temp.id) < 0) {
            temp = temp.izquierda.primero;
        } else {
            temp = temp.siguiente;
        }
    }
    return null;
}
```

Lista Doblemente Enlazada

Esta estructura contiene cuatro funciones de los cuales son utilizados para agregar álbumes en la lista y agregar imágenes en cada lista y demostrarlo.

```
public Lista_DE() {
    this.inicio = null;
    this.fin = null;
}

public void add(String id) { //Adding albums
    Node_DE newNode = new Node_DE(id);
    if (inicio == null) { //Agrega al inicio
        inicio = newNode;
    } else if (inicio.next == inicio) { //Agregar despues del inicio
        inicio.next = newNode;
        newNode.prev = inicio;
    } else { //Agregar al ultimo del circulo
        Node_DE current = inicio;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
        newNode.prev = current;
    }
}
```

```java
public void add_Images(String album_Name, ABB img) {
    Node_DE temp = this.inicio;
    Node_DE newNode = new Node_DE();
    while (temp != null) {
        if (temp.id == album_Name) {
            if (temp.down == null) {
                temp.down = newNode;
                newNode.node_Access = img;
            } else {
                temp = temp.down;
                while(temp != temp.node_Access = null ×
                    if(temp.node_Access.raiz.data == img.raiz.data){
                        System.out.println("Error: Duplicate image");
                        break;
                    }
                    if(temp.down == null){
                        temp.down = newNode;
                        newNode.node_Access = img;
                        break;
                    }
                    temp = temp.down;
                }
            }
            break;
        }
        temp = temp.next;
    }
}

public void display() {
    Node_DE current = inicio;
    if (inicio == null) {
        System.out.println("Lista esta vacio");
    } else {
        System.out.println("Nodos son ");
        do {
            System.out.println("Placeholder");
            current = current.next;
        } while (current != inicio);
        System.out.println();
    }
}
```

Matriz Dispersa

Esta estructura tiene varias funciones utilizadas para agregar los nodos a la matriz. Primero se toma en cuenta si la posición del nodo está afuera del rango de la matriz corriente. Si esta afuera, se aumenta el tamaño y agrega el nuevo nodo. Primero agrega los apuntes por las columnas y después por las filas. Si es un nodo duplicado, lo rechaza.

```
public class Matriz_Dispersa {

    int col_Max, row_Max, count = 0;
    Node_MD inicio;

    public Matriz_Dispersa() {
        this.col_Max = -1;
        this.row_Max = -1;
        this.inicio = null;
    }

    void add_Head(Node_MD newNode) {
        this.inicio = newNode;
    }

    void add_Header_Row(int x) {
        Node_MD temp = this.inicio;
        if (this.inicio.row == null) {
            while (x > row_Max) {
                ++row_Max;
                Node_MD newNode = new Node_MD(row_Max);
                temp.row = newNode;
                temp = temp.row;
            }
            return;
        }

        while (temp.row != null) {
            temp = temp.row;
        }
        while (x > row_Max) {
            ++row_Max;
            Node_MD newNode = new Node_MD(row_Max);
            temp.row = newNode;
            temp = temp.row;
        }
    }
```

```
void add_Header_Col(int y) {
    Node_MD temp = this.inicio;
    if (this.inicio.col == null) {
        while (y > col_Max) {
            ++col_Max;
            Node_MD newNode = new Node_MD(col_Max);
            temp.col = newNode;
            temp = temp.col;
        }
        return;
    }

    while (temp.col != null) {
        temp = temp.col;
    }
    while (y > col_Max) {
        ++col_Max;
        Node_MD newNode = new Node_MD(col_Max);
        temp.col = newNode;
        temp = temp.col;
    }
}

void add_Node(int x, int y, String data) {
    Node_MD newNode = new Node_MD(x, y, data);

    if (this.inicio == null) {
        add_Head(newNode);
        System.out.println("testing if it is working: " + data);
        return;
    }

    if (newNode.row_No > row_Max) {
        add_Header_Row(newNode.row_No);
    }

    if (newNode.col_No > col_Max) {
        add_Header_Col(newNode.col_No);
    }

    add_Col_Pointers(newNode);
    add_Row_Pointers(newNode);
}
```

```
void add_Col_Pointers(Node_MD newNode) {
    Node_MD temp = this.inicio.col;                 //Col
    //System.out.println("test");
    Node_MD temp_;

    while (temp.id != newNode.col_No) {                         //Correct Col
        temp = temp.col;
        //System.out.println(temp.col.id);
    }

    if (temp.node_Access == null) {
        temp.node_Access = newNode;
    } else if (temp.node_Access.row_No > newNode.row_No) {
        newNode.down = temp.node_Access;
        temp.node_Access.up = newNode;
        temp.node_Access = newNode;
    } else {
        temp_ = temp.node_Access;
        while (temp_ != null) {
            if (temp_.row_No == newNode.row_No) {
                //System.out.println("ERROR: Nodo Duplicado");
                return;
            } else if (temp_.down == null) {
                temp_.down = newNode;
                newNode.up = temp_;
                break;
            } else if (temp_.down.row_No > newNode.row_No) {
                temp_.down.up = newNode;
                newNode.up = temp_;
                newNode.down = temp_.down;
                temp_.down = newNode;
            }
            temp_ = temp_.down;
        }
    }
}
```

```
void add_Row_Pointers(Node_MD newNode) {
    Node_MD temp = this.inicio.row;                 //Col
    Node_MD temp_;

    while (temp.id != newNode.row_No) {                         //Correct Col
        temp = temp.row;
        //System.out.println(temp.row.id);
    }

    if (temp.node_Access == null) {
        temp.node_Access = newNode;
    } else if (temp.node_Access.col_No > newNode.col_No) {
        newNode.right = temp.node_Access;
        temp.node_Access.left = newNode;
        temp.node_Access = newNode;
    } else {
        temp_ = temp.node_Access;
        while (temp_ != null) {
            if (temp_.col_No == newNode.col_No) {
                //System.out.println("ERROR: Nodo Duplicado");
                return;
            } else if (temp_.right == null) {
                temp_.right = newNode;
                newNode.left = temp_;
                break;
            } else if (temp_.right.col_No > newNode.col_No) {
                temp_.right.left = newNode;
                newNode.left = temp_;
                newNode.right = temp_.right;
                temp_.right = newNode;
            }
            temp_ = temp_.right;
        }
    }
}
```

```
void display() {
    Node_MD temp = inicio;
    Node_MD temp_ = temp.col;
    //Print headers
    System.out.print(temp.data + " ");
    while (temp_ != null) {
        System.out.print(temp_.id + " ");
        temp_ = temp_.col;
    }
    temp = inicio.row;                                  //Setting to the 1st row
    System.out.println("");//Setting to the row nodes
    while (temp != null) {
        temp_ = temp.node_Access;
        System.out.print(temp.id + " ");
        while (temp_ != null) {
            System.out.print(temp_.data + " ");
            temp_ = temp_.right;
        }
        System.out.println("");
        temp = temp.row;
    }
}
```

```
void display2() {
    Node_MD temp = inicio;
    Node_MD temp_ = temp.row;
    //Print headers
    System.out.print(temp.data + " ");
    while (temp_ != null) {
        System.out.print(temp_.id + " ");
        temp_ = temp_.row;
    }
    temp = inicio.col;                                  //Setting to the 1st row
    System.out.println("");//Setting to the row nodes
    while (temp != null) {
        temp_ = temp.node_Access;
        System.out.print(temp.id + " ");
        while (temp_ != null) {
            System.out.print(temp_.data + " ");
            temp_ = temp_.down;
        }
        System.out.println("");
        temp = temp.col;
    }
}
```

```
void add_Matrix(Matriz_Dispersa second) {
    Node_MD temp = second.inicio.row;
    while (temp != null) {
        if (temp.node_Access == null) {
            temp = temp.row;
            continue;
        } else {
            Node_MD temp_ = temp.node_Access;
            while (temp_ != null) {
                this.add_Node(temp_.row_No, temp_.col_No, temp_.data);
                temp_ = temp_.right;
            }
        }
        temp = temp.row;
    }
}

public void graph(String usuarioID, Long fileName) {
    try {
        String path = "src\\Salidas\\" + usuarioID + "\\Capas\\" + fileName + ".txt";
        FileWriter myWriter = new FileWriter(path);
        myWriter.write("digraph G\n{\nrankdir=\"TB\"\nnode[shape = rectangle]\nlabel=\"Carnet: 201612174\"\n");
        myWriter.write(graficadora());
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir(path, usuarioID, fileName, 0);
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
public void graph2(String usuarioID, Long fileName) {
    try {
        String path = "src\\Salidas\\" + usuarioID + "\\Images\\" + fileName + ".txt";
        FileWriter myWriter = new FileWriter(path);
        myWriter.write("digraph G\n{\nrankdir=\"TB\"\nnode[shape = rectangle]\nlabel=\"Carnet: 201612174\"\n");
        myWriter.write(graficadora());
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir(path, usuarioID, fileName);
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
public String graficadora() {
    String cadena;
    String rowInfo = "MD;";
    cadena = "MD -> ";
    for (int i = 0; i <= row_Max; i++) {
        if (i == row_Max) {
            cadena += "a" + (i) + "\n";
        } else {
            cadena += "a" + (i) + " -> ";
        }
    }
    cadena += "MD -> ";
    for (int i = 0; i <= col_Max; i++) {
        if (i == col_Max) {
            cadena += "b" + (i) + "\n";
        } else {
            cadena += "b" + (i) + " -> ";
        }
        rowInfo += "b" + (i) + ";";
    }
    cadena += "{rank=same;" + rowInfo + "}\n";
    Node_MD temp = inicio.row;
    while (temp != null) {
        rowInfo = "";
        rowInfo += "a" + temp.id + ";";
        Node_MD temp_ = temp.node_Access;
        if (temp.node_Access == null) {
            temp = temp.row;
            continue;
        } else {
            cadena += "x" + temp_.row_No + "y" + temp_.col_No + "[label = \"\", style = filled, fillcolor = \"" + temp_.data + "\"]" + "\n";
            cadena += "a" + temp_.row_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> a" + temp_.row_No + "[constraint=false]\n";
            rowInfo += "x" + temp_.row_No + "y" + temp_.col_No + ";"; //data -> color
            if (temp_.up == null) {
                System.out.println(temp_.data);
                cadena += "b" + temp_.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> b" + temp_.col_No + "\n";
            } else {
                cadena += "x" + temp_.up.row_No + "y" + temp_.up.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> x" + temp_.up.row_No + "y" + temp_.up.col_No + "
            }
            temp_ = temp_.right;
        }
```

```
    } else {
        cadena += "x" + temp_.row_No + "y" + temp_.col_No + "[label = \"\", style = filled, fillcolor = \"" + temp_.data + "\"]" + "\n";
        cadena += "a" + temp_.row_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> a" + temp_.row_No + "[constraint=false]\n";
        rowInfo += "x" + temp_.row_No + "y" + temp_.col_No + ";"; //data -> color
        if (temp_.up == null) {
            System.out.println(temp_.data);
            cadena += "b" + temp_.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> b" + temp_.col_No + "\n";
        } else {
            cadena += "x" + temp_.up.row_No + "y" + temp_.up.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> x" + temp_.up.row_No + "y" + temp_.up.col_No +
        }
        temp_ = temp_.right;
    }
    while (temp_ != null) {
        cadena += "x" + temp_.row_No + "y" + temp_.col_No + "[label = \"\", style = filled, fillcolor = \"" + temp_.data + "\"]" + "\n";
        cadena += "x" + temp_.left.row_No + "y" + temp_.left.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> x" + temp_.left.row_No + "y" + temp_.left.col_
        rowInfo += "x" + temp_.row_No + "y" + temp_.col_No + ";";
        if (temp_.up == null) {
            cadena += "b" + temp_.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> b" + temp_.col_No + "\n";
        } else {
            cadena += "x" + temp_.up.row_No + "y" + temp_.up.col_No + " -> x" + temp_.row_No + "y" + temp_.col_No + " -> x" + temp_.up.row_No + "y" + temp_.up.col_No +
        }
        temp_ = temp_.right;
    }
    temp = temp.row;
    cadena += "\n{rank=same;" + rowInfo + "}\n";
}
return cadena;
```