Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Estructuras de Datos

MANUAL TÉCNICO

Henry Ronely Mendoza Aguilar
Carné: 202004810
PROYECTO FASE 2

Objetivos y Alcances del Sistema

Objetivo Principal

- Brindar la solución optima para la generación de imágenes y manejo de información de usuarios

Objetivos Específicos

- Diseñar una plataforma agradable e intuitiva por medio del lenguaje de programación Java.
- Aprovechar de mejor maneral los recursos brindados por el entorno de trabajo NetBeans
- Programar un sistema funcional para cualquier usuario y hardware en el que se utilice.

Requisitos del Hardware

- 512 MB de RAM
- 20MB de disco duro para almacenamiento del software (ya que puede aumentar con la generación de imágenes).
- Procesador Pentium 2 a 266Mhz

Requisitos del Software

- Java
- Graphviz
- Windows 10 (64-bit), Linux o Mac OS

Librerías Empleadas

Graphviz:

Graphviz (abreviatura de Graph Visualization Software) es un paquete de herramientas de código abierto iniciado por AT&T Labs Research para dibujar gráficos especificados en scripts de lenguaje DOT que tienen la extensión de nombre de archivo "gv". También proporciona bibliotecas para que las aplicaciones de software utilicen las herramientas. Graphviz es un software gratuito con licencia de Eclipse Public License.

Formato JSON:

JSON (JavaScript Object Notation) es un formato de archivo estándar abierto y un formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos de datos que consisten en pares atributo-valor y arreglos (u otros valores serializables). Es un formato de datos común con diversos usos en el intercambio electrónico de datos, incluido el de aplicaciones web con servidores.

JSON es un formato de datos independiente del idioma. Se derivó de JavaScript, pero muchos lenguajes de programación modernos incluyen código para generar y analizar datos en formato JSON. Los nombres de archivo JSON usan la extensión. json.

Estructuras de Datos

Matriz Dispersa: Son matrices en las cuales la gran mayoría de las entradas son cero. En inglés se les conoce como “sparse matrices”, en algunos países de habla hispana también se les conoce como “matrices ralas”. Con las matrices dispersas se pueden trabajar fácilmente matrices de millones de renglones por millones de columnas. A veces se requiere solo visualizar las entradas distintas de cero.

Árbol binario de búsqueda: Un árbol binario de búsqueda también llamado BST (acrónimo del inglés Binary Search Tree) es un tipo particular de árbol binario que presenta una estructura de datos en forma de árbol usada en informática. es un árbol binario que cumple que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores. Para estas definiciones se considera que hay una relación de orden establecida entre los elementos de los nodos. Que cierta relación esté definida, o no, depende de cada lenguaje de programación. De aquí se deduce que puede haber distintos árboles binarios de búsqueda para un mismo conjunto de elementos.

Árbol AVL: Un árbol AVL es un tipo especial de árbol binario ideado por los matemáticos soviéticos Adelson-Velskii y Landis. Fue el primer árbol de búsqueda binario auto-balanceable que se ideó. Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en

más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

Árbol B: En las ciencias de la computación, los árboles-B o B-árboles son estructuras de datos de árbol que se encuentran comúnmente en las implementaciones de bases de datos y sistemas de archivos. Al igual que los árboles binarios de búsqueda, son árboles balanceados de búsqueda, pero cada nodo puede poseer más de dos hijos.1 Los árboles B mantienen los datos ordenados y las inserciones y eliminaciones se realizan en tiempo logarítmico amortizado.

Lista enlazada: En ciencias de la computación, una lista enlazada es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento. Una lista enlazada es un tipo de dato autor referenciado porque contienen un puntero o enlace (en inglés link, del mismo significado) a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

Descripción de Métodos

Para el desarrollo de la aplicación se emplearon los métodos de lectura de archivos de texto plano que posee Java las cuales son las pertenecientes al paquete IO. Por medio de un JFileChooser se escogen los archivos para una interfaz gráfica.

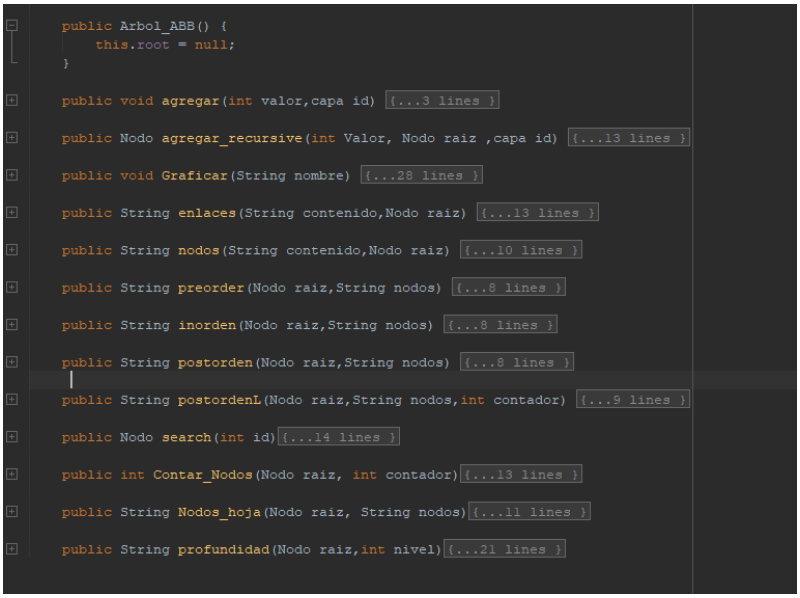
```
FileReader fr = null;
try {
    String file="";
    File archivo=null;
    JFileChooser fc = new JFileChooser();
    int op = fc.showOpenDialog(this);
    if (op == JFileChooser.APPROVE_OPTION) {
        archivo = fc.getSelectedFile();
    }
    fr = new FileReader(archivo);
    BufferedReader br = new BufferedReader(fr);
    String linea;
    while ((linea = br.readLine()) != null) {
        file += "\n"+linea;
    }
```

Para la lectura de los archivos json se utilizó el JsonParser, JsonArray y JsonObject con las claves según el archivo que se ha cargado.

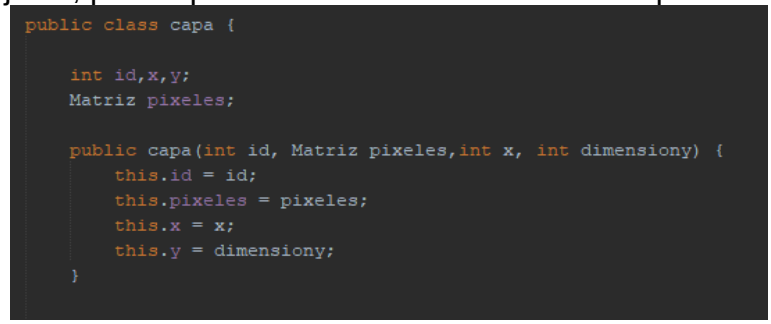
```
JsonParser parser = new JsonParser();
JsonArray capas = parser.parse(file).getAsJsonArray();

for(int i=0;i<capas.size();i++){
    JsonObject cap = capas.get(i).getAsJsonObject();
    int id_capa = cap.get("id_capa").getAsInt();
    JsonArray pixel = cap.get("pixeles").getAsJsonArray();
    Matriz pixeles = new Matriz();
    int fila,columna;
    String color;
    int x = 0, y=0;
    for (int j = 0; j < pixel.size(); j++) {
        JsonObject pxl = pixel.get(j).getAsJsonObject();
        fila = pxl.get("fila").getAsInt()+1;
        columna = pxl.get("columna").getAsInt()+1;
        color = pxl.get("color").getAsString();
```

Matriz Dispersa: Se implementó una matriz dispersa para el almacenamiento de las coordenadas y colores que posee el archivo de capas.

Árbol ABB: Se implementó un árbol binario de búsqueda para almacenar las capas cargadas al sistema



Dentro de las hojas del árbol ABB se guardaron datos empleando la programación orientada a objetos, por lo que se creo una clase llamada capa.



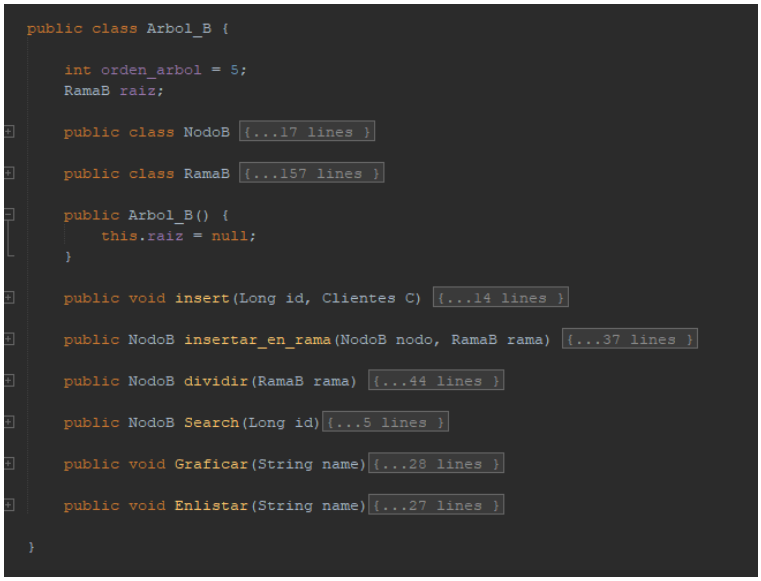
Árbol AVL: Para el almacenamiento de las imágenes creadas en el sistema se implementó un árbol AVL.

```
public class Arbol_AVL {

    Nodo root = null;

    public class Nodo {...15 lines }

    void add(int value, imagen img) {...3 lines }

    Nodo add(int value, Nodo tmp, imagen img) {...30 lines }

    int altura(Nodo tmp) {...4 lines }

    int maxi(int val1, int val2) {...3 lines }

    Nodo srl(Nodo t1) {...9 lines }

    Nodo srr(Nodo t1) {...9 lines }

    Nodo drl(Nodo tmp) {...4 lines }

    Nodo drr(Nodo tmp) {...4 lines }

    public void Graficar(String nombre) {...28 lines }

    public String enlaces(String contenido,Nodo raiz) {...13 lines }

    public String nodos(String contenido,Nodo raiz) {...10 lines }

    public Nodo search(int id){...19 lines }

    public int Contar_Nodos(Nodo raiz, int contador){...13 lines }

    public void eliminar(int id){...3 lines }

    private Nodo eliminarAVL(Nodo actual, int id){...57 lines }

    private Nodo Maximo(Nodo node) {...9 lines }

    private int getFactorEquilibrio(Nodo nodoActual) {...7 lines }
}
```

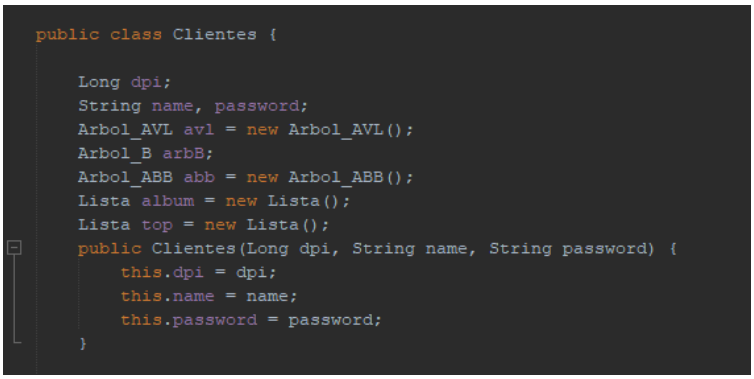
De igual forma se utilizó la programación orientada a objetos para guardar información en las hojas del árbol AVL en este caso se creó una clase llamada imagen.

```
public class imagen {

    int id;
    String ruta;
    Arbol_ABB arb;

    public imagen(int id, String ruta, Arbol_ABB arb) {
        this.id = id;
        this.ruta = ruta;
        this.arb = arb;
    }
```

Árbol B: Se realizó la implementación de un árbol B para la gestión de información de los clientes a los que utilizando la programación orientada a objetos se creo la clase cliente para almacenar en el Árbol B.

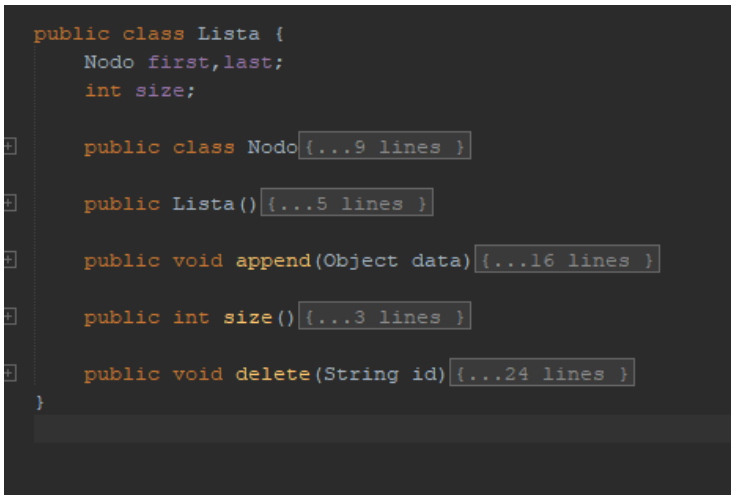


```
public class Arbol_B {

    int orden_arbol = 5;
    RamaB raiz;

    public class NodoB {...17 lines }

    public class RamaB {...157 lines }

    public Arbol_B() {
        this.raiz = null;
    }

    public void insert(Long id, Clientes C) {...14 lines }

    public NodoB insertar_en_rama(NodoB nodo, RamaB rama) {...37 lines }

    public NodoB dividir(RamaB rama) {...44 lines }

    public NodoB Search(Long id){...5 lines }

    public void Graficar(String name){...28 lines }

    public void Enlistar(String name){...27 lines }

}
```



```
public class Clientes {

    Long dpi;
    String name, password;
    Arbol_AVL avl = new Arbol_AVL();
    Arbol_B arbB;
    Arbol_ABB abb = new Arbol_ABB();
    Lista album = new Lista();
    Lista top = new Lista();
    public Clientes(Long dpi, String name, String password) {
        this.dpi = dpi;
        this.name = name;
        this.password = password;
    }
```

Lista enlazada: Se implemento una lista enlazada para el uso en diversos métodos, en este caso se empleó para los reportes, gestión de información en la interfaz y la carga de álbumes los cuales era una implementación de “Lista de Listas”



```
public class Lista {
    Nodo first,last;
    int size;

    public class Nodo{...9 lines }

    public Lista(){...5 lines }

    public void append(Object data){...16 lines }

    public int size(){...3 lines }

    public void delete(String id){...24 lines }
}
```

Implementación de Graphviz: Toda imagen y reporte generado dentro de la aplicación fue mediante el uso de Graphviz, el cual, mediante la escritura de archivo de texto plano, se creo uno en formato DOT y utilizando Proccess Builder para la ejecución de comandos en consola.

```
public void imagen_png(int dimx,int dimy,String name){
    FileWriter capa = null;
    PrintWriter pw = null;
    try{
        capa = new FileWriter(name+"HD.dot");
        pw = new PrintWriter(capa);

        pw.println("digraph G {");
        pw.println("a0 [label=< \n <TABLE cellspacing=\"0\" cellpadding=\"10\">");

        String contenido="";
        NodoCabeceraColumna r = this.columnas.primero;
        for (int i = 0; i < dimy+1; i++) {
            if (r!=null) {
                if (r.x == i) {
                    NodoOrtogonal c = r.columna.primero;
                    contenido+="<TR>";
                    for (int j = 0; j < dimx+1; j++) {
                        if (c != null) {
                            if (c.y == j) {
                                contenido+="<TD bgcolor=\""+c.dato.toString()+"\"></TD>\n";
                                c = c.abajo;
                            } else contenido+="<TD bgcolor=\"#ffffff\"></TD>\n";
                        } else contenido+="<TD bgcolor=\"#ffffff\"></TD>\n";
                    }
                    r = r.siguiente;
                } else {
                    contenido+="<TR>";
                    for (int j = 0; j < dimx+1; j++) {
                        contenido+="<TD bgcolor=\"#ffffff\"></TD>\n";
                    }
                }
            } else {
                contenido+="<TR>";
                for (int j = 0; j < dimx+1; j++) {
                    contenido+="<TD bgcolor=\"#ffffff\"></TD>\n";
                }
            }
            contenido+="</TR>";
        }
        contenido+="</TABLE>>][color=white]";
        pw.println(contenido);
        pw.println("}");

    }catch(Exception e){
        e.printStackTrace();
    }finally{
    try{
        if(null != capa){
            capa.close();
            ProcessBuilder buil = new ProcessBuilder("dot","-Tpng","-o",name+"HD.png",name+"HD.dot");
            buil.redirectErrorStream(true);
            buil.start();
        }
    }catch(Exception e2){
    e2.printStackTrace();
    }
    }
```