MANUAL TÉCNICO

Fase #1

Estructura de Datos

Javier Alejandro Gutierrez de León

202004765

Guatemala, 28 de diciembre de 2021

Introducción

Se 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 catalogo 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 pasword: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 del vendedor son:

- Id
- Nombre
- Edad
- Correo
- password
- Lista de clientes

Los datos para almacenar de los clientes son:

- Id
- Nombre
- correo

Proveedores

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

- id
- nombre
- dirección
- teléfono
- correo

Objetivos

Objetivos Generales

- ✔ Aplicar los conceptos adquiridos en el curso de Lenguajes Formales y de Programación en el desarrollo de la aplicación.

Objetivos Específicos

- ✔ Familiarice con el lenguaje de programación JavaScript.
- ✔ Envolverse en el ámbito del manejo de la memoria.
- ✔ Familiarizarse con el uso de Git.
- ✔ Familiarice 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
- 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:**
    - **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 utilizadas**
    - **JavaScript:** Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
    - **HTML5:** lenguaje de marcado para la elaboración de páginas web, con el que se realizaron los reportes generados desde Python.
    - **Tkinter:** Binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario para Python y es el que viene por defecto con la instalación para Microsoft Windows.
    - **Boostrap:** Librería para el diseño del reporte HTML.
    - **Graphviz:** Es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.

Estructuras Utilizadas

Vendedores:

Para la implementacion de los usuarios Vendedores se utilizo un arbol AVL, el cual contiene los datos:

- Id
- Nombre
- Edad
- Correo
- Password
- Lista de Clientes
- Calendario

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

```
class nodo_avl{
    constructor(id, nombre, edad, correo, pass){
        this.id=id;
        this.nombre=nombre;
        this.edad=edad;
        this.correo=correo;
        this.pass=pass;
        this.clientes=new listaD();
        this.calendario= new listaM();
        this.izq=null;
        this.der=null;
        this.altura=null;
    }
}
```

Luego se comenzó a construir el arbol AVL utilizando este nodo, por lo cual fue necesario el uso de un metodo insertar y en el constructor de la clase un atributo llamado *“raiz”* para guardar el primer objeto y no perderlo en la memoria, ya que este nodo contiene un atributo *“izq”,* refiriendose al nodo que se almacena a la izquierda, y un atributo *“der”,* refiriendose 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 continuacion:

```
class arbol_avl{
    constructor(){
        console.log("Inicializado AVL")
        this.raiz = null;
        this.listaa = null;
        this.dotM = ''
        this.busc=null
        this.tablaC=''
        this.nodoB = ''
    }
    insertar(id, nombre, edad, correo, pass){
        let nuevo = new nodo_avl(id, nombre, edad, correo, pass);

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

insertarNodo(raizA,nuevo){
        if (raizA!=null){
            if(raizA.id > nuevo.id){
                raizA.izq = this.insertarNodo(raizA.izq,nuevo)
                if(this.altura(raizA.der)-this.altura(raizA.izq)==-2){
//Rotacion I
                    //console.log("Rotacion Izquierda")
                    if(nuevo.id<raizA.izq.id){
                        //console.log("Rotacion Izquierda Izquierda")
                        raizA=this.RIzq(raizA)
                    }else{
                        //console.log("Rotacion Izquierda Derecha")
                        raizA=this.RIzqD(raizA)
                    }
                }
            }else if(raizA.id < nuevo.id){
                raizA.der = this.insertarNodo(raizA.der,nuevo)
                if(this.altura(raizA.der)-this.altura(raizA.izq)==2){
//Rotacion D
                    //console.log("Rotacion Derecha")
                    if(nuevo.id>raizA.der.id){
                        ///console.log("Rotacion Derecha Derecha")
                        raizA=this.RDer(raizA)
```

```
                    }else{
                        //console.log("Rotacion Derecha Izquierda")
                        raizA=this.RDerI(raizA)
                    }
                }
            }else{
                alert("Ya existe un Vendedor con ID = "+nuevo.id);
                console.log("Ya existe un Vendedor con ID = "+nuevo.id);
            }
            raizA.altura =
this.alturaM(this.altura(raizA.der),this.altura(raizA.izq))+1;
            return raizA
        }else{
            raizA=nuevo;
            //console.log('Ingresado '+nuevo.id)
            return raizA
        }
    }
```

Como se muestra anteriormente en el metodo insertar tambien fue necesario el uso de 4 metodos mas, los cuales se encargan de hacer las rotaciones y con estas asegurar que sea un arbol AVL (tenga un factor de balance de 0, 1 o -1), siendo estos metodos los que se muestran a continuacion:

```
RIzq(nodo){
        let aux = nodo.izq;
        nodo.izq= aux.der;
        aux.der = nodo;
        nodo.altura =
this.alturaM(this.altura(nodo.der),this.altura(nodo.izq)) +1;
        aux.altura =
this.alturaM(nodo.altura.altura,this.altura(nodo.izq))+1;
        return aux;
    }
RIzqD(nodo){
        nodo.izq = this.RDer(nodo.izq);
        let aux = this.RIzq(nodo);
        return aux;
    }
```

Para ciertas funciones en las cuales se necesita un dato en especifico como por ejemplo el login, se realizo un metodo de busqueda por ID, tal y como se muestra a continuacion:

```
buscar (id,raizA){
        console.log("Entre a buscar")
        if (raizA !=null){
            if (raizA.id==id){
                this.busc = raizA;
            }else if (raizA.id>id){
                this.buscar(id,raizA.izq)
            }else if (raizA.id<id){
                this.buscar(id,raizA.der)
            }
        }
    }
```

Tambien para mostrar los datos se realizaron metodos de recorrido del arbol como lo es el PreOrden (para pruebas) y el InOrden para mostrar los datos en los reportes generados con el fin de que estos se muestren en orden ascendente de ID, tal y como se muestra a continuacion:

```
PreOrder(raizA){
        if(raizA != null){
            console.log(raizA.id+' '+raizA.nombre);
            this.PreOrder(raizA.izq);
            this.PreOrder(raizA.der);
        }
    }

    Tabla(raizA){
        if (raizA!=null){
            this.Tabla(raizA.izq);
            this.tab += '<tr>'
            this.tab += '<td>'+raizA.id+'</td>'
            this.tab += '<td>'+raizA.nombre+'</td>'
            this.tab += '<td>'+raizA.edad+'</td>'
            this.tab += '<td>'+raizA.correo+'</td>'
            this.tab += '<td>'+raizA.pass+'</td>'

            this.tab += '<td>'
            this.tab += '<button value=\"'+raizA.id+'\"
onclick="eliminarV(this.value);VerA(\'Vendedor\');" style="background-
```

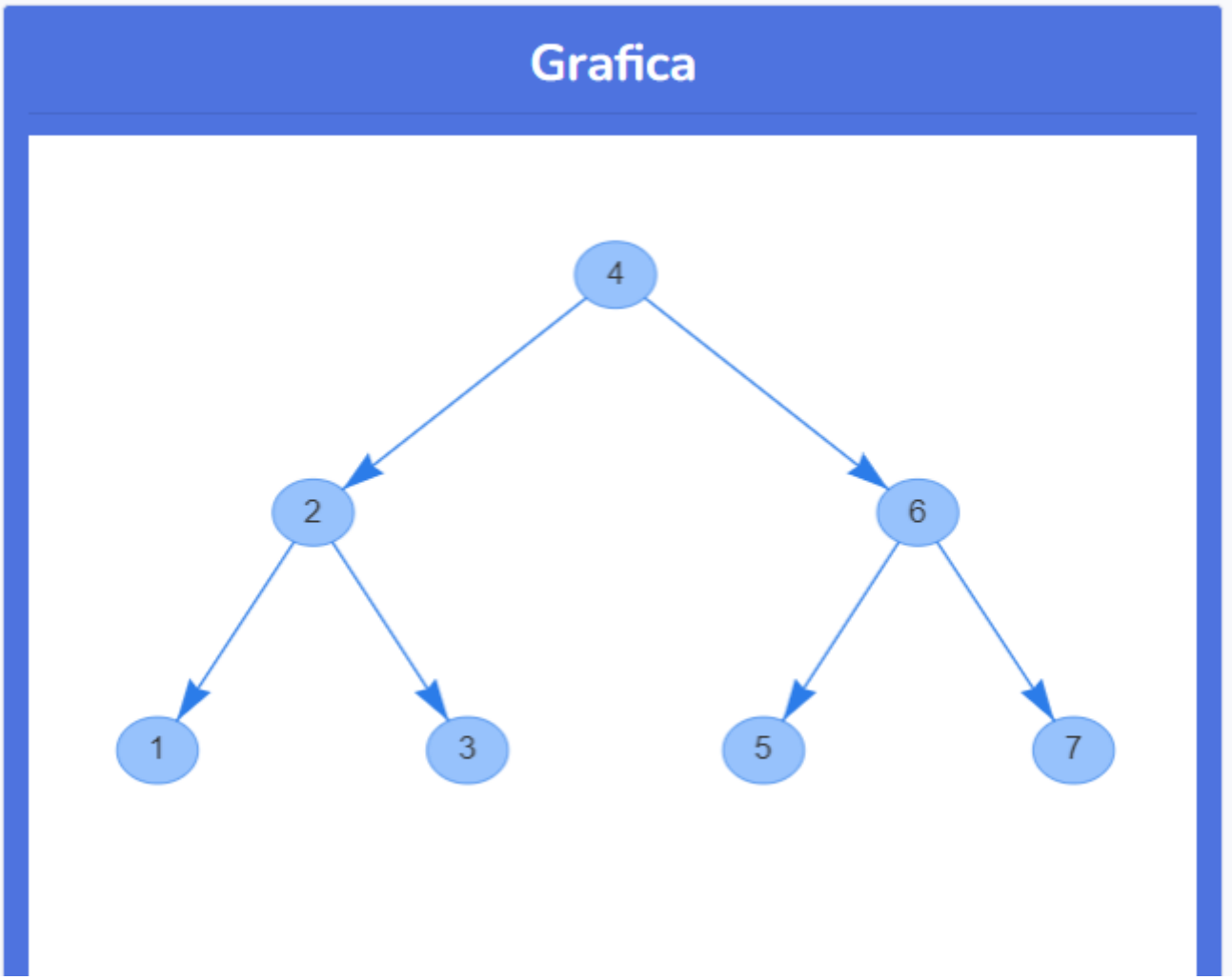
```
color:#C82807 " type="button" class="btn btn-outline-dark"><i class="fas fa-trash"></i></button> </td>'
            this.tab += '</tr>'
            this.Tabla(raizA.der);
        }
    }
```

Obteniendo una tabla como la que se muestra a continuacion:

Tambien se implemento un metodo para graficar, con el cual se genera un String que contiene lenguajen DOT para que sea mostrado en la aplicación utilizando viz-network, como se muestra a continuacion:

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

        return cadena
    }

    enlazar(raizA){
        let cadena="";
        if(raizA != null){
            cadena += this.enlazar(raizA.izq);
            cadena += this.enlazar(raizA.der);
            //validaciones
            if(raizA.izq != null){
                cadena+=raizA.id + "-> "+raizA.izq.id+"\n";
            }
            if(raizA.der != null){
                cadena+=raizA.id + "-> "+raizA.der.id+"\n";
            }


        }
        return cadena;
    }
```

Obteniendo una grafica como la siguiente:



Clientes:

Para la implementacion de los Clientes se utilizo una lista doblemente enlazada, la que esta contenida en cada nodo Vendedor que componen el arbol AVL (cada vendedor tiene su lista de clientes) el cual contiene los datos:

- Id
- Nombre
- Correo

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

```
class nodoL{
    constructor(idV,idC,nombre,correo){
        this.idC = idC;
        this.idV = idV;
        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 metodo 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”, refiriendose al nodo que se almacena a continuacion, y un atributo “anterior”, refiriendose al nodo que se almacena a antes, 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 continuacion:

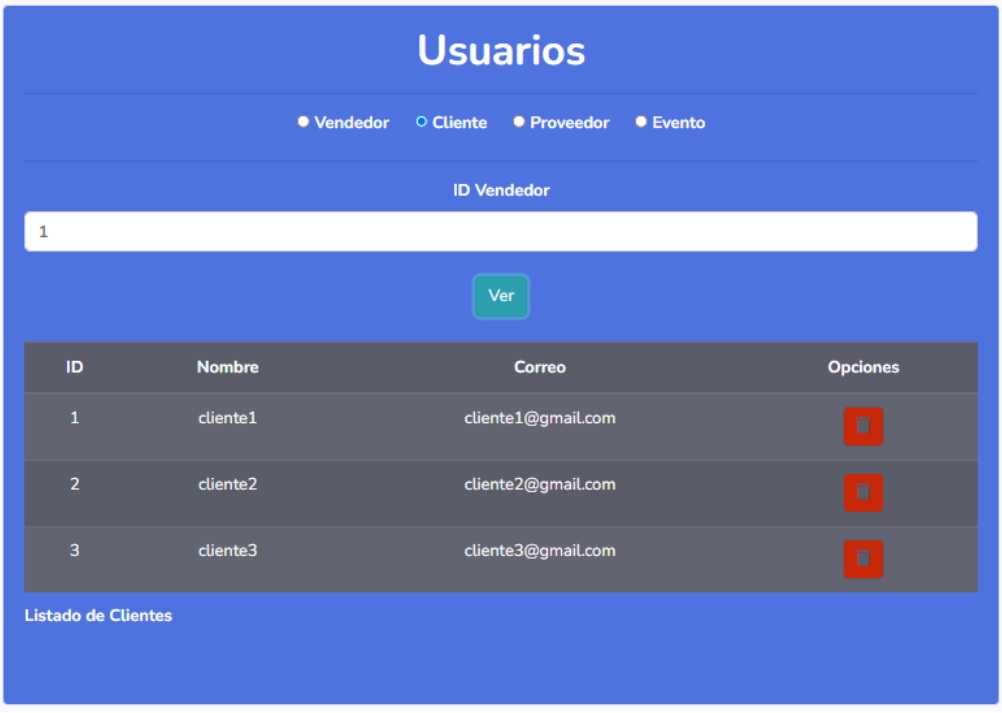
```
class listaD{
    constructor(){
        this.primero = null;
    }

    insertar(idV,idC,nombre,correo){
        let nuevo = new nodoL(idV,idC,nombre,correo);

        if(this.primero == null){
            this.primero = nuevo;
```

```
        }else{
            let aux = this.primero;
            let exists=false
            while(aux.siguiente != null) {
                if (idC==aux.idC){
                    exists = true
                }
                aux = aux.siguiente
            }
            if (idC==aux.idC){
                exists = true
            }

            if (!exists){
                aux.siguiente = nuevo;
                nuevo.anterior = aux;
                console.log("Ingresado")

                console.log(aux.siguiente)
                console.log(nuevo.anterior)
            }else{
                console.log("El cliente con ID "+idC+" ya existe en la
cartera de clientes de: "+idV)
                alert("El cliente con ID "+idC+" ya existe en la cartera de
clientes de: "+idV)
            }
            aux = this.primero;
            console.log("Recorriendo")
            while(aux != null) {
                console.log(aux.siguiente)
                console.log(aux.anterior)
                aux = aux.siguiente
            }
        }
    }
```

Se creo un metodo para recorrer y mostrar por medio de una tabla los clientes que possee el vendedor, tal y como se muestra a continuacion:

```
tabla(id){
        let aux = this.primero;
        //console.log("********** Mostar Lista de "+id+" **********")
        let list=""
        while(aux != null){

            list += '<tr>'
            list += '<td>'+aux.idC+'</td>'
            list += '<td>'+aux.nombre+'</td>'
            list += '<td>'+aux.correo+'</td>'

            list += '<td>'
            list += '<button value=\"'+aux.idC+'\"
onclick="eliminarC(this.value);VerA(\'Cliente\');" style="background-
color:#C82807 " type="button" class="btn btn-outline-dark"><i class="fas fa-
trash"></i></button> </td>'
            list += '</tr>'

            aux = aux.siguiente;
        }
        return list
    }
```

Obteniendo una tabla como la siguiente:



Se creo un metodo eliminar para remover clientes de la lista de un vendedor, según los casos en los cuales se encuentra el nodo que se desea eliminar, tal y como se muestra a continuacion:

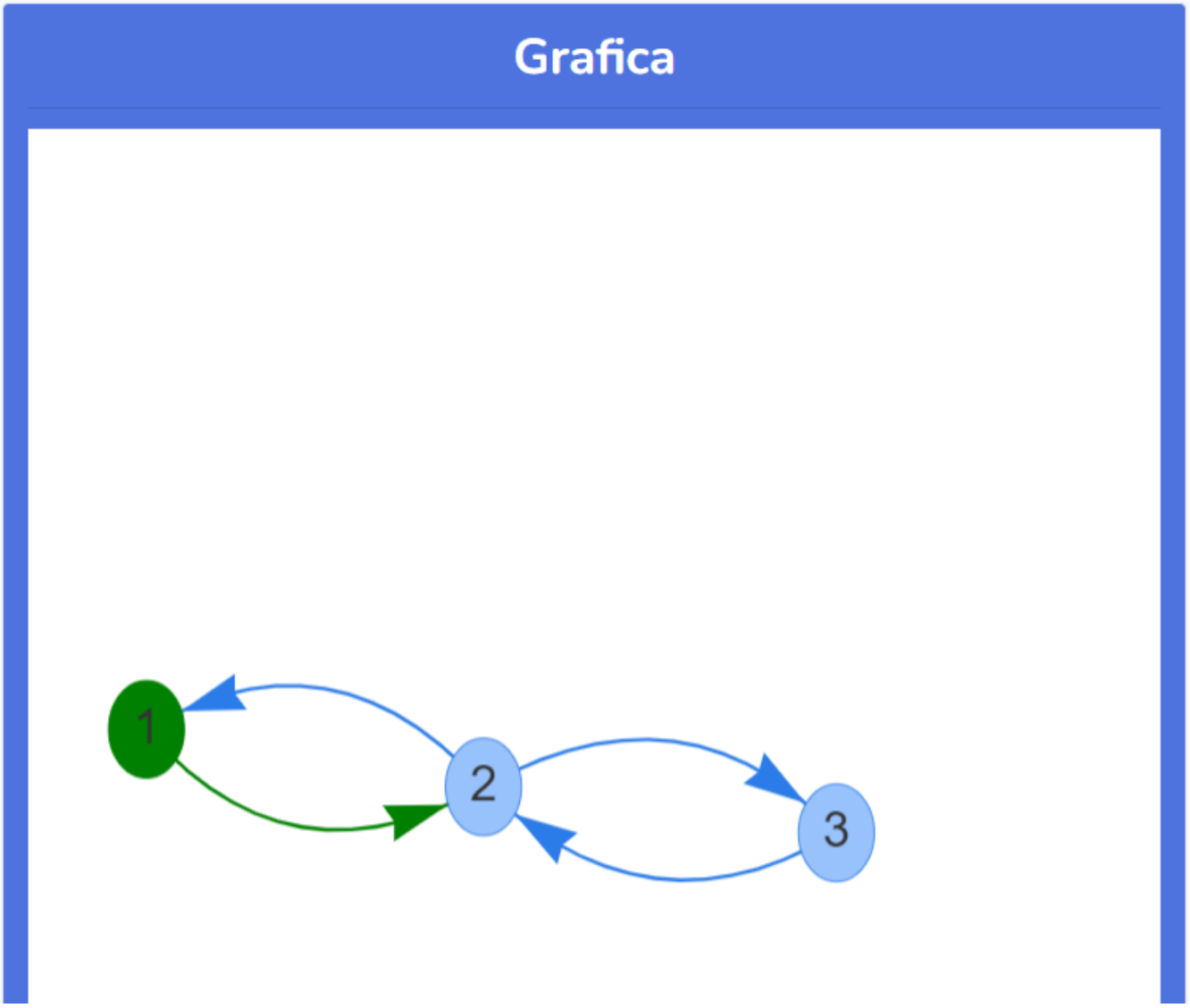
```
eliminar(valor){
        let aux = this.primero;
        let ant = null
        while(aux != null){
            if (valor==aux.idC){
                if (this.primero.idC==aux.idC){ //Eliminar primero
                    console.log("Primero")
                    aux.siguiente.anterior=null
                    this.primero=aux.siguiente
                }else if (aux.siguiente==null){ //Eliminar ultimo
                    console.log("Ultimo")
                    ant.siguiente=null
                }else{  //Eliminar enmedio
                    ant.siguiente=aux.siguiente
                    aux.siguiente.anterior=ant
                }
                break
            }
            ant = aux
            aux = aux.siguiente;
        }
    }
```

Tambien se implemento un metodo para graficar, con el cual se genera un String que contiene lenguaje DOT para que sea mostrado en la aplicación utilizando viz-network, como se muestra a continuacion:

```
generar(){
        let aux = this.primero;

        let list=""
        let l = true
        let dot='\n'+this.primero.idC +' [color = green];\n'
        while(aux != null){
            if (aux.siguiente!=null){
                dot +=aux.idC +'->'+aux.siguiente.idC+';\n'
                dot +=aux.siguiente.idC +'->'+aux.idC+';\n'
            }
```

```
            //alert(aux.anterior)
            aux = aux.siguiente;
        }
        return dot
    }
```

Obteniendo una grafica como la siguiente:



Meses:

Para la implementacion de los Eventos primero se utilizo una lista doblemente enlazada, la que esta contenida en cada nodo Vendedor que componen el arbol AVL (cada vendedor tiene su lista de meses) el cual contiene el mes en que se realiza el evento, este necesita los siguientes datos:

- Mes
- Calendario

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

```
class nodoM{
    constructor(mes){
        this.mes = mes;
        this.calendario = new matriz();
        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 metodo 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”, refiriendose al nodo que se almacena a continuacion, y un atributo “anterior”, refiriendose al nodo que se almacena a antes, 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 continuacion:

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

    insertar(mes){
        let nuevo = new nodoM(mes);

        if(this.primero == null){
            this.primero = nuevo;
        }else{
            let aux = this.primero;
```

```
            let exists=false
            while(aux.siguiente != null) {
                if (mes==aux.mes){
                    exists = true
                }
                aux = aux.siguiente
            }
            if (mes==aux.mes){
                exists = true
            }
            if (!exists){
                aux.siguiente = nuevo;
                nuevo.anterior = aux;
            }else{
                console.log("El mes "+mes+" ya existe ")
                //alert("El mes "+idC+" ya existe")
            }
        }
    }
```

Evento:

Para la implementacion de los Eventos luego de ingresar el mes en la lista si este no existe, dentro de cada nodo se crea o se se utiliza una matriz dinamica en la cual se almacenaran los datos restantes para componer el evento, los cuales son:

- Día
- Hora
- Descripcion

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

```
class nodo_interno{
    constructor(valor,x,y){
        this.valor = valor;
        this.x = x;
        this.y = y;
        //apuntadores
        this.sig = null;
        this.ant = null;
        this.arriba = null;
        this.abajo = null;
    }
}
```

Luego se comenzó a construir la matriz dinamica utilizando este nodo, por lo cual fue necesario el uso de un metodo insertar en el cual se comienzan a crear nodos cabecera, los cuales son necesarios para pocicionar el evento en el dia y hora que se solicita relacionandolas y agregandole la descripcion (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 metodo de insersion se relaciona con la insercion a estas otras estructuras como se puede apreciar en el siguiente codigo:

```
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);
    }
```

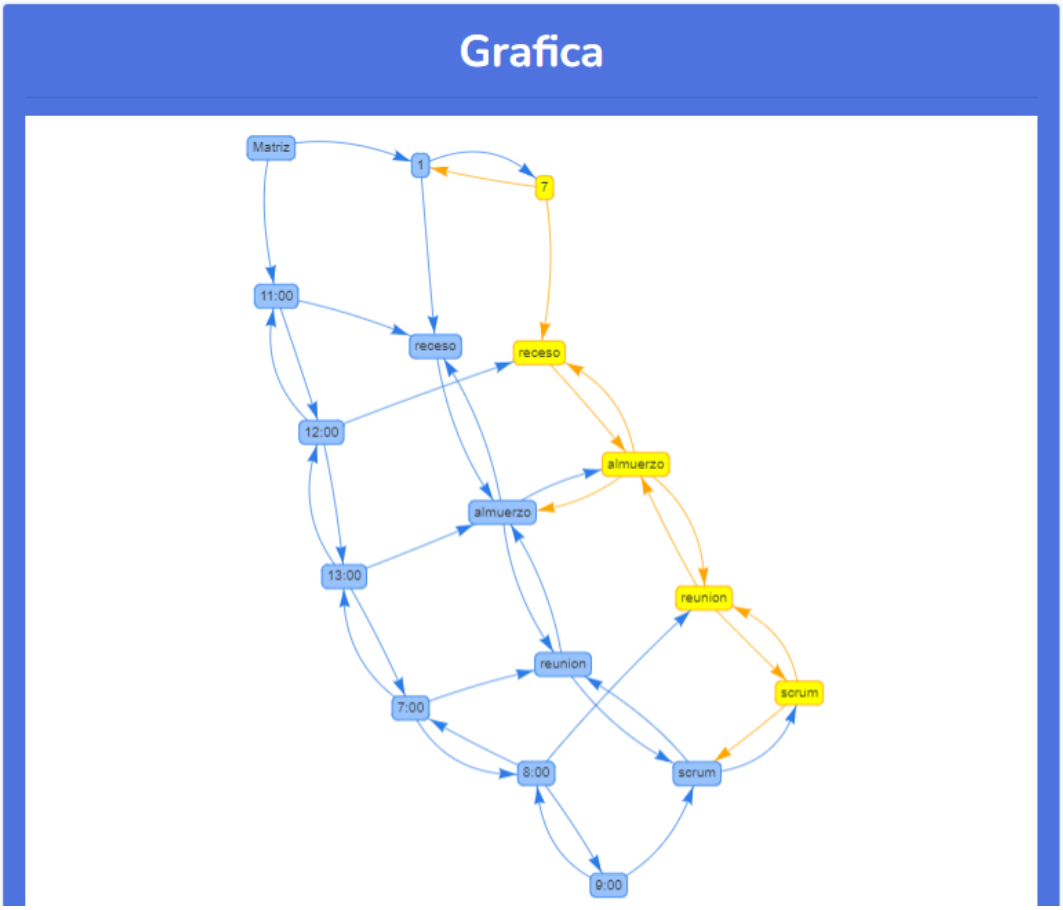
Tambien se implemento un metodo para graficar, con el cual se genera un String que contiene lenguaje DOT para que sea mostrado en la aplicación utilizando viz-network, como se muestra a continuacion:

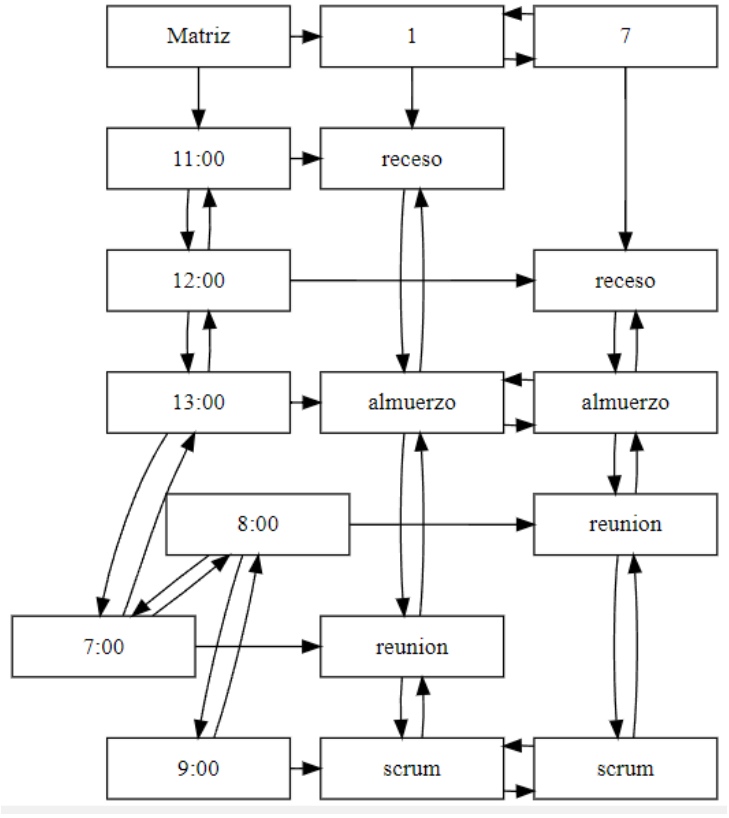
```
graficar_matriz(){
        let cadena="";
        cadena+= 'digraph Matriz{ \n';
        cadena+= 'node [shape=box];\n';
        //Nodo matriz
        cadena+='Mt[ label = "Matriz", width = 1.5, group = 1 ];\n'
        //Y
        let aux_y = this.cabecetas_y.primero;
        while(aux_y!=null){
            cadena+='U'+aux_y.dato.replace(':',"")+' [label =
"'+aux_y.dato+'"    width = 1.5  group = 1 ];\n'
            aux_y = aux_y.sig;
        }
        aux_y = this.cabecetas_y.primero;
        while(aux_y.sig != null){
            cadena+='U'+aux_y.dato.replace(':',"")+'-
>'+'U'+aux_y.sig.dato.replace(':',"")+';\n'
            cadena+='U'+aux_y.sig.dato.replace(':',"")+'-
>'+'U'+aux_y.dato.replace(':',"")+';\n'
            aux_y = aux_y.sig;
        }
```

```
        if(this.cabecetas_x.primero!= null){
            cadena+='Mt-
>U'+this.cabecetas_y.primero.dato.replace(":","")+';\n';
        }
        //X
        let aux_x = this.cabecetas_x.primero;
        while(aux_x!=null){
            cadena+='A'+aux_x.dato+' [label = \"'+aux_x.dato+'\"   width =
1.5  group = '+aux_x.dato+' ];\n'
            aux_x = aux_x.sig;
        }
        aux_x = this.cabecetas_x.primero;
        while(aux_x.sig != null){
            cadena+='A'+aux_x.dato+'->'+'A'+aux_x.sig.dato+';\n'
            cadena+='A'+aux_x.sig.dato+'->'+'A'+aux_x.dato+';\n'
            aux_x = aux_x.sig;
        }
        if(this.cabecetas_x.primero!= null){
            cadena+='Mt->A'+this.cabecetas_x.primero.dato+';\n';
        }
        cadena+='{ rank = same; Mt;'
        aux_x = this.cabecetas_x.primero;
        while(aux_x != null){
            cadena+='A'+aux_x.dato+'; '
            aux_x = aux_x.sig;
        }
        cadena+='}\n'
        aux_x = this.cabecetas_x.primero;
        while(aux_x!=null){
            let aux = aux_x.lista_interna.primero;
            while(aux!=null){
                cadena+='A'+aux.x+'_U'+aux.y.replace(':',"")+' [label =
"'+aux.valor+'" width = 1.5, group = '+aux.x+' ];\n'
                aux = aux.sig;
            }
            aux = aux_x.lista_interna.primero;
            while(aux.sig!= null){
                cadena+='A'+aux.x+'_U'+aux.y.replace(':',"")+'-
>A'+aux.sig.x+'_U'+aux.sig.y.replace(':',"")+';\n';
                cadena+='A'+aux.sig.x+'_U'+aux.sig.y.replace(':',"")+'-
>A'+aux.x+'_U'+aux.y.replace(':',"")+';\n';
                aux= aux.sig;
```

```
            }
            if(aux_x.lista_interna.primero!= null){
                cadena+='A'+aux_x.dato+'-
>'+'A'+aux_x.lista_interna.primero.x+'_U'+aux_x.lista_interna.primero.y.repl
ace(':',"")+';\n';
            }
            aux_x = aux_x.sig;
        }
        aux_y = this.cabecetas_y.primero;
        while(aux_y!=null){
            let aux = aux_y.lista_interna.primero;
            let nodos = ''
            let nfila = ''
            while(aux.abajo!= null){
                cadena+='A'+aux.x+'_U'+aux.y.replace(':',"")+'-
>A'+aux.abajo.x+'_U'+aux.abajo.y.replace(':',"")+';\n';
                cadena+='A'+aux.abajo.x+'_U'+aux.abajo.y.replace(':',"")+'-
>A'+aux.x+'_U'+aux.y.replace(':',"")+';\n';
                nodos+='A'+aux.x+'_U'+aux.y.replace(':','')+';'
                aux= aux.abajo;
            }
            if(aux_y.lista_interna.primero!= null){
                nfila = 'U'+aux_y.dato.replace(':',"")+';'
                nodos+='A'+aux.x+'_U'+aux.y.replace(':','')+'; '
                cadena+='U'+aux_y.dato.replace(':',"")+'-
>'+'A'+aux_y.lista_interna.primero.x+'_U'+aux_y.lista_interna.primero.y.repl
ace(':',"")+';\n';
            }
            aux_y = aux_y.sig;
            cadena+='{ rank = same; '+nfila+' '+nodos+'}\n'
        }
        cadena+="}"
        //console.log(cadena);
        return cadena
    }
```

Generando una grafica como la siguiente, en la cual se pueden apreciar las conexiones pero no se ve tan estetica como se desearia por lo que tambien se muestra el codigo DOT en HTML para colocarlo en un editor y verlo mejor:





Proveedores:

Para la implementacion de los Proveedores se utilizo un arbol ABB, el cual contiene los datos:

- Id
- Nombre
- Direccion
- Telefono
- Correo

Por lo cual se realizo 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.izq = null;
        this.der = null;
    }
}
```

Luego se comenzó a construir el arbol ABB utilizando este nodo, por lo cual fue necesario el uso de un metodo insertar y en el constructor de la clase un atributo llamado *“raiz”* para guardar el primer objeto y no perderlo en la memoria, ya que este nodo contiene un atributo *“izq”,* refiriendose al nodo que se almacena a la izquierda, y un atributo *“der”,* refiriendose 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 continuacion:

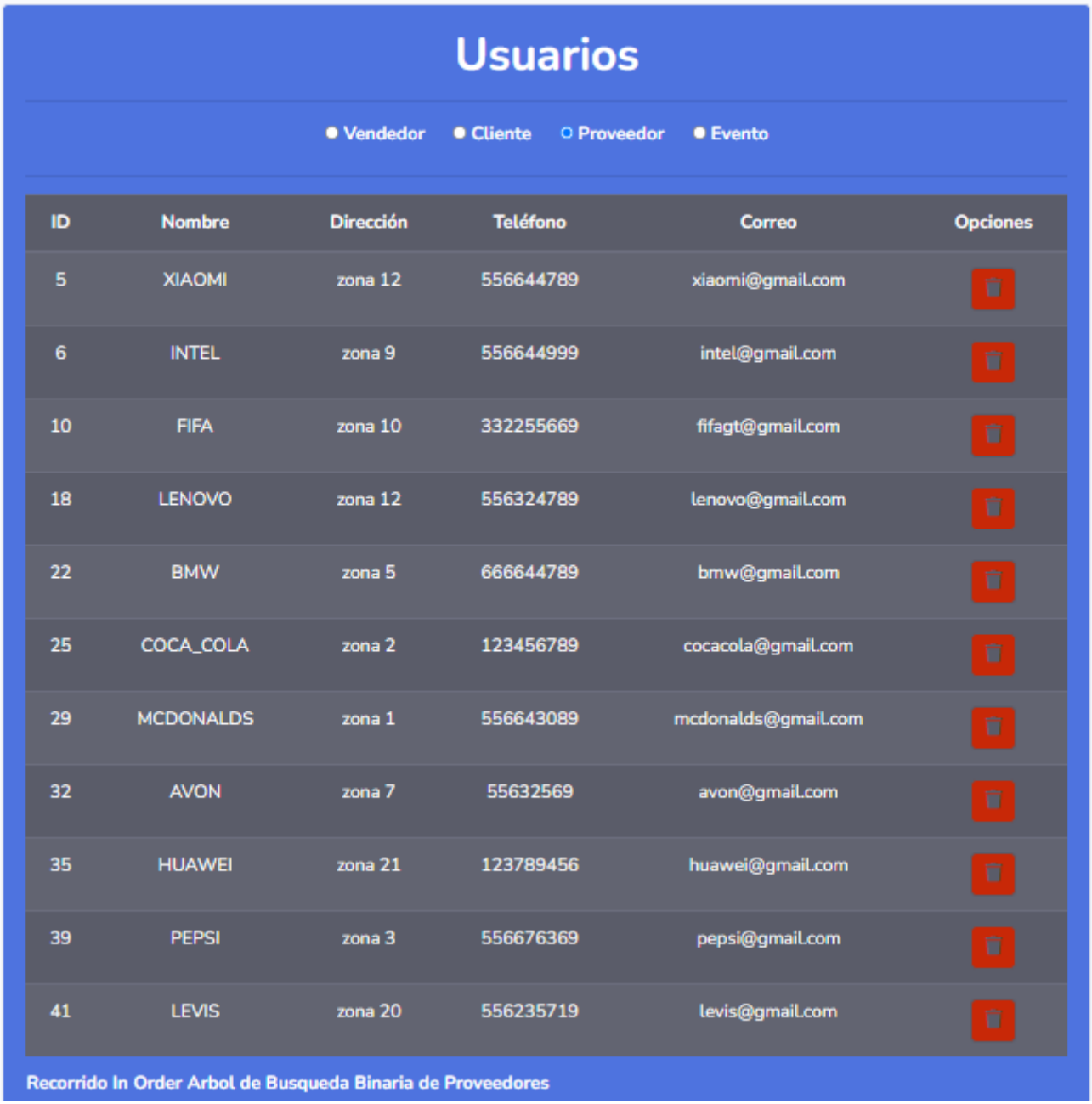
```
class abb{
    constructor(){
        this.listaa = null
        this.raiz = null;
        this.dot = '';
        this.busc=''
        this.tab=''
        this.nodoB=''
        //Object.assign(this.insertar,this.insertar_nodo,this.PreOrder,this.
generar,this.buscar)
```

```
        console.log("Inicializao")
    }

    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.insertar_nodo(this.raiz,nuevo);


        }
    }

    insertar_nodo(raiz_actual,nuevo){
        if(raiz_actual != null){

            if(raiz_actual.id > nuevo.id){
                raiz_actual.izq = this.insertar_nodo(raiz_actual.izq,nuevo);
            }else if(raiz_actual.id < nuevo.id){
                raiz_actual.der = this.insertar_nodo(raiz_actual.der,nuevo);
            }else{
                console.log("Ya existe un Proveedor con ID = "+nuevo.id);
                alert("Ya existe un Proveedor con ID = "+nuevo.id);
            }

            return raiz_actual;
        }else{
            raiz_actual = nuevo;
            //alert("Se ha ingresado el Proveedor con ID = "+nuevo.id);
            return raiz_actual;
        }
    }
```

Tambien para mostrar los datos se realizaron metodos de recorrido del arbol como lo es el PreOrden (para pruebas) y el InOrden para mostrar los datos en los reportes generados con el fin de que estos se muestren en orden ascendente de ID, tal y como se muestra a continuacion:

```
PreOrder(raizA){
        if (raizA!=null){
            console.log(raizA.id);
            this.PreOrder(raizA.izq);
            this.PreOrder(raizA.der);
        }
    }
Tabla(raizA){
        if (raizA!=null){
            this.Tabla(raizA.izq);
            this.tab += '<tr>'
            this.tab += '<td>'+raizA.id+'</td>'
            this.tab += '<td>'+raizA.nombre+'</td>'
            this.tab += '<td>'+raizA.direccion+'</td>'
            this.tab += '<td>'+raizA.telefono+'</td>'
            this.tab += '<td>'+raizA.correo+'</td>'
            this.tab += '<td>'
            this.tab += '<button value=\"'+raizA.id+'\"
onclick="eliminar(this.value);VerA(\'Proveedor\');" style="background-
color:#C82807 " type="button" class="btn btn-outline-dark"><i class="fas fa-
trash"></i></button> </td>'
            this.tab += '</tr>'
            this.Tabla(raizA.der);
        }
    }
```

Obteniendo una tabla como la que se muestra a continuacion:



Usuarios

● Vendedor ● Cliente ○ Proveedor ● Evento

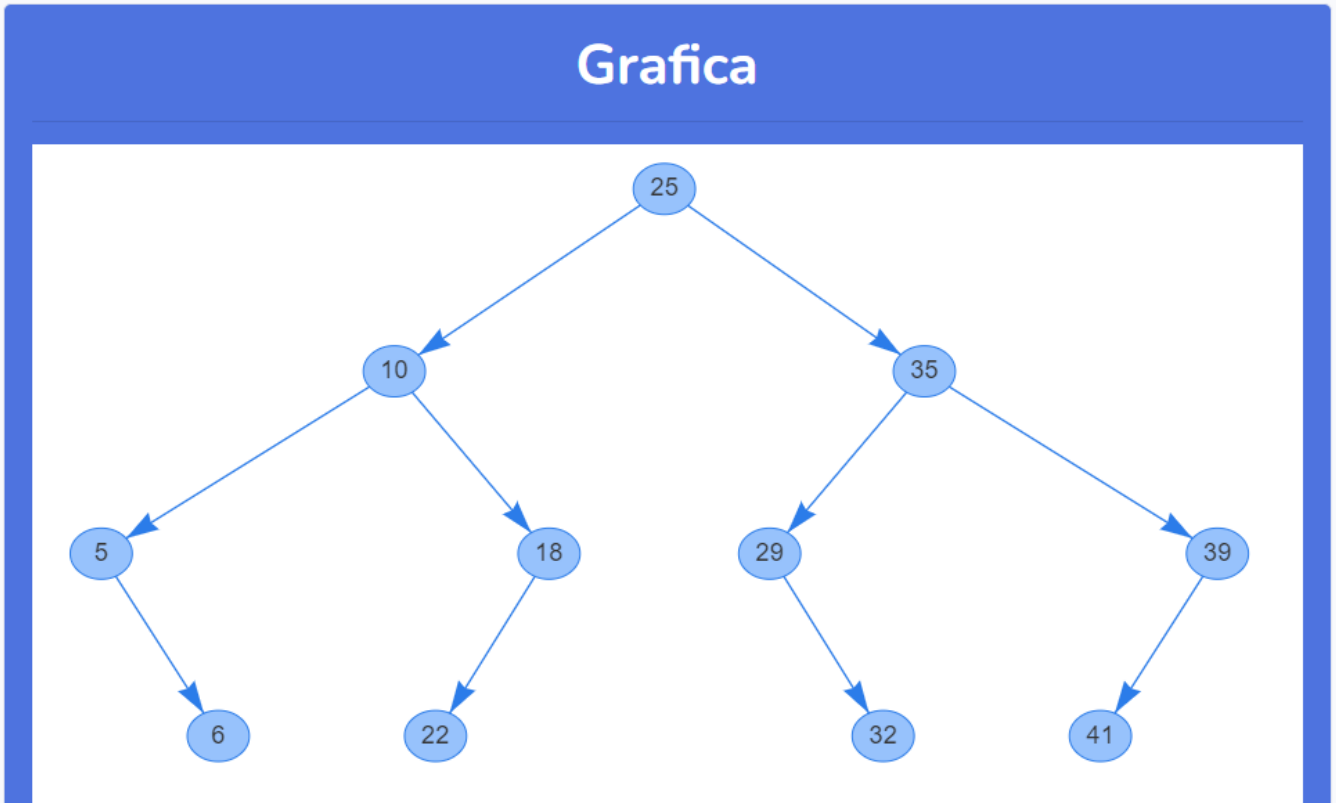
ID	Nombre	Dirección	Teléfono	Correo	Opciones
5	XIAOMI	zona 12	556644789	xiaomi@gmail.com	
6	INTEL	zona 9	556644999	intel@gmail.com	
10	FIFA	zona 10	332255669	fifagt@gmail.com	
18	LENOVO	zona 12	556324789	lenovo@gmail.com	
22	BMW	zona 5	666644789	bmw@gmail.com	
25	COCA_COLA	zona 2	123456789	cocacola@gmail.com	
29	MCDONALDS	zona 1	556643089	mcdonalds@gmail.com	
32	AVON	zona 7	55632569	avon@gmail.com	
35	HUAWEI	zona 21	123789456	huawei@gmail.com	
39	PEPSI	zona 3	556676369	pepsi@gmail.com	
41	LEVIS	zona 20	556235719	levis@gmail.com	

Recorrido In Order Arbol de Busqueda Binaria de Proveedores

Tambien se implemento un metodo para graficar, con el cual se genera un String que contiene lenguajen DOT para que sea mostrado en la aplicación utilizando viz-network, como se muestra a continuacion:

```
generar(raizA){
        if (raizA!=null){
            try{
                this.dot +=raizA.id +'->'+raizA.izq.id+';'
            }catch(error){

            }
            try{
                this.dot +=raizA.id +'->'+raizA.der.id+';'
            }catch(error){

            }

            this.generar(raizA.izq);
            this.generar(raizA.der);
        }
    }
```

Obteniendo una grafica como la siguiente:



Conclusiones

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