Universidad de San Carlos de Guatemala
Escuela de Ciencias y Sistemas
Estructuras De Datos

Manual Técnico
Proyecto 1 Fase 1
UDrawing Paper

Arnoldo Luis Antonio González Camey
Carné: 201701548

INTRODUCCIÓN

Este manual va dirigido a todo público que cuente con conocimientos técnicos sobre la programación y que quiera ver la interacción entre clases, objetos y el uso de árboles para el manejo de la información y los datos, tales como Árboles binarios de búsqueda, Árbol AVL, Árbol B además de Matrices Dispersas y listas simples por medio de una simulación de una empresa denominada "Drawing Paper" la cual requiere registrar clientes y que estos clientes puedan generar imágenes especiales construidas por capas. Para poder hacer uso de la aplicación el cliente debe registrarse.
El reporte grafico se realiza a través del Graphviz, para ello se hizo uso del IDE NetBeans 11.0. Para la lectura de los archivos json se utiliza la librería gson y para la generación de las imágenes se hace uso de la librería aspose-html.

La principal funcionalidad de la aplicación consiste en un generador de imágenes por capas, la aplicación cuenta con un conjunto de capas cargadas previamente y almacenadas en memoria para ser utilizadas; estas capas se utilizarán para generar imágenes hechas con pixeles, cada capa contendrá la información de los distintos pixeles y al colocar una capa sobre otra estas irán formando una imagen más completa.

REQUERIMIENTOS MÍNIMOS

Para poder operar el programa se necesita: Una computadora con un procesador de 32 o 64 bits, sistema operativo Windows 7 en adelante, Linux o cualquier otro sistema que permita la instalación de JAVA, tener instalado JAVA en la versión más reciente, tener la capacidad de acceder a los comandos de Windows o Linux, contar con los archivos en formato json para la utilización de este y las gráficas de las estructuras son generadas a través de Graphviz y las imágenes a través de HTML.

OBJETIVOS

General

- Aplicar los conocimientos del curso de Estructuras de Datos en el desarrollo de una aplicación que permita manipular la información de forma óptima.

Específicos

- Aplicar los conocimientos adquiridos sobre estructuras de datos lineales y no lineales como matrices y árboles
- Implementar una aplicación de escritorio utilizando el lenguaje de programación Java.
- Familiarizarse con la lectura y escritura de archivos de JSON.
- Utilizar la herramienta Graphviz para graficar estructuras de datos no lineales.
- Definir e implementar algoritmos de búsqueda, recorrido y eliminación en estructuras de datos.

Manual Técnico

- Método insertar Árbol Binario

Este método es utilizado para insertar un nodo en el árbol binario de búsqueda para ello se piden de parámetros el valor del nodo y la matriz dispersa de pixeles; se valida si la raiz es nula, si lo es se crea un nuevo nodo que sea igual a la raiz y se aumenta la cantidad, si es nula se hace la instancia del método insertar de la clase NodoBinario para que inserte en el nodo dentro del árbol y al final se retorna la raiz.

```
public NodoBinario insertar(Comparable val, MatrizDispersa pixeles) {
    if (raiz == null) {
        raiz = new NodoBinario(val, pixeles);
        cantidad++;
    } else {
        raiz.insertar(val, pixeles);
    }
    return raiz;
}
```

- Método insertar Nodo Binario

Este método es utilizado para insertar un nodo en el árbol binario de búsqueda para ello se piden de parámetros el valor del nodo y la matriz dispersa de pixeles; se compara si el valor del nodo actual es menor a cero si lo es y el el nodo izquierdo es nulo insertar el nodo en esa posición, si no es nulo, recursivamente avanza al siguiente nodo izquierdo.

Si el valor del nodo actual es mayor a cero se debe insertar a la derecha. Si el nodo derecho es nulo se inserta ahí el valor, si no hace una llamada recursiva al método con el nodo derecho. Si no sucede nada de lo anterior signfica que son igual y no se inserta.

```
public NodoBinario insertar(Comparable val, MatrizDispersa pixeles) {
    //Si el valor es menor que el nodo actual, entonces insertar a la izquierda de este.
    if (val.compareTo(getValor()) < 0) { //Si la izquierda del nodo actual null insertar.
        if (getIzquierdo() == null) {
            NodoBinario nuevo = new NodoBinario(val, pixeles);
            setIzquierdo(nuevo);
            return nuevo;
        } //Si no mover a la izq para buscar donde
        else {
            getIzquierdo().insertar(val, pixeles);
        }
    } //Si el valor  es mayor que el nodo actual, entonces insertat a la derecha de este de este.
    else if (val.compareTo(getValor()) > 0) { //Si la derecha del nodo actual es nula insertar
        if (getDerecho() == null) {
            NodoBinario nuevo = new NodoBinario(val, pixeles);
            setDerecho(nuevo);
            return nuevo;
        } //Si no mover a la derecha
        else {
            getDerecho().insertar(val, pixeles);
        }
        //Si no es mayor ni menor, significa que es igual
    } else {
        System.err.println("No se permiten los valores duplicados: \""
                + String.valueOf(val) + "\".");
    }
    return null;
}
```

- Método insertar Árbol AVL

Esté método recursivo es utilizado para insertar un nodo en el árbol AVL, se envian los parametros necesarios. Se valida si la raiz es nula, si lo es, se asigna el valor del nuevo nodo a la raiz y se le añaden las capas, si no lo es se realiza la comparación del nodo con el valor nuevo, si es menor a cero se hace la instancia recursiva de este método con la raiz a la izquierda, luego se realiza la validación del factor de equilibrio si se necesita rotacióna la izquierda o derecha. Luego se compara si el valor de es mayor a cero, se realiza la llamada recursiva al metodo con el nodo derecho y se realiza la validación del factor de equilibrio y la posible rotación ya sea a la derecha o a la izquierda.

Al finalizar se modifica el valor de la altura, la cual es la altura del hijo mas grande.

```
private NodoAVL insertar(Comparable valor, NodoAVL raiz, JsonArray capas, int numCapas) {
    if (raiz == null) {
        this.cantidadNodo++;
        raiz = new NodoAVL(valor, numCapas);
        for (Object objt2 : capas) {
            raiz.capas.insertarCapa(Integer.parseInt(objt2.toString()));
        }
        //Si el nuevo valor es menor que el nodo actual
    } else if (valor.compareTo(raiz.getValor()) < 0) { //Subarbol izquierdo
        raiz.setIzquierdo(insertar(valor, raiz.getIzquierdo(), capas, numCapas));
        //Si el factor de equilibrio == -2 esta desbalanceo y se hace la rotación
        if (altura(raiz.getDerecho()) - altura(raiz.getIzquierdo()) == -2) {
            if (valor.compareTo(raiz.getIzquierdo().getValor()) < 0) {//Si el valor es menor
                // realizar una rotación simple por la izquierda
                raiz = IzquierdaIzquierda(raiz);
            } else { //si se inserta a la derecha es una rotacion doble por la izquierda
                raiz = IzquierdaDerecha(raiz);
            }
        }
    } else if (valor.compareTo(raiz.getValor()) > 0) { //Subarbol derecho
        raiz.setDerecho(insertar(valor, raiz.getDerecho(), capas, numCapas));
        //Si el factor de equilibrio == -2 esta desbalanceo y se hace la rotación
        if (altura(raiz.getDerecho()) - altura(raiz.getIzquierdo()) == 2) {
            if (valor.compareTo(raiz.getDerecho().getValor()) > 0) {//Si el valor es menor e
                // realizar una rotación simple por la derecha
                raiz = DerechaDerecha(raiz);
            } else {//si se inserta a la derecha es una rotacion doble por la derecha
                raiz = DerechaIzquierda(raiz);
            }
        }
    } else {
        System.err.println("No se permiten los valores duplicados: \""
                + String.valueOf(valor) + "\".");
    }
    //la altura, será la altura del hijo que tiene la altura más grande
    raiz.altura = mayor(altura(raiz.getIzquierdo()), altura(raiz.getDerecho())) + 1;
    return raiz;
}
```

- Método insertar Nodo AVL

Este método es utilizado para insertar un nodo en el árbol avl para ello se piden de parámetros correspondientes; se compara si el valor del nodo actual es menor a cero si lo es y el el nodo izquierdo es nulo insertar el nodo en esa posición, si no es nulo, recursivamente avanza al siguiente nodo izquierdo.

Si el valor del nodo actual es mayor a cero se debe insertar a la derecha. Si el nodo derecho es nulo se inserta ahí el valor, si no hace una llamada recursiva al método con el nodo derecho. Si no sucede nada de lo anterior signfica que son igual y no se inserta.

```
public void insertar(Comparable val, int numCapas) {
    //Si el valor a insertar es menor que el nodo actual insertar a la izquierda
    if (val.compareTo(getValor()) < 0) {
        if (getIzquierdo() == null) {
            setIzquierdo(new NodoAVL(val, numCapas));
        } else { //llamada recursiva a la izquierda
            getIzquierdo().insertar(val, numCapas);
        }
    } //Si el valor a insertar es mayor que el nodo actual insertar a la derecha
    else if (val.compareTo(getValor()) > 0) {
        if (getDerecho() == null) {
            setDerecho(new NodoAVL(val, numCapas));
        } else {//llamada recursiva a la derecha
            getDerecho().insertar(val, numCapas);
        }
    } else {
        System.err.println("No se permiten los valores duplicados: \""
                + String.valueOf(val) + "\".");
    }
}
```

- Método insertar Árbol B

Esté método recursivo es utilizado para insertar un nodo en el árbol B, se envian el parametro necesario. Se valida si la raiz es nula, si lo es, se asigna el valor del nuevo nodo a la raiz, si no lo es se verifica el valor del numero de laves y el grado, si es igual se agrega el nodo y se inserta en el arbol. Si esto no sucede se hace la instancia del metodo insertar en un nodo no lleno.

```
public void insertar(long key) {
    if (raiz == null) {
        raiz = new NodoB(grado, true);
        raiz.ClavesNodo[0] = key;
        raiz.numClavesNodo = 1;
    } else {
        if (raiz.numClavesNodo == 2 * grado - 1) {// Cuando el nodo raiz está lleno, el árbol crecerá más alto
            NodoB nuevo = new NodoB(grado, false);
            nuevo.hijos[0] = raiz; // La raiz se convierte en hijo de la nueva raiz
            nuevo.dividirHijo(0, raiz);
            int i = 0;
            if (nuevo.ClavesNodo[0] < key) {
                i++;
            }
            nuevo.hijos[i].insertarNoLleno(key);

            raiz = nuevo;
        } else {
            raiz.insertarNoLleno(key);
        }
    }
}
```

- Método eliminar Árbol B

Este método es usado para eliminar un elemento del arbol b para ello se encuentra el indice donde se encuentra el nodo, luego si es hoja y se hace la instancia del metodo eliminar de hoja y si no del metodo eliminar no hoja, si no se cumple la primera validación se ingresa a la otra parte del metodo en el cual se busca el nodo en alguno de los hijos de forma recursiva.

```
public void eliminar(long key) {
    int indice = EncontrarClave(key);
    if (indice < numClavesNodo && ClavesNodo[indice] == key) { // Encontrar la llave
        if (isHoja()) { // la clave está en el nodo hoja
            eliminarDeHoja(indice);
        } else {// la clave no está en el nodo hoja
            elminarNoHoja(indice);
        }
    } else {
        if (isHoja()) { // Si el nodo es un nodo hoja, entonces el nodo no está en el árbol B
            System.out.printf("El dpi %d no existe\n", key);
            return;
        }
        boolean bandera = indice == numClavesNodo;//La clave existe en el subárbol hijo en su ul
        if (hijos[indice].numClavesNodo < GradoMin) { //Llenar nodo hijo si no esta lleno
            llenar(indice);
        }
        if (bandera && indice > numClavesNodo) { //revisar si hay que entrar a la pos anterior d
            hijos[indice - 1].eliminar(key);
        } else {
            hijos[indice].eliminar(key);
        }
    }
}
```

- Método eliminarNoHoja

Este método es utilizado para eliminar un nodo que no es hoja, se obtiene el valor del nodo luego se valida si la cantidad de claves del hijo de mayor o igual al grado, si lo es se obtiene el predecesor y se sustituye el valor del nodo predecesor en la posición con el que se quiere eliminar y se eliminar el predecesor. Si no sucede se valida si es mayor o igual si lo es se obtiene el sucesor y se realiza el mismo proceso. Si no sucede ninguno de lo anterior se hace la instancia del metodo unir y se elimina el nodo.

```
public void elminarNoHoja(int indice) {
    long key = ClavesNodo[indice];
    if (hijos[indice].numClavesNodo >= GradoMin) { //la cantidad
        long predecesor = getPredecesor(indice);
        ClavesNodo[indice] = predecesor;
        hijos[indice].eliminar(predecesor);
    } else if (hijos[indice + 1].numClavesNodo >= GradoMin) { //
        long sucesor = getSucesor(indice);
        ClavesNodo[indice] = sucesor;
        hijos[indice + 1].eliminar(sucesor);
    } else { //tienen menos los dos unir
        unir(indice); //
        hijos[indice].eliminar(key);
    }
}
```

- Método eliminarDeHoja Árbol B

En este metodo se busca eliminar un nodo hoja, se recorre el arreglo y modifica el valor siguiente con el actual dentro de las claves y se reduce la cantidad de nodos clave.

```
public void eliminarDeHoja(int indice) {
    for (int i = indice + 1; i < numClavesNodo; ++i) {
        ClavesNodo[i - 1] = ClavesNodo[i];
    }
    numClavesNodo--;
}
```

- Método getPredecesor, getSucesor

Métodos utilizados para encontrar ya sea el valor siguiente a nodo, como al valor anterior al nodo y se retorna la clave en la primera posición.

```
public long getPredecesor(int indice) {
    // Moverse al nodo más a la derecha hasta que llegue al nodo hoja
    NodoB aux = hijos[indice];
    while (!aux.isHoja()) {
        aux = aux.hijos[aux.numClavesNodo];
    }
    return aux.ClavesNodo[aux.numClavesNodo - 1];
}

public long getSucesor(int idx) {
    //Mover al nodo más a la izquierda de los hijos hasta que llegue al
    NodoB sucesor = hijos[idx + 1];
    while (!sucesor.isHoja()) {
        sucesor = sucesor.hijos[0];
    }
    return sucesor.ClavesNodo[0];
}
```

- Método insertar Lista Simple

Este método es utilizado para insertar los albumes en una lista simple para ello se crea un nuevo objeto y se aumenta de tamaño la cantidad de nodos de la lista, luego se valida si la cabeza de la lista es igual a nulo, es decir, no existe ningun elemento, si es así la nueva cabeza es el nuevo nodo, si no se recorre la lista y se valida cuando el siguiente del nodo actual es nulo, para poder insertarlo en esa posición y se retorna el nuevo album.

```
public Album insertarAlbum(String nombre, ListaSimple imagenes) {
    Album nuevo = new Album(nombre, imagenes);
    this.sizeAlbum++;
    if (this.getCabezaAlbum() == null) {
        this.setCabezaAlbum(nuevo);
    } else {
        Album actual = this.getCabezaAlbum();
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();
        }
        actual.setSiguiente(nuevo);
    }
    return nuevo;
}
```

- Método insertar Matriz Dispersa

Este método es utilizado para insertar un nodo en la matriz dispesa, para ello se crea un nuevo nodo con los parametros necesarios y se obitenen los nodos encabezado, se valida si los nodos encabezados son nulos para crearlos. Luego se insertar el nuevo en las filas, se valida si el acceso es nulo si lo es se crea uno, si no se obtiene la información del acceso y se busca la posición donde va el nuevo nodo. Seguido se insertar el nuevo en las columnas, se valida si el acceso es nulo si lo es se crea uno, si no se obtiene la información del acceso y se busca la posición donde va el nuevo nodo.

```
public void insertar(int idCapa, int posX, int posY, String color) {
    NodoMatriz nuevo = new NodoMatriz(idCapa, color, posX, posY);
    //buscar si ya existen encabezados
    NodoEncabezado nodo_X = this.filas.getEncabezado(posX);
    NodoEncabezado nodo_Y = this.columnas.getEncabezado(posY);
    //crearlos
    if (nodo_X == null) {
        nodo_X = new NodoEncabezado(posX);
        this.filas.insertarNodoEncabezado(nodo_X);
    }
    if (nodo_Y == null) {
        nodo_Y = new NodoEncabezado(posY);
        this.columnas.insertarNodoEncabezado(nodo_Y);
    }
```

```
//Insertar nuevo en Fila
if (nodo_X.getAcceso() == null) { //si no apunta a un nodo
    nodo_X.setAcceso(nuevo);
} else { //validar si la posicion de la columna del nuevo nodo es menor a la posici
    if (nuevo.getY() < nodo_X.getAcceso().getY()) {
        nuevo.setSiguiente(nodo_X.getAcceso());
        nodo_X.getAcceso().setAnterior(nuevo);
        nodo_X.setAcceso(nuevo);
    } else { //buscar posicion donde va
        NodoMatriz tmp = nodo_X.getAcceso();
        while (tmp != null) {
            if (nuevo.getY() < tmp.getY()) {
                nuevo.setSiguiente(tmp);
                nuevo.setAnterior(tmp.getAnterior());
                tmp.getAnterior().setSiguiente(nuevo);
                tmp.setAnterior(nuevo);
                break;
            } else if (nuevo.getX() == tmp.getX() && nuevo.getY() == tmp.getY()) {
                break;
            } else {
                if (tmp.getSiguiente() == null) {
                    tmp.setSiguiente(nuevo);
                    nuevo.setAnterior(tmp);
                    break;
                } else {
                    tmp = tmp.getSiguiente();
                }
            }
        }
    }
}
```

```
//Insertar nuevo en Columna
if (nodo_Y.getAcceso() == null) { //si no apunta a un nodo
    nodo_Y.setAcceso(nuevo);
} else { //validar si la posicion de la columna del nuevo nodo es menor a la posici
    if (nuevo.getX() < nodo_Y.getAcceso().getX()) {
        nuevo.setAbajo(nodo_Y.getAcceso());
        nodo_Y.getAcceso().setArriba(nuevo);
        nodo_Y.setAcceso(nuevo);
    } else { //buscar posicion donde va
        NodoMatriz tmp = nodo_Y.getAcceso();
        while (tmp != null) {
            if (nuevo.getX() < tmp.getX()) {
                nuevo.setAbajo(tmp);
                nuevo.setArriba(tmp.getArriba());
                tmp.getArriba().setAbajo(nuevo);
                tmp.setArriba(nuevo);
                break;
            } else if (nuevo.getX() == tmp.getX() && nuevo.getY() == tmp.getY()) {
                break;
            } else {
                if (tmp.getAbajo() == null) {
                    tmp.setAbajo(nuevo);
                    nuevo.setArriba(tmp);
                    break;
                } else {
                    tmp = tmp.getAbajo();
                }
            }
        }
    }
}
```

- Método cargaMasivaJson

Este método es utilizado para realizar la carga masiva del archivo json, para ello se hace la instancia del método leerArchivoJson, luego se hace uso de la librería gson y se hace un parser de la información obtenido, luego se convierte el Json de tipo elemento a tipo objeto. Luego esta variable se realiza un parse con un JsonArray, seguido de esto se recorre el JsonArray y se obtiene los atributos y se hace la instancia del método insertar.

```
public boolean cargaMasivaCliente(String Jsontxt, Registro registro) {
    JsonParser parser = new JsonParser();
    JsonArray gsonArr = parser.parse(Jsontxt).getAsJsonArray();
    for (JsonElement objt : gsonArr) {
        JsonObject gsonObj = objt.getAsJsonObject();
        String dpi = gsonObj.get("dpi").getAsString().trim();
        String nombre = gsonObj.get("nombre_cliente").getAsString().trim();
        String contrasena = gsonObj.get("password").getAsString().trim();
        NodoB user = registro.insertarUsuario(dpi, nombre, contrasena);
        long numDpi = Long.parseLong(dpi);
        if (user != null) {
            arbolB.insertar(numDpi);
        }
        if (user == null) {
            System.out.println("El DPI " + dpi + " ya existe");
        }
    }
    return true;
}
```

```java
public boolean cargaMasivaCapas(String Jsontxt, NodoB usuarioActual) {
    JsonParser parser = new JsonParser();
    JsonArray gsonArr = parser.parse(Jsontxt).getAsJsonArray();
    for (JsonElement objt : gsonArr) {
        JsonObject gsonObj = objt.getAsJsonObject();
        int id_capa = gsonObj.get("id_capa").getAsInt();
        NodoBinario nodoVerificar = usuarioActual.getCapasUser().inorderBus(usuarioActu
        if (nodoVerificar.getValor().equals(-1)) {
            JsonArray pixeles = gsonObj.get("pixeles").getAsJsonArray();
            for (JsonElement objt2 : pixeles) {
                JsonObject gsonObj2 = objt2.getAsJsonObject();
                int fila = gsonObj2.get("fila").getAsInt();
                int columna = gsonObj2.get("columna").getAsInt();
                String color = gsonObj2.get("color").getAsString();
                this.matrizPixeles.insertar(id_capa, fila, columna, color);
            }
            usuarioActual.getCapasUser().insertar(id_capa, this.matrizPixeles);//insert
            this.matrizPixeles = new MatrizDispersa(1);
        } else {
            System.out.println("La Capa " + id_capa + " ya existe");
        }
    }
    return true;
}
```

```java
public boolean cargaMasivaImagen(String Jsontxt, NodoB usuarioActual) {
    JsonParser parser = new JsonParser();
    JsonArray gsonArr = parser.parse(Jsontxt).getAsJsonArray();
    for (JsonElement objt : gsonArr) {
        JsonObject gsonObj = objt.getAsJsonObject();
        int id = gsonObj.get("id").getAsInt();
        NodoAVL nodoVerificar = usuarioActual.getImagenesUser().inorderBus(usua
        if (nodoVerificar.getValor().equals(-1)) {
            JsonArray capas = gsonObj.get("capas").getAsJsonArray();
            int contador = 0;
            for (JsonElement objt2 : capas) {
                contador++;
            }
            usuarioActual.getImagenesUser().insertar(id, capas, contador);
        } else {
            System.out.println("La Imagen " + id + " ya existe");
        }
    }
    return true;
}
```

```java
public boolean cargaMasivaAlbum(String Jsontxt, NodoB usuarioActual) {
    JsonParser parser = new JsonParser();
    JsonArray gsonArr = parser.parse(Jsontxt).getAsJsonArray();
    for (JsonElement objt : gsonArr) {
        JsonObject gsonObj = objt.getAsJsonObject();
        String nombreAlbum = gsonObj.get("nombre_album").getAsString();
        if (!usuarioActual.getAlbumUser().existeAlbum(nombreAlbum)) {
            JsonArray imgs = gsonObj.get("imgs").getAsJsonArray();
            for (JsonElement objt2 : imgs) {
                int id = objt2.getAsInt();
                this.imagenes.insertarImagen(id);
            }
            usuarioActual.getAlbumUser().insertarAlbum(nombreAlbum, this.imagenes);
            this.imagenes = new ListaSimple();
        }else{
            System.out.println("El Album con el nombre "+nombreAlbum+ " ya existe");
        }
    }
    return true;
}
```

- Método crearGrafico

Este método es utilizado para escribir el archivo para generar el grafo, para ello dentro de un try – catch se hace la instancia del método FileWriter en el cual se escribe como parámetro el nombre del archivo, luego se escribe el archivo con el método PrintWriter y se cierra dicho archivo; seguido de esto se hace uso de la librería Runtime la cual sirve para acceder a los comandos como si fuese una consola, dentro de los parámetros de este se escribe el comando para convertir un archivo con extensión dot a png, se hace una pausa de medio segundo con un hilo para evitar algún error y luego se imprime un mensaje en consola.

```
public void crearGrafico(String contenidoGrafica, String nombre) {
    try {
        FileWriter archivo = new FileWriter(nombre + ".dot", false);
        PrintWriter pw = new PrintWriter(archivo);
        pw.write(contenidoGrafica);
        pw.close();
        archivo.close();
        try {
            Runtime.getRuntime().exec("dot -Tpng " + nombre + ".dot -o " + nombre + ".png");
            //Esperar para evitar posibles errores mas adelante
            Thread.sleep(500);
            System.out.println("Gráfica generada exitosamente");
        } catch (Exception ex) {
            System.err.println("Error al generar la imagen del archivo .dot");
        }
    } catch (IOException e) {
        System.out.println("Error Archivo: " + e.getMessage());
    }
}
```