Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Estructuras de Datos

Ing. Luis Espino

Aux. Josué Pérez

MANUAL TÉCNICO

Elder Anibal Pum Rojas 201700761

Guatemala, 28 de diciembre del 2021

INTRODUCCIÓN

Se solicita crear una aplicación web mediante HTML y el lenguaje JavaScript en la cual se manejara por medio de Estructuras de datos Dinámicas los usuarios, clientes, calendario de actividades y catálogo de productos registrados en el sistema. Para lo cual se necesita que existan diferentes tipos de usuarios los cuales serán:

Administrador

Este usuario será el encargado de la carga masiva de datos y podrá ver todos los reportes de las estructuras de datos. Este usuario accede al sistema con el username Admin y el password 1234.

Empleados

Para esta ocasión los usuarios de tipo empleado serán vendedores, los cuales se almacenarán en un Árbol AVL, cada usuario vendedor tendrá una cartera de clientes asociada, los cuales se almacenarán en una lista doblemente enlazada.

Los datos para almacenar al vendedor son:

- ID
- Nombre
- Edad
- Correo
- Password
- Lista de clientes

Los datos para almacenar a los clientes son:

- ID
- Nombre
- Correo

Proveedores

Para almacenar los proveedores, se utiliza un Árbol Binario de Búsqueda, al cual solo el administrador tendrá acceso y podría realizar la carga de estos y ver el reporte respectivo. Los datos para almacenar los proveedores son:

- ID
- Nombre
- Dirección
- Teléfono
- Correo

OBJETIVOS

Objetivos Generales

- Aplicar los conocimientos adquiridos en el curso de Lenguajes Formales y de Programación en el desarrollo de la aplicación

Objetivos Específicos

- Familiarizarse con el lenguaje de programación JavaScript
- Envolverse en el ámbito del manejo de memoria
- Familiarizarse con el uso de Git
- Familiarizarse con el manejo de lectura de archivos
- Comprender el uso de estructuras de datos lineales y no lineales

ESPECIFICACIÓN TÉCNICA

Requisitos de Hardware

- RAM: 128 MB mínimo
- Procesador: Mínimo Pentium 2 a 266 Mhz

Requisitos de Software

- Sistema Operativo:
  - MS Windows XP o superior
  - Apple OSX 10.4.X o superior
  - GNU/LINUX 2.6.X o superior
- Exploradores Web:
  - Google Chrome (en todas las plataformas)
  - Mozilla Firefox (en todas las plataformas)
  - Internet Explorer (Windows)
  - Microsoft Edge (Windows, Android, iOS, Windows 10 móvil)
  - Safari (Mac, iOS)
  - Opera (Mac, Windows)
- Lenguaje de Programación e IDE:
  - Para el desarrollo de este software se utilizó JavaScript y como entorno se utilizó Visual Studio Code.
- Tecnologías aplicadas:
  - JavaScript
  - HTML
  - CSS
  - Bootstrap
  - Graphviz

EXPLICACIÓN GENERAL

Vendedores

Para la implementación de los usuarios vendedores se utilizó un árbol AVL, el cual contiene los datos de ID, nombre, edad, correo, password, por lo cual se implementó el nodo que se ve en la siguiente imagen:

```
class nodo{
    constructor(id, nombre, edad, correo, password){
        this.id = id
        this.nombre = nombre
        this.edad = edad
        this.correo = correo
        this.password = password
        this.izquierda = null
        this.derecha = null
        this.altura = 0
    }
}
```

Luego se procedió a construir el AVL utilizando este nodo, por lo cual fue necesario el uso de un método de Insertar y en el constructor de la clase un atributo llamado “raíz” para guardar el primer objeto y no perderlo en memoria, ya que este nodo contiene un atributo “izquierda”, refiriéndose al nodo que se almacena en la izquierda y un atributo “derecha” para referirse al nodo que se almacena en la derecha, por lo cual al continuar insertando se enlazaran los nodos nuevos con el primero que creamos evitando perderlos en memoria, tal y como se muestra a continuación:

```
insertar(id, nombre, edad, correo, password){
    let nuevo = new nodo(id, nombre, edad, correo, password);

    if (this.raiz == null) {
        this.raiz= nuevo
    } else {
        this.raiz = this.insertarNodo(this.raiz,nuevo)
    }
}
```

```
insertarNodo(raizActual, nuevo){
    if (raizActual != null) {
        //recorrer hijos
        if (raizActual.id > nuevo.id) {
            raizActual.izquierda = this.insertarNodo(raizActual.izquierda,nuevo)
            //validaciones

            if (this.altura(raizActual.derecha)-this.altura(raizActual.izquierda) == -2) {
                console.log("entra a rotacion IZQUIERDA")
                //if(this.altura(raizActual.izquierda.derecha)-this.altura(raizActual.izquierda.izquierda))
                if (nuevo.id < raizActual.izquierda.id) { //-1 ROTACION IZQUIERDA
                    console.log("entra a rotacion IZQUIERDA IZQUIERDA")
                    raizActual = this.rotacionIzquierda(raizActual)
                } else { //1 ROTACION izquierda-DERECHA
                    console.log("entra a rotacion IZQUIERDA DERECHA")
                    raizActual = this.rotacionIzquierdaDerecha(raizActual)
                }
            }
         }else if (raizActual.id < nuevo.id) {
            raizActual.derecha = this.insertarNodo(raizActual.derecha,nuevo)
            //validaciones
            if (this.altura(raizActual.derecha)-this.altura(raizActual.izquierda) == 2) {
                console.log("entra a rotacion DERECHA")
                if (nuevo.id > raizActual.derecha.id) { // 1 ROTACION DERECHA
                    console.log("entra a rotacion DERECHA DERECHA")
                    raizActual=this.rotacionDerecha(raizActual);
                } else {//-1 ROTACION DERECHA IZQUIERDA
                    console.log("entra a rotacion DERECHA IZQUIERDA")
                    raizActual = this.rotacionDerechaIzquierda(raizActual)
                }
            }

        } else {
            console.log("NO SE PUEDE INSERTAR EL DATO PORQUE YA EXISTE")
        }

        raizActual.altura = this.alturaMaxima(this.altura(raizActual.derecha),this.altura(raizActual.izquierda)) + 1
        return raizActual
    } else {
        raizActual = nuevo;
        return raizActual
    }
}
```

Posteriormente tenemos dos métodos utilizados para calcular la altura de nuestro árbol AVL:

```
altura(nodo) {
    if (nodo != null) {
        return nodo.altura
    } else {
        return -1
    }
}

alturaMaxima(h1,h2) {
    if (h2>=h1) { //************************ MAYOR O IGUAL */
        return h2
    } else {
        return h1
    }

}
```

Posteriormente, fue necesario la implementación de 4 métodos más, estos se encargan de hacer las rotaciones y con estas asegurar que sea un árbol AVL (que tenga un factor de balance de 0, 1 o -1), siendo estos métodos:

```
//ROTACIONES
//simple izquerda
rotacionIzquierda(nodo) {
    let aux = nodo.izquierda
    nodo.izquierda = aux.derecha
    aux.derecha = nodo
    nodo.altura = this.alturaMaxima(this.altura(nodo.derecha), this.altura(nodo.izquierda)) + 1
    aux.altura = this.alturaMaxima(nodo.altura.altura, this.altura(nodo.izquierda)) + 1
    return aux
}

//simple derecha
rotacionDerecha(nodo) {
    let aux = nodo.derecha
    nodo.derecha= aux.izquierda
    aux.izquierda = nodo
    nodo.altura = this.alturaMaxima(this.altura(nodo.izquierda),this.altura(nodo.derecha)) + 1
    aux.altura = this.alturaMaxima(nodo.altura.altura,this.altura(nodo.derecha)) + 1
    return aux
}

//rotacion izquierda-derecha
rotacionIzquierdaDerecha(nodo) {
    nodo.izquierda = this.rotacionDerecha(nodo.izquierda)
    let aux = this.rotacionIzquierda(nodo)
    return aux
}

//rotacion derecha-izquierda
rotacionDerechaIzquierda(nodo) {
    nodo.derecha = this.rotacionIzquierda(nodo.derecha)
    let aux = this.rotacionDerecha(nodo)
    return aux
}
```

Posteriormente fue necesario implementar funciones para los distintos tipos de órdenes para nuestro árbol:

```
preorden(raizActual) {
    if (raizActual != null) {
        console.log(raizActual.id)
        this.preorden(raizActual.izquierda)
        this.preorden(raizActual.derecha)
    }
}

inOrden(raizActual) {
    if (raizActual != null) {
        this.inOrden(raizActual.izquierda)
        console.log(raizActual.id)
        console.log("altura= " + (this.altura(raizActual.derecha)-this.altura(raizActual.izquierda)))
        this.inOrden(raizActual.derecha)
    }
}

postOrden(raizActual) {
    if (raizActual != null) {
        this.postOrden(raizActual.izquierda)
        this.postOrden(raizActual.derecha)
        console.log(raizActual.id)
    }
}
```

Y finalmente dos funciones más para generar el reporte mediante graphviz:

```
generarNodos(raizActual) { //metodo preorden
    let nodos ="";
    if (raizActual != null) {
        nodos += "n"+raizActual.id+"[label=\""+raizActual.id+", "+ raizActual.nombre+ ", " + raizActual.edad+ ", "+ raizActual.correo+ ", " +raizActual.password +"\"]\n"
        nodos += this.generarNodos(raizActual.izquierda)
        nodos += this.generarNodos(raizActual.derecha)
    }
    return nodos;
}

enlazar(raizActual) {
    let cadena = ""
    if(raizActual != null){
        cadena += this.enlazar(raizActual.izquierda)
        cadena += this.enlazar(raizActual.derecha)
        //validaciones
        if (raizActual.izquierda != null) {
            cadena+="n"+raizActual.id + "-> n"+raizActual.izquierda.id+"\n"
        }
        if (raizActual.derecha != null) {
            cadena+="n"+raizActual.id + "-> n"+raizActual.derecha.id+"\n"
        }
    }
    return cadena;
}
```

Clientes

Para la implementación de los clientes se utilizó una lista doblemente enlazada, la que está contenida en cada nodo vendedor que componen el AVL (cada vendedor contiene una lista de clientes) el cual contiene los datos de ID, nombre y correo.

Por lo cual se realizó un nodo que contuviera los datos, tal y como se muestra a continuación:

```
class Nodo {
    constructor(id, nombre, correo) {
        this.id = id
        this.nombre = nombre
        this.correo = correo
        this.siguiente = null
        this.anterior = null
    }
}
```

Luego se comenzó a construir la lista doblemente enlazada utilizando este nodo, por lo cual fue necesario el uso de un método Insertar y en el constructor de la clase un atributo llamado “primero” para guardar el primer objeto y no perderlo en la memoria, ya que este nodo contiene un atributo “siguiente”, refiriéndose al nodo que se almacena a continuación, y un atributo “anterior” refiriéndose al nodo que se almacena antes, por lo cual al continuar insertando se enlazan los nodos nuevos con el primero que creamos evitando perderlos en memoria, tal y como se muestra a continuación:

```
class ListaDobleClientes {
    constructor() {
        this.primero = null
        this.longitud = 0
    }

    insertar(id, nombre, correo) {
        let nuevo = new Nodo(id, nombre, correo)
        if (this.primero == null) { //Si la lista se encuentra vacia
            this.primero = nuevo
        } else {
            let aux = this.primero
            while (aux.siguiente != null) {
                if (aux.id == id) { //Si el id del cliente ya esta registrado, mostrara un mensaje de error
                    console.log("Valor ya ingresado, no se puede volver a insertar")
                    return
                }
                aux = aux.siguiente
            }
            if (aux.id == id) { //Lo mismo, si el id del cliente ya fue registrado, muestra el mismo mensaje de error
                console.log("Valor ya ingresado, no se puede volver a insertar")
                return
            }
            aux.siguiente = nuevo
            nuevo.anterior = aux
        }
    }
```

Lo siguiente fue un método para recorrer la lista creada mediante la consola:

```
mostrar() {
    let aux = this.primero
    console.log("///Mostrar Lista///")
    while (aux != null) {
        var idCliente = aux.id
        var nombreCliente = aux.nombre
        var correoCliente = aux.correo
        console.log("////////////////////////")
        console.log("Id: " + idCliente)
        console.log("Nombre: " + nombreCliente)
        console.log("Correo: " + correoCliente)
        console.log("////////////////////////")
        aux = aux.siguiente
    }
}
```

Lo siguiente fue un método que borra mediante la búsqueda del ID del cliente, lo cual el método lo que hace es buscar el ID y si es el único en la lista, lo borra sin más. La complejidad es cuando hay más elementos en la lista, por lo cual procede a eliminarlo y reacomodar los punteros de la lista hacia los elementos restantes de la lista, como se muestra a continuación:

```
borrar(id) {
    if (this.primero.id == id) {
        this.primero = this.primero.siguiente
        this.longitud--
        if (this.primero != null) {
            this.primero.anterior = null
        }
    } else {
        let aux = this.primero
        while (aux.siguiente.id) {
            if (aux.siguiente.id == id) {
                aux.siguiente = aux.siguiente.siguiente
                this.longitud--
                if (aux.siguiente != null) {
                    aux.siguiente.anterior = aux
                }
                break
            }
            aux = aux.siguiente
        }
    }
}
```

Finalmente, unas funciones (que son generales para todos los métodos) que sirven para recuperar datos y las funciones respectivas para mostrar, insertar y borrar datos en la lista y que estos cambios se reflejen en los elementos usados en el archivo HTML:

```
let listaClientes = new ListaDobleClientes()

function imprimirLista() {
    listaClientes.mostrar()
}

function recuperarLista() {
    var listaTemporal = JSON.parse(sessionStorage.getItem("ListaDobleClientes"))
    listaClientes = new ListaDobleClientes()
    listaTemporal = CircularJSON.parse(listaTemporal)
    Object.assign(listaClientes, listaTemporal)
}

function insertarLista() {
    let idNuevo = document.getElementById("idCliente").value
    let nombreNuevo = document.getElementById("nombreCliente").value
    let correoNuevo = document.getElementById("correoCliente").value
    listaClientes.insertar(idNuevo, nombreNuevo, correoNuevo)
    alert("Cliente ingresado correctamente")
    document.getElementById("idCliente").value = ""
    document.getElementById("nombreCliente").value = ""
    document.getElementById("correoCliente").value = ""
    imprimirLista()
}

function borrarEnLista() {
    let idBuscar = document.getElementById("idCliente").value
    listaClientes.borrar(idBuscar)
    alert("Cliente borrado correctamente")
    document.getElementById("idCliente").value = ""
    document.getElementById("nombreCliente").value = ""
    document.getElementById("correoCliente").value = ""
    imprimirLista()
}
```

Meses

Para la implementación de los eventos, primero se utilizó una lista doblemente enlazada, la que está contenida en cada nodo Vendedor que componen el árbol AVL (cada vendedor tiene su lista de meses) el cual contiene el mes en que se realiza el evento, este necesita el dato de cada mes.

Por lo cual se realizó un nodo el cual estuviera estos datos:

```
class Nodo {
    constructor(mes) {
        this.mes = mes
        this.siguiente = null
        this.anterior = null
    }
}
```

Así como se explicó en el apartado de Clientes, es el mismo concepto para los meses, por lo cual tenemos los siguientes apartados de código:

```
class ListaDobleMeses {
    constructor() {
        this.primero = null
        this.longitud = 0
    }

    insertar(mes) {
        let nuevo = new Nodo(mes)
        if (this.primero == null) { //Si la lista se encuentra vacia
            this.primero = nuevo
        } else {
            let aux = this.primero
            while (aux.siguiente != null) {
                if (aux.mes == mes) { //Si el id del cliente ya esta registrado, mostrara un mensaje de error
                    console.log("Valor ya ingresado, no se puede volver a insertar")
                    return
                }
                aux = aux.siguiente
            }
            if (aux.mes == mes) { //Lo mismo, si el id del cliente ya fue registrado, muestra el mismo mensaje de error
                console.log("Valor ya ingresado, no se puede volver a insertar")
                return
            }
            aux.siguiente = nuevo
            nuevo.anterior = aux
        }
    }
```

```
mostrar() {
    let aux = this.primero
    console.log("///Mostrar Lista///")
    while (aux != null) {
        var mesIngresado = aux.mes
        console.log("////////////////////////")
        console.log("Mes: " + mesIngresado)
        aux = aux.siguiente
    }
}

borrar(mes) {
    if (this.primero.mes == mes) {
        this.primero = this.primero.siguiente
        this.longitud--
        if (this.primero != null) {
            this.primero.anterior = null
        }
    } else {
        let aux = this.primero
        while (aux.siguiente.mes) {
            if (aux.siguiente.mes == mes) {
                aux.siguiente = aux.siguiente.siguiente
                this.longitud--
                if (aux.siguiente != null) {
                    aux.siguiente.anterior = aux
                }
                break
            }
            aux = aux.siguiente
        }
    }
}
```

```
let listaMeses = new ListaDobleMeses()

function imprimirLista() {
    listaMeses.mostrar()
}

function recuperarLista() {
    var listaTemporal = JSON.parse(sessionStorage.getItem("ListaDobleMeses"))
    listaMeses = new ListaDobleMeses()
    listaTemporal = CircularJSON.parse(listaTemporal)
    Object.assign(listaMeses, listaTemporal)
}

function insertarLista() {
    let mesNuevo = document.getElementById("mesCalendario").value
    listaMeses.insertar(mesNuevo)
    alert("Mes ingresado correctamente")
    document.getElementById("mesCalendario").value = ""
    imprimirLista()
}

function borrarLista() {
    let mesBuscar = document.getElementById("mesCalendario").value
    listaMeses.borrar(mesBuscar)
    alert("Mes borrado correctamente")
    document.getElementById("mesCalendario").value = ""
    imprimirLista()
}
```

Evento

Para la implementación de los eventos luego de ingresar el mes en la lista, si este no existe dentro de cada nodo se crea o se utiliza una matriz dinámica en la cual se almacena los datos restantes para componer el evento, los cuales son: Día, Hora y Descripción.

Por lo cual se realizó un nodo el cual contuviera estos datos, tal y como se muestra a continuación:

```
class nodo_interno{
    constructor(valor,x,y){
        this.valor = valor; //descripcion
        this.x = x; //dia
        this.y = y; //hora
        //apuntadores
        this.sig = null;
        this.ant = null;

        this.arriba = null;
        this.abajo = null;
    }
}
```

Luego se comenzó a construir la matriz dinámica utilizando este nodo, por lo cual fue necesario el uso de un método Insertar en el cual se comienzan a crear nodos cabecera, los cuales son necesarios para posicionar el evento en el día y hora que se solicita, relacionándolas y agregándole la descripción (la cual se crea o ingresa en una lista interna que luego se relaciona con los nodos cabecera por medio de los atributos “sig”, “ant”, “arriba” y “abajo”, por lo cual el método de inserción se relaciona con la inserción a estas otras estructuras como se puede apreciar en el siguiente código:

```
class lista_interna{
    constructor(){
        this.primero = null;
    }

    insertar_x(valor, x,y){ //para las X usamos sig y ant, y el valor para compara y ordenar es Y
        let nuevo = new nodo_interno(valor,x,y);

        if(this.primero == null){
            this.primero = nuevo;
        }else{
            if(nuevo.y < this.primero.y){
                nuevo.sig = this.primero;
                this.primero.ant = nuevo;
                this.primero = nuevo;
            }else{
                let aux = this.primero;
                while(aux != null){
                    if(nuevo.y < aux.y){
                        nuevo.sig = aux;
                        nuevo.ant = aux.ant;
                        aux.ant.sig = nuevo;
                        aux.ant= nuevo;
                        break;
                    }else if(nuevo.x == aux.x && nuevo.y == aux.y){
                        console.log("La posicion ya esta ocupada-> "+nuevo.x+","+nuevo.y);
                        break;
                    }else{
                        if(aux.sig ==null){
                            aux.sig=nuevo;
                            nuevo.ant = aux;
                            break;
                        }else{
                            aux = aux.sig;
                        }
                    }
                }
            }
        }
    }
```

```
insertar_y(valor, x,y){ //para las Y usamos arriba y abajo, y el valor para compara y ordenar es X
    let nuevo = new nodo_interno(valor,x,y);

    if(this.primero == null){
        this.primero = nuevo;
    }else{
        if(nuevo.x < this.primero.x){
            nuevo.abajo = this.primero;
            this.primero.arriba = nuevo;
            this.primero = nuevo;
        }else{
            let aux = this.primero;
            while(aux != null){
                if(nuevo.x < aux.x){
                    nuevo.abajo = aux;
                    nuevo.arriba = aux.arriba;
                    aux.arriba.abajo = nuevo;
                    aux.arriba= nuevo;
                    break;
                }else if(nuevo.x == aux.x && nuevo.y == aux.y){
                    console.log("La posicion ya esta ocupada-> "+nuevo.x+","+nuevo.y);
                    break;
                }else{
                    if(aux.abajo ==null){
                        aux.abajo=nuevo;
                        nuevo.arriba = aux;
                        break;
                    }else{
                        aux = aux.abajo;
                    }
                }
            }
        }
    }
}
```

Luego tenemos los métodos para poder recorrer la matriz, tanto en X como en Y:

```
    recorrer_x(){
        let aux = this.primero;
        while(aux != null){
            console.log("valor =",aux.valor," - x = ",aux.x , " y = ",aux.y);
            aux = aux.sig;
        }
    }
    recorrer_y(){
        let aux = this.primero;
        while(aux != null){
            console.log("valor =",aux.valor," - x = ",aux.x , " y = ",aux.y);
            aux = aux.abajo;
        }
    }
}
```

Después tenemos los métodos para las cabeceras que nos servirán en nuestra matriz:

```
class lista_cabecera{
    constructor(){
        this.primero = null;
    }

    insertar_cabecera(nuevo){

        if(this.primero == null){
            this.primero = nuevo;
        }else{
            if(nuevo.dato<this.primero.dato){
                nuevo.sig = this.primero;
                this.primero.ant=nuevo;
                this.primero = nuevo;
            }else{
                let aux = this.primero;
                while(aux != null){
                    if(nuevo.dato < aux.dato){
                        nuevo.sig = aux;
                        nuevo.ant = aux.ant;
                        aux.ant.sig = nuevo;
                        aux.ant = nuevo;
                        break;
                    }else{
                        if(aux.sig == null){
                            aux.sig = nuevo;
                            nuevo.ant = aux;
                            break;
                        }else{
                            aux = aux.sig;
                        }
                    }
                }
            }
        }
    }
```

```
    buscar_cabecera(dato){
        let aux = this.primero;
        while(aux != null){
            if(aux.dato == dato){
                return aux;
            }else{
                aux = aux.sig;
            }
        }
        return null;
    }

    recorrer(){
        let aux = this.primero;
        while(aux != null){
            console.log("dato =",aux.dato);
            aux = aux.sig;
        }
    }
}
```

Luego tenemos lo importante, que es la propia construcción de la matriz:

```
class matriz{
    constructor(){
        this.cabecetas_x = new lista_cabecera();
        this.cabecetas_y = new lista_cabecera();
    }

    insertar(valor,x,y){
        let nodo_cabecera_X = this.cabecetas_x.buscar_cabecera(x);
        let nodo_cabecera_y = this.cabecetas_y.buscar_cabecera(y);

        if(nodo_cabecera_X == null){
            nodo_cabecera_X =  new nodo_cabecera(x);
            this.cabecetas_x.insertar_cabecera(nodo_cabecera_X);
        }

        if(nodo_cabecera_y == null){
            nodo_cabecera_y =  new nodo_cabecera(y);
            this.cabecetas_y.insertar_cabecera(nodo_cabecera_y);
        }

        //insertar en cabecera X
        nodo_cabecera_X.lista_interna.insertar_x(valor,x,y);
        //insertar en cabecera Y
        nodo_cabecera_y.lista_interna.insertar_y(valor,x,y);
    }
```

```
recorrer_matriz(){
    console.log("cabeceras en X");
    let aux = this.cabecetas_x.primero;
    while(aux != null){
        console.log("   pos->"+aux.dato);
        let aux2 = aux.lista_interna.primero;
        while(aux2!= null){
            console.log("       -"+aux2.valor);
            aux2 = aux2.sig;
        }
        aux = aux.sig;
    }

    console.log("cabeceras en Y");
    aux = this.cabecetas_y.primero;
    while(aux != null){
        console.log("   pos->"+aux.dato);
        let aux2 = aux.lista_interna.primero;
        while(aux2!= null){
            console.log("       -"+aux2.valor);
            aux2 = aux2.abajo;
        }
        aux = aux.sig;
    }
}
```

```
graficar_matriz(){
    let cadena="";
    cadena+= "digraph Matriz{ \n";
    cadena+= "node[shape = box,width=0.7,height=0.7,fillcolor=\"azure2\" color=\"white\" style=\"filled\"];\n";
    cadena+= "edge[style = \"bold\"]; \n"
    //graficar el nodo matriz
    cadena+="node[label = Matriz fillcolor=\" darkolivegreen1\" pos = \"-1,1!\"]principal;"
    //graficar cabeceras X
    let aux_x = this.cabecetas_x.primero;
    while(aux_x!=null){
        cadena+="node[label = "+aux_x.dato+" fillcolor=\" azure1\" pos = \""+aux_x.dato+",1!\"]x"+aux_x.dato+";\n"
        aux_x = aux_x.sig;
    }
    aux_x = this.cabecetas_x.primero;
    while(aux_x.sig != null){
        cadena+="x"+aux_x.dato+"->"+"x"+aux_x.sig.dato+";\n"
        cadena+="x"+aux_x.sig.dato+"->"+"x"+aux_x.dato+";\n"
        aux_x = aux_x.sig;
    }

    if(this.cabecetas_x.primero!= null){
        cadena+="principal->x"+this.cabecetas_x.primero.dato+";\n";
    }
    //graficar cabeceras Y
    let aux_y = this.cabecetas_y.primero;
    while(aux_y!=null){
        cadena+="node[label = "+aux_y.dato+" fillcolor=\" azure1\" pos = \"-1,-"+aux_y.dato+"!\"]y"+aux_y.dato+";\n"
        aux_y = aux_y.sig;
    }
    aux_y = this.cabecetas_y.primero;
    while(aux_y.sig != null){
        cadena+="y"+aux_y.dato+"->"+"y"+aux_y.sig.dato+";\n"
        cadena+="y"+aux_y.sig.dato+"->"+"y"+aux_y.dato+";\n"
        aux_y = aux_y.sig;
    }

    if(this.cabecetas_x.primero!= null){
        cadena+="principal->y"+this.cabecetas_y.primero.dato+";\n";
    }
```

```
        //graficar nodos internos
        aux_x = this.cabecetas_x.primero;
        while(aux_x!=null){ //recorrer listas de x para graficar los nodos de sus lista interna
            let aux = aux_x.lista_interna.primero;
            while(aux!=null){
                cadena+="   node[label = "+aux.valor+" fillcolor=\" gold2\" pos = \""+aux.x+",-"+aux.y+"!\"]x"+aux.x+"y"+aux.y+";\n"
                aux = aux.sig;
            }

            //graficar flechitas
            aux = aux_x.lista_interna.primero;
            while(aux.sig!= null){
                cadena+="   x"+aux.x+"y"+aux.y+"->x"+aux.sig.x+"y"+aux.sig.y+";\n";
                cadena+="   x"+aux.sig.x+"y"+aux.sig.y+"->x"+aux.x+"y"+aux.y+";\n";
                aux= aux.sig;
            }
            if(aux_x.lista_interna.primero!= null){
                cadena+="x"+aux_x.dato+"->"+"x"+aux_x.lista_interna.primero.x+"y"+aux_x.lista_interna.primero.y+";\n";
            }

            aux_x = aux_x.sig;
        }

        aux_y = this.cabecetas_y.primero;
        while(aux_y!=null){ //recorrer la lista de y para graficar cada lista
            //graficar flechitas Y
            let aux = aux_y.lista_interna.primero;
            while(aux.abajo!= null){
                cadena+="   x"+aux.x+"y"+aux.y+"->x"+aux.abajo.x+"y"+aux.abajo.y+";\n";
                cadena+="   x"+aux.abajo.x+"y"+aux.abajo.y+"->x"+aux.x+"y"+aux.y+";\n";
                aux= aux.abajo;
            }
            if(aux_y.lista_interna.primero!= null){
                cadena+="y"+aux_y.dato+"->"+"x"+aux_y.lista_interna.primero.x+"y"+aux_y.lista_interna.primero.y+";\n";
            }
            aux_y = aux_y.sig;
        }

        cadena+= "\n}"
        console.log(cadena);
    }
}
```

```
let matriz1 = new matriz();

function imprimirMatriz() {
    matriz1.recorrer_matriz()
}

function recuperarMatriz() {
    var matrizTemporal = JSON.parse(sessionStorage.getItem("matriz"))
    matriz1 = new matriz()
    matrizTemporal = CircularJSON.parse(matrizTemporal)
    Object.assign(matriz1, matrizTemporal)
}

function insertarMatriz() {
    let diaNuevo = document.getElementById("diaCalendario").value
    let horaNuevo = document.getElementById("horaCalendario").value
    let descripcionNuevo = document.getElementById("descripcionCalendario").value
    matriz1.insertar(descripcionNuevo, diaNuevo, horaNuevo)
    alert("Evento agregado exitosamente")
    document.getElementById("diaCalendario").value = ""
    document.getElementById("horaCalendario").value = ""
    document.getElementById("descripcionCalendario").value = ""
    imprimirMatriz()
}

function graficar() {
    matriz1.graficar_matriz()
}
```

Proveedores

Para la implementación de los proveedores se utilizó un árbol ABB, el cual contiene los datos de ID, nombre, dirección, teléfono y correo.

Por lo cual se realizó un nodo el cual contuviera estos datos, tal y como se muestra a continuación:

```
class Nodo {
    constructor(id, nombre, direccion, telefono, correo) {
        this.id = id
        this.nombre = nombre
        this.direccion = direccion
        this.telefono = telefono
        this.correo = correo
        this.izquierda = null
        this.derecha = null
    }
}
```

Luego se comenzó a construir un árbol ABB utilizando este nodo, por lo cual fue necesario el uso de un método Insertar y en el constructor de la clase un atributo llamado “raíz” para guardar el primer objeto y no perderlo en la memoria, ya que este nodo contiene un atributo “izquierda” para referirse al nodo que se almacena a la izquierda y además contiene un atributo “derecha” para referirse al nodo que se almacena a la derecha, por lo cual al continuar insertando se enlazaran los nodos nuevos con el primero que creamos, evitando perderlos en memoria, tal y como se muestra a continuación:

```
class ABB {
    constructor() {
        this.raiz = null
    }

    insertar(id, nombre, direccion, telefono, correo) {
        let nuevo = new Nodo(id, nombre, direccion, telefono, correo)
        if (this.raiz == null) {
            this.raiz = nuevo
        } else {
            this.raiz = this.insertarNodo(this.raiz.nuevo)
        }
    }
```

```
insertarNodo(raizActual, nuevo) {
    if (raizActual != null) {
        //Recorrer hijos del arbol
        if (raizActual.id > nuevo.id) {
            raizActual.izquierda = this.insertarNodo(raizActual.izquierda, nuevo)
        } else if (raizActual.id < nuevo.id) {
            raizActual.derecha = this.insertarNodo(raizActual.derecha, nuevo)
        } else {
            console.log("No se puede insertar, porque ya existe el proveedor")
        }

        return raizActual
    } else {
        raizActual = nuevo
        return raizActual
    }
}
```

Luego tenemos un método para borrar un nodo del árbol:

```
borrar(id) {
    let aux = this.raiz
    let padre = this.raiz
    let esHijoIzq = true
    while (aux.id != id) {
        padre = aux
        if (id < aux.id) {
            esHijoIzq = true
            aux = aux.izquierda
        } else {
            esHijoIzq = false
            aux = aux.derecha
        }
        if (aux == null) {
            return false
        }
    }
```

```
        //Salimos de la busqueda del elemento
        if (aux.izquierda == null && aux.derecha == null) {
            if (aux == this.raiz) {
                this.raiz == null
            } else if (esHijoIzq) {
                padre.izquierda = null
            } else {
                padre.derecha = null
            }
        } else if (aux.derecha == null) {
            if (aux == this.raiz) {
                this.raiz = aux.izquierda
            } else if (esHijoIzq) {
                padre.izquierda = aux.izquierda
            } else {
                padre.derecha = aux.izquierda
            }
        } else if (aux.izquierda == null) {
            if (aux == this.raiz) {
                this.raiz = aux.derecha
            } else if (esHijoIzq) {
                padre.izquierda = aux.derecha
            } else {
                padre.derecha = aux.derecha
            }
        } else {
            let reemplazo = this.obtenerReemplazo(aux)
            if (aux == this.raiz) {
                this.raiz = reemplazo
            } else if (esHijoIzq) {
                padre.izquierda = reemplazo
            } else {
                padre.derecha = reemplazo
            }
            reemplazo.izquierda = aux.izquierda
        }
        console.log("Nodo: " + id + " eliminado correctamente")
        return true
    }
```

```
obtenerReemplazo(nodoReemplazo) {
    let reemplazarPadre = nodoReemplazo
    let reemplazo = nodoReemplazo
    let aux = nodoReemplazo.derecha
    while (aux != null) {
        reemplazarPadre = reemplazo
        reemplazo = aux
        aux = aux.izquierda
    }
    if (reemplazo != nodoReemplazo.derecha) {
        reemplazarPadre.izquierda = reemplazo.derecha
        reemplazo.derecha = nodoReemplazo.derecha
    }
    return reemplazo
}
```

Ahora tenemos métodos para los distintos tipos de orden en nuestro árbol:

```
preOrden(raizActual) {
    if (raizActual != null) {
        console.log(raizActual.id)
        this.preOrden(raizActual.izquierda)
        this.preOrden(raizActual.derecha)
    }
}

inOrden(raizActual) {
    if (raizActual != null) {
        this.inOrden(raizActual.izquierda)
        console.log(raizActual.id)
        this.inOrden(raizActual.derecha)
    }
}

postOrden(raizActual) {
    if (raizActual != null) {
        this.postOrden(raizActual.izquierda)
        this.postOrden(raizActual.derecha)
        console.log(raizActual.id)
    }
}
```

Y métodos para poder generar el reporte en graphviz:

```
generarDot() {
    let cadena = "digraph arbol {\n";
    cadena += this.generarNodos(this.raiz)
    cadena +="\n"
    cadena +=this.enlazar(this.raiz)
    cadena +="\n}"

    console.log(cadena)
}

generarNodos(raizActual) { //metodo de preorden
    let nodos = ""
    if (raizActual != null) {
        nodos += "n" + raizActual.id + "[label=\"" + raizActual.id + ", "+ raizActual.nombre+ ", "+ raizActual.direccion+ ", "+ raizActual.telefono+ ", "+ raizActual.correo + "\"]
        nodos += this.generarNodos(raizActual.izquierda)
        nodos += this.generarNodos(raizActual.derecha)
    }
    return nodos
}

enlazar(raizActual) {
    let cadena = ""
    if (raizActual != null) {
        cadena += this.enlazar(raizActual.izquierda)
        cadena += this.enlazar(raizActual.derecha)

        if (raizActual.izquierda != null) {
            cadena += "n" + raizActual.id + "-> n" + raizActual.izquierda.id + "\n"
        }
        if (raizActual.derecha != null) {
            cadena += "n" + raizActual.id + "-> n" + raizActual.derecha.id + "\n"
        }
    }
    return cadena
}
```

```
let abbProveedores = new ABB()

function mostrarInOrden() {
    abbProveedores.inOrden()
}

function recuperarABB() {
    var arbolTemporal = JSON.parse(sessionStorage.getItem("ABB"))
    abbProveedores = new ABB()
    arbolTemporal = CircularJSON.parse(arbolTemporal)
    Object.assign(abbProveedores, arbolTemporal)
}

function insertarArbol() {
    let idNuevo = document.getElementById("idProveedor").value
    let nombreNuevo = document.getElementById("nombreProveedor").value
    let direccionNuevo = document.getElementById("direccionProveedor").value
    let telefonoNuevo = document.getElementById("telefonoProveedor").value
    let correoNuevo = document.getElementById("correoProveedor").value
    abbProveedores.insertar(idNuevo, nombreNuevo, direccionNuevo, telefonoNuevo, correoNuevo)
    alert("Proveedor ingresado exitosamente")
    document.getElementById("idProveedor").value = ""
    document.getElementById("nombreProveedor").value = ""
    document.getElementById("direccionProveedor").value = ""
    document.getElementById("telefonoProveedor").value = ""
    document.getElementById("correoProveedor").value = ""
    mostrarInOrden()
}

function borrarArbol() {
    let idBuscar = document.getElementById("idProveedor").value
    abbProveedores.borrar(idBuscar)
    alert("Proveedor borrado correctamente")
    document.getElementById("idProveedor").value = ""
    document.getElementById("nombreProveedor").value = ""
    document.getElementById("direccionProveedor").value = ""
    document.getElementById("telefonoProveedor").value = ""
    document.getElementById("correoProveedor").value = ""
    mostrarInOrden()
}

function graficar() {
    abbProveedores.generarDot()
}
```

CONCLUSIÓN

Conocer el uso de las estructuras de datos es importante, ya que como futuros ingenieros deberemos desarrollar soluciones que sean las más eficientes para no utilizar tantos recursos en los dispositivos donde se ejecute nuestra aplicación, por lo cual, al implementar estructuras de datos en la memoria RAM logramos optimizar procesos de búsqueda y almacenamiento con lo cual podemos lograr el objetivo de tener una aplicación funcional y bien optimizada.