Nueva Guatemala de la Asunción

Reporte de Fase 2

Proyecto EDD

Elaborado por: Estuardo Gabriel Son Mux
Carné: 202003894
Fecha: 03/01/2022

Alcances y Objetivos del Programa

El programa realizado tiene como objetivo la implementación de los conocimientos adquiridos durante las primeras semanas del curso de Estructuras de Datos, realizando una página web en la que se incluyeron las estructuras: Tabla Hash, Árbol B, Grafos y BlockChain, sin embargo, debido a cuestiones de tiempo, no se pudo realizar la implementación de la estructura BlockChain por lo que no sé explicara en este documento.

Especificaciones Técnicas

Requisitos de Hardware

- Computadora con todos sus componentes para su correcto funcionamiento
- Laptop

Requisitos de Software

Sistema Operativo con el que se Realizo

El programa fue desarrollado en una laptop con sistema operativo Windows 10, 8GB de ram, Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz.

Nombre del dispositivo	DESKTOP-LJSJP7U
Procesador	Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz
RAM instalada	8.00 GB
Id. del dispositivo	515F0098-1032-4937-BDB1-C191D818D118

Lenguaje de Programación

JavaScript

Librerías

- Vis-Network
- CirculaJSON
- SJCL

IDE o Editor de Código

Visual Estudio

Lógica del Programa

Archivos Utilizados

La página web consta de varios archivos de los cuales se dividen entre el lenguaje de programación JavaScript y el lenguaje de etiquetado HTML. Sin embargo, solo se hará mención de las estructuras utilizadas para la elaboración del proyecto que se encuentran en los archivos JS.

Archivos JavaScricpt (.js)

ArbolB.js

En este archivo se encuentran las clases utilizadas para la creación del árbol B para el registro de productos.

Clase Producto

```
class Producto{
    constructor(id, nombre, precio, cantidad){
        this.id = id
        this.nombre = nombre
        this.precio = precio
        this.cantidad = cantidad
    }
}
```

La primera clase que se encuentra en el archivo Arbol.js es Producto con la cual se generan los objetos Producto, cuyos atributos son:

- **Id**
- **Nombre**
- **Precio**
- **Cantidad**

Clase Pagina

```
class Pagina{
    constructor(){
        this.lista = new lista()
    }
}
```

Esta clase se utiliza para crear las paginas que almacenaran el listado de nodos que conformaran el árbol B.

Clase lista

```
class lista{
    constructor(){
        this.primero = null;
        this.ultimo = null;
        this.longitud = 0;
    }

    insertarEnPagina(id, nombre, precio, cantidad){···
    }
}
```

Clase que se utiliza para la creación de listas con las cuales almacenar cada nodo que formara parte de una página. Sus atributos son:

- **Primero:** Es el primer nodo que forma la lista.
- **Ultimo:** Es el ultimo nodo de la lista.
- **Longitud:** Numero de nodos que conforman la lista.

Método insertarEnPagina(id, nombre, precio, cantidad)

```
insertarEnPagina(id, nombre, precio, cantidad){
    let nuevo = new nodoB(id, nombre, precio, cantidad);
    if (this.primero == null){
        this.primero=this.ultimo=nuevo;
        this.longitud += 1
    }
    else if (this.primero.producto.id < nuevo.producto.id && this.primero.siguiente == null){
        nuevo.anterior = this.primero;
        this.primero.siguiente = nuevo;
        this.ultimo = nuevo;
        this.longitud += 1
    }else if (this.primero.producto.id > nuevo.producto.id){
        this.primero.anterior = nuevo;
        nuevo.siguiente = this.primero;
        this.primero = nuevo;
        this.longitud += 1
    }else{
        let aux = this.primero;
        while(aux != null){
            if (aux == this.ultimo && aux.producto.id < nuevo.producto.id){
                nuevo.anterior = aux;
                aux.siguiente = nuevo;
                this.ultimo = nuevo;
                this.longitud += 1
                break;
            }
            else if (aux != this.ultimo && aux.siguiente.producto.id > nuevo.producto.id && aux.producto.id < nuevo.producto.id){
                nuevo.siguiente = aux.siguiente;
                nuevo.anterior = aux;
                nuevo.siguiente.anterior = nuevo;
                aux.siguiente = nuevo;
                this.longitud += 1
                break;
            }

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

Función encargada de ingresar nodos en la lista, los cuales contienen objetos de tipos producto con los parámetros recibidos, la cual ingresa los nodos de forma ascendente.

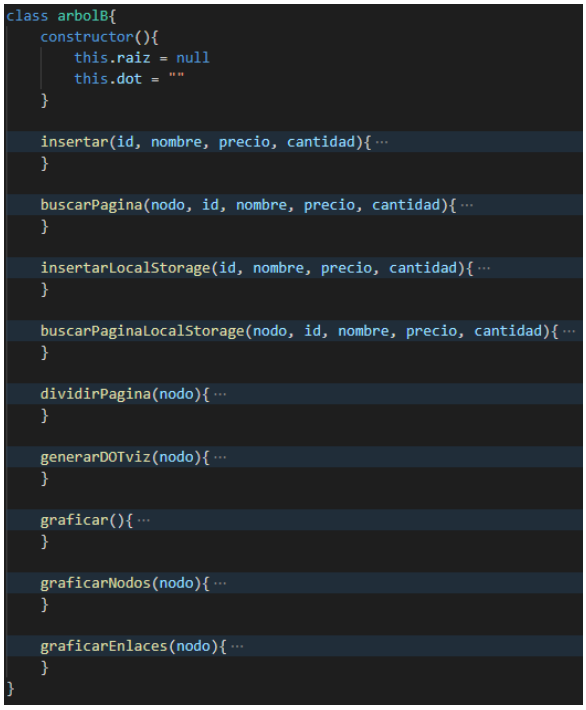
Clase nodoB

```
class nodoB{
    constructor(id, nombre, precio, cantidad){
        this.producto = new Producto(id, nombre, precio, cantidad)
        this.siguiente = null
        this.anterior = null
        this.derecha = null
        this.izquierda = null
    }
}
```

Clase que se utiliza para la creación de los nodos del árbol B. Sus atributos son:

- **Producto:** Guarda el objeto de tipo Producto que contendrá el nodo.
- **Siguiente:** Apuntador al nodo siguiente de la misma página.
- **Anterior:** Apuntador al nodo predecesor de la misma página.
- **Derecha:** Apuntador a la página derecha del nodo actual.
- **Izquierda:** Apuntador a la página izquierda del nodo actual.

Clase ArbolB



Clase para la inicialización del árbol B. Posee los atributos:

- **Raíz:** Apuntador a la página principal del árbol B.

- **Dot:** Variable para almacenar un texto con formato DOT para graphviz.

Esta clase posee 9 funciones o métodos para su funcionamiento:

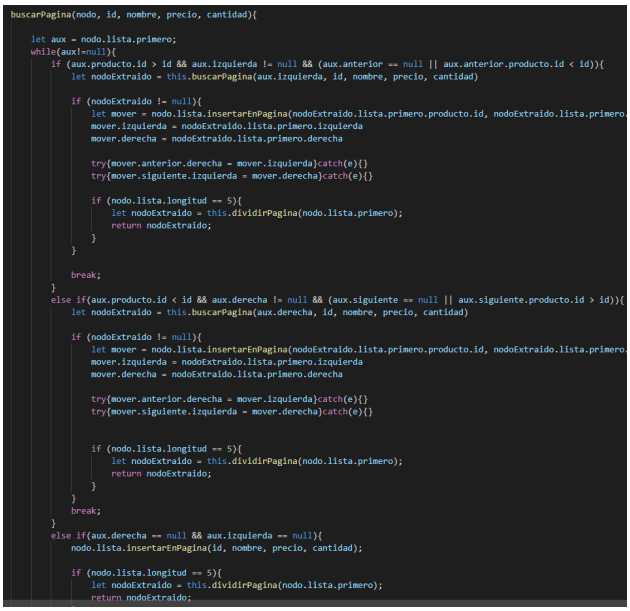
Insertar(id, nombre, precio, cantidad) y insertarLocalStorage(id, nombre, precio, cantidad)

```
insertar(id, nombre, precio, cantidad){
    if (this.raiz == null){
        this.raiz = new Pagina();
        this.raiz.lista.insertarEnPagina(id, nombre, precio, cantidad);
    }
    else if (this.raiz.lista.primero.izquierda == null && this.raiz.lista.primero.derecha == null){
        this.raiz.lista.insertarEnPagina(id, nombre, precio, cantidad);

        if (this.raiz.lista.longitud == 5){
            this.raiz = this.dividirPagina(this.raiz.lista.primero)
        }
    }
    else{
        let cambio = this.buscarPagina(this.raiz, id, nombre, precio, cantidad)

        if (cambio != null){
            this.raiz = cambio
        }
    }
}
```

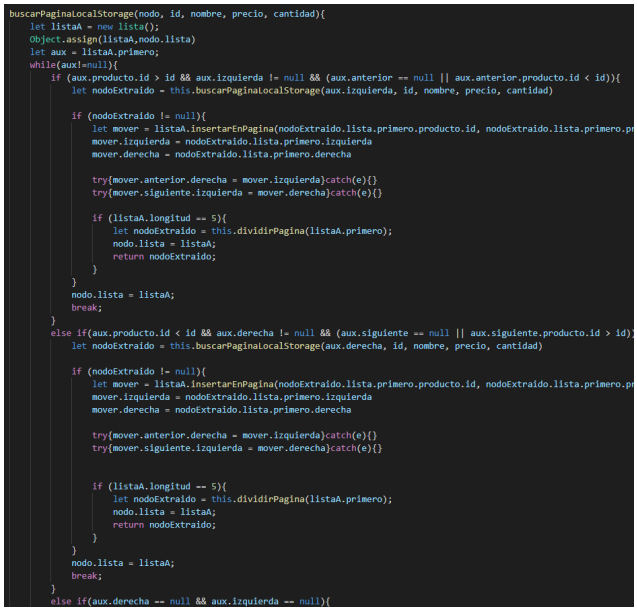
```
insertarLocalStorage(id, nombre, precio, cantidad){
    if (this.raiz == null){
        this.raiz = new Pagina();
        let listaA = new lista();
        Object.assign(listaA,this.raiz.lista)
        listaA.insertarEnPagina(id, nombre, precio, cantidad);
        this.raiz.lista = listaA;
    }
    else if (this.raiz.lista.primero.izquierda == null && this.raiz.lista.primero.derecha == null){
        let raizAux = new Pagina();
        Object.assign(raizAux,this.raiz)
        let listaA = new lista();
        Object.assign(listaA,raizAux.lista)
        listaA.insertarEnPagina(id, nombre, precio, cantidad);
        raizAux.lista = listaA;

        if (raizAux.lista.longitud == 5){
            raizAux = this.dividirPagina(raizAux.lista.primero)
        }

        this.raiz = raizAux;
    }
    else{
        let raizAux = new Pagina();
        Object.assign(raizAux,this.raiz)
        let cambio = this.buscarPaginaLocalStorage(raizAux, id, nombre, precio, cantidad)

        if (cambio != null){
            raizAux = cambio
        }
        this.raiz = raizAux
    }
}
```

Métodos que se utilizan para la inserción de nodos dentro de la página principal o página raíz la cual realizando sus respectivos ajustes en caso de que dicha página alcance una longitud de 5.

BuscarPagina(nodo, id , nombre, precio, cantidad) y buscarPaginaLocalStorage(nodo, id , nombre, precio, cantidad)



```
buscarPagina(nodo, id, nombre, precio, cantidad){

    let aux = nodo.lista.primero;
    while(aux!=null){
        if (aux.producto.id > id && aux.izquierda != null && (aux.anterior == null || aux.anterior.producto.id < id)){
            let nodoExtraido = this.buscarPagina(aux.izquierda, id, nombre, precio, cantidad)

            if (nodoExtraido != null){
                let mover = nodo.lista.insertarEnPagina(nodoExtraido.lista.primero.producto.id, nodoExtraido.lista.primero.
                mover.izquierda = nodoExtraido.lista.primero.izquierda
                mover.derecha = nodoExtraido.lista.primero.derecha

                try{mover.anterior.derecha = mover.izquierda}catch(e){}
                try{mover.siguiente.izquierda = mover.derecha}catch(e){}

                if (nodo.lista.longitud == 5){
                    let nodoExtraido = this.dividirPagina(nodo.lista.primero);
                    return nodoExtraido;
                }
            }

            break;
        }
        else if(aux.producto.id < id && aux.derecha != null && (aux.siguiente == null || aux.siguiente.producto.id > id)){
            let nodoExtraido = this.buscarPagina(aux.derecha, id, nombre, precio, cantidad)

            if (nodoExtraido != null){
                let mover = nodo.lista.insertarEnPagina(nodoExtraido.lista.primero.producto.id, nodoExtraido.lista.primero.
                mover.izquierda = nodoExtraido.lista.primero.izquierda
                mover.derecha = nodoExtraido.lista.primero.derecha

                try{mover.anterior.derecha = mover.izquierda}catch(e){}
                try{mover.siguiente.izquierda = mover.derecha}catch(e){}


                if (nodo.lista.longitud == 5){
                    let nodoExtraido = this.dividirPagina(nodo.lista.primero);
                    return nodoExtraido;
                }
            }
            break;
        }
        else if(aux.derecha == null && aux.izquierda == null){
            nodo.lista.insertarEnPagina(id, nombre, precio, cantidad);

            if (nodo.lista.longitud == 5){
                let nodoExtraido = this.dividirPagina(nodo.lista.primero);
                return nodoExtraido;
```



```
buscarPaginaLocalStorage(nodo, id, nombre, precio, cantidad){
    let listaA = new lista();
    Object.assign(listaA,nodo.lista)
    let aux = listaA.primero;
    while(aux!=null){
        if (aux.producto.id > id && aux.izquierda != null && (aux.anterior == null || aux.anterior.producto.id < id)){
            let nodoExtraido = this.buscarPaginaLocalStorage(aux.izquierda, id, nombre, precio, cantidad)

            if (nodoExtraido != null){
                let mover = listaA.insertarEnPagina(nodoExtraido.lista.primero.producto.id, nodoExtraido.lista.primero.pr
                mover.izquierda = nodoExtraido.lista.primero.izquierda
                mover.derecha = nodoExtraido.lista.primero.derecha

                try{mover.anterior.derecha = mover.izquierda}catch(e){}
                try{mover.siguiente.izquierda = mover.derecha}catch(e){}

                if (listaA.longitud == 5){
                    let nodoExtraido = this.dividirPagina(listaA.primero);
                    nodo.lista = listaA;
                    return nodoExtraido;
                }
            }
            nodo.lista = listaA;
            break;
        }
        else if(aux.producto.id < id && aux.derecha != null && (aux.siguiente == null || aux.siguiente.producto.id > id)
            let nodoExtraido = this.buscarPaginaLocalStorage(aux.derecha, id, nombre, precio, cantidad)

            if (nodoExtraido != null){
                let mover = listaA.insertarEnPagina(nodoExtraido.lista.primero.producto.id, nodoExtraido.lista.primero.pr
                mover.izquierda = nodoExtraido.lista.primero.izquierda
                mover.derecha = nodoExtraido.lista.primero.derecha

                try{mover.anterior.derecha = mover.izquierda}catch(e){}
                try{mover.siguiente.izquierda = mover.derecha}catch(e){}


                if (listaA.longitud == 5){
                    let nodoExtraido = this.dividirPagina(listaA.primero);
                    nodo.lista = listaA;
                    return nodoExtraido;
                }
            }
            nodo.lista = listaA;
            break;
        }
        else if(aux.derecha == null && aux.izquierda == null){
```

Métodos recursivos que se encargan de realizar la búsqueda de la página correcta en la cual realizar la inserción de un nodo, dirigiéndose al nodo izquierdo o derecho del nodo actual dependiendo de si es menor o mayor. Así mismo, hace uso de la función dividirPagina, en caso de que la página en la cual se haga la inserción tenga una longitud de 5.

dividirPagina(nodo)

```
dividirPagina(nodo){
    let aux = nodo.siguiente;
    let temp = aux.siguiente;
    aux.siguiente.anterior = null;
    aux.siguiente=null;
    temp.izquierda = new Pagina();
    temp.izquierda.lista.primero = nodo;
    temp.izquierda.lista.primero.siguiente = temp.izquierda.lista.ultimo = nodo.siguiente;
    temp.izquierda.lista.longitud=2;
    temp.derecha = new Pagina();
    temp.derecha.lista.primero = temp.siguiente;
    temp.derecha.lista.primero.siguiente = temp.derecha.lista.ultimo = temp.siguiente.siguiente;
    temp.derecha.lista.longitud=2;
    temp.siguiente.anterior = null;
    temp.siguiente = null;
    let raiz = new Pagina();
    raiz.lista.primero = raiz.lista.ultimo = temp;
    raiz.lista.longitud = 1
    return raiz;
}
```

Función que separa una página por la mitad, tomando el nodo central y colocando los nodos a su izquierda y derecha como nuevas páginas, y convierte al nodo extraído en una nueva página.

generarDOTviz(nodo)

```
generarDOTviz(nodo){
    let cadena = "";
    if(nodo!=null){
        let aux = nodo.lista.primero;
        let identificador = aux.producto.id
        cadena+= identificador+'[label="|';
        let cadenaIzq = "";
        let cadenaDer = "";
        while(aux!=null){
            cadena += aux.producto.id +' '+aux.producto.nombre+' '+aux.producto.precio+'| ';

            if(aux.izquierda != null){
                cadenaIzq += identificador+"->"+aux.izquierda.lista.primero.producto.id+";";
                cadenaIzq += this.generarDOTviz(aux.izquierda);
            }

            if(aux.derecha != null){
                cadenaDer += identificador+"->"+aux.derecha.lista.primero.producto.id+";";
                cadenaDer += this.generarDOTviz(aux.derecha);
            }
            aux = aux.siguiente;
        }
        cadena += '"];';
        cadena += cadenaIzq + cadenaDer;
    }
    return cadena;
}
```

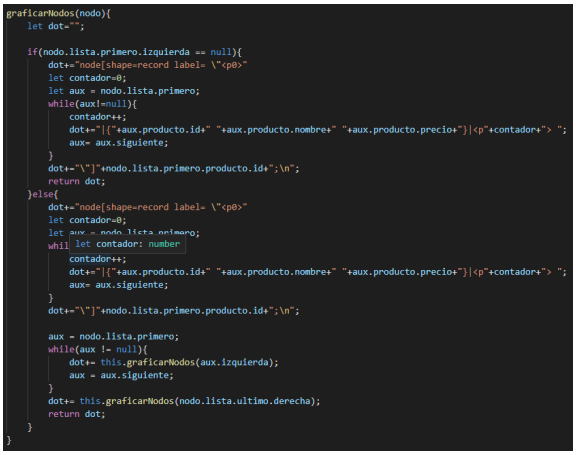
Función que recorre todo el árbol B y retorna una cadena String con formato DOT la cual contiene toda la información y enlaces de cada página contenido dentro del mismo.

Graficar()

```
graficar(){
    let dot="{\n";
    dot+="rankr=TB;\n";
    dot+='node[shape = box,fillcolor="azure2" color="black" style="filled"];\n';
    if(this.raiz!=null){
        dot+= this.graficarNodos(this.raiz);
        dot+=  this.graficarEnlaces(this.raiz);
    }
    dot+="}\n"

    return dot;
}
```

Función que retorna una cadena de texto con formato DOT para la su uso en Graphviz, recorre cada nodo que conforma el árbol con la ayuda de las funciones graficarNodos y graficarEnlaces.

GraficarNodos(nodo)



```
graficarNodos(nodo){
    let dot="";

    if(nodo.lista.primero.izquierda == null){
        dot+="node[shape=record label= \"<p0>"
        let contador=0;
        let aux = nodo.lista.primero;
        while(aux!=null){
            contador++;
            dot+="|{"+aux.producto.id+" "+aux.producto.nombre+" "+aux.producto.precio+"}|<p"+contador+"> ";
            aux= aux.siguiente;
        }
        dot+="\"]"+nodo.lista.primero.producto.id+";\n";
        return dot;
    }else{
        dot+="node[shape=record label= \"<p0>"
        let contador=0;
        let aux = nodo.lista.primero;
        whil let contador: number
            contador++;
            dot+="|{"+aux.producto.id+" "+aux.producto.nombre+" "+aux.producto.precio+"}|<p"+contador+"> ";
            aux= aux.siguiente;
        }
        dot+="\"]"+nodo.lista.primero.producto.id+";\n";

        aux = nodo.lista.primero;
        while(aux != null){
            dot+= this.graficarNodos(aux.izquierda);
            aux = aux.siguiente;
        }
        dot+= this.graficarNodos(nodo.lista.ultimo.derecha);
        return dot;
    }
}
```

Función que retorna una cadena String con los datos de los nodos del árbol en formato DOT.

GraficarEnlaces(nodo)

```
graficarEnlaces(nodo){
    let dot="";
    if(nodo.lista.primero.izquierda == null){
        return ""+nodo.lista.primero.producto.id+";\n";
    }else{

        let aux = nodo.lista.primero;
        let contador =0;
        let nodoIdentificador = nodo.lista.primero.producto.id;
        while(aux != null){
            dot+= "\n"+nodoIdentificador+":p"+contador+"->"+this.graficarEnlaces(aux.izquierda);
            contador++;
            aux = aux.siguiente;
        }
        dot+="\n"+nodoIdentificador+":p"+contador+"->"+this.graficarEnlaces(nodo.lista.ultimo.derecha);
        return dot;
    }
}
```

Función que retorna los enlaces que forma el árbol B, en una cadena String con formato DOT.

TablaHash.js

Clase listaProductos

```
class listaProductos{
    constructor(){
        this.primero = null;
        this.total = 0;
        this.ultimo = null;
    }

    insertarProducto(id, nombre, precio, cantidad){
    }
}
```

Clase que se utiliza para la creación de listados los cuales almacenaran nodos que contienen objetos productoVendido. Posee los parámetros:

- **Primero:** Apuntador al primer nodo de la lista.
- **Ultimo:** Apuntador al ultimo nodo de la lista.
- **Total:** Total de la suma de la multiplicación del precio de cada producto por la cantidad pedida.

Posee el método insertarProducto.

insertarProducto(id, nombre, precio, cantidad)

```
insertarProducto(id, nombre, precio, cantidad){
    let nuevo = new productoVendido(id, nombre, precio, cantidad);
    if (this.primero == null){
        this.primero = this.ultimo = nuevo;
        this.total += precio * cantidad;
    }
    else{
        this.ultimo.siguiente = nuevo;
        this.ultimo = nuevo;
        this.total += precio * cantidad;
    }
}
```

Función para agregar nodos que contienen objetos del tipo productoVendido a una lista.

Clase Venta

```
class Venta{
    constructor(id, vendedor, cliente, listaProductos){
        this.id = id;
        this.vendedor = vendedor;
        this.cliente = cliente;
        this.total = listaProductos.total;
        this.listaProductos = listaProductos;
    }
}
```

Clase que se utiliza para la creación objetos de tipo venta. Posee los atributos:

- **Id:** Id de la venta.
- **Vendedor:** nombre del vendedor.
- **Cliente:** nombre del cliente.
- **Total:** total de la factura.
- **listaProductos:** lista de productos de la venta.

Clase productoVendido

```
class productoVendido{
    constructor(id, nombre, precio, cantidad){
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
        this.siguiente = null;
    }
}
```

Clase para la creación de objetos del tipo productoVendido. Posee los atributos:

- **id:** id del producto.
- **Nombre:** Nombre del producto.
- **Precio:** Precio del producto.
- **Cantidad:** Cantidad solicitada del producto.
- **Siguiente:** Apuntador al siguiente producto de la lista.

Clase TablaHash

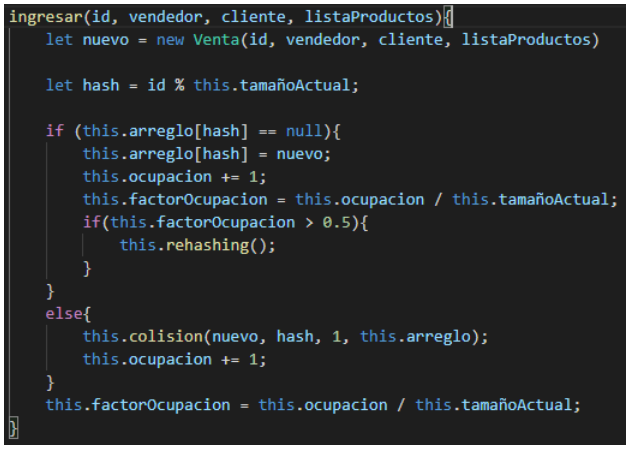
```
class TablaHash{
    constructor(){
        this.arreglo = new Array(7)
        this.tamañoActual = 7;
        this.ocupacion = 0;
        this.factorOcupacion = 0;
    }

    ingresar(id, vendedor, cliente, listaProductos){ ···
    }

    colision(nuevo, hashAnt, contador, arreglo){ ···
    }

    rehashing(){ ···
    }

    primo(numero, contador){ ···
    }

    imprimir(){ ···
    }

    imprimirVentasVendedor(nombre){ ···
    }
}
```

Clase para la iniciación de la tabla hash. Posee los atributos:

- **Arreglo:** Arreglo que almacenara los objetos de tipo Venta.
- **tamañoActual:** Tamaño del arreglo actual.
- **Ocupacion:** Cantidad de espacios ocupados del arreglo.
- **factorOcupacion:** Ocupación dividido tamañoActual.

Esta clase posee 6 métodos o funciones:

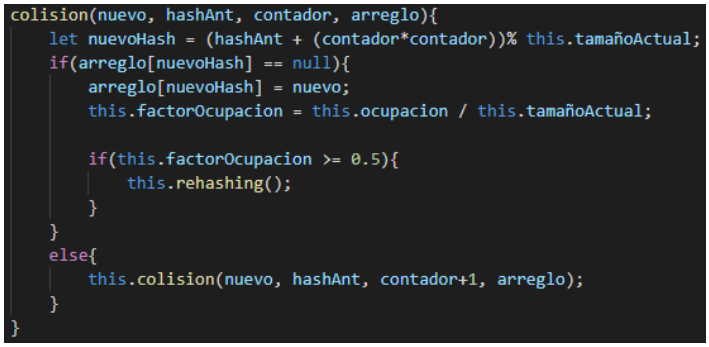
Ingresar(id, vendedor, cliente, listaProductos)



```
ingresar(id, vendedor, cliente, listaProductos){
    let nuevo = new Venta(id, vendedor, cliente, listaProductos)

    let hash = id % this.tamañoActual;

    if (this.arreglo[hash] == null){
        this.arreglo[hash] = nuevo;
        this.ocupacion += 1;
        this.factorOcupacion = this.ocupacion / this.tamañoActual;
        if(this.factorOcupacion > 0.5){
            this.rehashing();
        }
    }
    else{
        this.colision(nuevo, hash, 1, this.arreglo);
        this.ocupacion += 1;
    }
    this.factorOcupacion = this.ocupacion / this.tamañoActual;
}
```

Método para ingresar una venta dentro de la tabla hash, utiliza las funciones colision y rehashing para el manejo de las respectivas colisiones y rehashing, respectivamente.

Colision(nuevo, hashAnt, contador, arreglo)



```
colision(nuevo, hashAnt, contador, arreglo){
    let nuevoHash = (hashAnt + (contador*contador))% this.tamañoActual;
    if(arreglo[nuevoHash] == null){
        arreglo[nuevoHash] = nuevo;
        this.factorOcupacion = this.ocupacion / this.tamañoActual;

        if(this.factorOcupacion >= 0.5){
            this.rehashing();
        }
    }
    else{
        this.colision(nuevo, hashAnt, contador+1, arreglo);
    }
}
```

Método utilizado para realizar una inserción dentro de la tabla hash en caso de que la posición Hash obtenida ya se encuentre ocupada.

Rehashing();

```
rehashing(){
    let primoN = this.tamañoActual + 1
    while(!this.primo(primoN, primoN-1)){
        primoN +=1;
    }
    this.tamañoActual = primoN

    let cambioAHash = new Array(this.tamañoActual);
    for(let i = 0; i<this.arreglo.length; i++){
        if (this.arreglo[i] != null){
            let hash=this.arreglo[i].id % this.tamañoActual;

            if(cambioAHash[hash] == null){
                cambioAHash[hash] = this.arreglo[i]
            }
            else{
                this.colision(this.arreglo[i], hash, 1, cambioAHash);
            }
        }
    }
    this.arreglo = cambioAHash;
}
```

Método encargado de asignar el nuevo tamaño de la tabla hash en caso de que su ocupación sea mayor a 0.5.

Primo(numero, contador)

```
primo(numero, contador){
    if ((numero%contador) == 0 && contador!=1){
        return false;
    }
    else if((numero%contador) != 0){
        return this.primo(numero, contador-1);
    }
    else{
        return true;
    }
}
```

Función que recibe como parámetro un numero y un contador que es igual al numero ingresado menos 1. La función realizara comparaciones para determinar si el numero ingresado es primo y devolver true o false.

Imprimir()

```
imprimir(){
    let cadena = "";
    let enlaces = "";
    let i = 0;
    for(i; i<this.tamañoActual; i++){
        if(this.arreglo[i] != null){
            cadena += 'node[fillcolor="orange" label="'+this.arreglo[i].id+' '+this.arreglo[i].vendedor+' '+thi

            let aux = this.arreglo[i].listaProductos.primero;
            let contador = 0;
            enlaces += 'Hash'+i+'->P'+i+'_'+contador+';\n';

            while(aux!=null){
                cadena += 'node[label="'+aux.id+' '+aux.nombre+' '+aux.cantidad+'"]P'+i+'_'+contador+';\n';
                enlaces += 'P'+i+"_"+contador+'->P'+i+"_"+(contador+1)+';\n';
                contador += 1;
                aux = aux.siguiente;
            }
            cadena += 'node[label="Fin"]P'+i+'_'+contador+';\n';
        }
        else{
            cadena += 'node[fillcolor="orange" label="Vacio"]Hash'+i+';\n';
        }
        let siguiente = i+1;
        enlaces += 'Hash'+i+'->Hash'+siguiente+';\n';
    }
    cadena += 'node[fillcolor="azure2" label="Tabla Hash tamaño: '+i+'"]Hash'+i+';\n';
    return cadena+enlaces;
}
```

Función que devuelve una cadena String con formato DOT que contiene los nodos y enlaces para la creación de una grafica con la información de la tabla hash.

imprimirVentasVendedor(nombre)

```
imprimirVentasVendedor(nombre){
    let cadena = "";
    let i = 0;
    for(i; i<this.tamañoActual; i++){
        if(this.arreglo[i] != null && this.arreglo[i].vendedor == nombre){
            cadena += 'Informacion de la Venta:\n   ID: '+this.arreglo[i].id+'  Cliente: '+this.arreglo[i].cliente+'  Total: '+this.arreglo[i].total+'\nProductos: \n';

            let aux = this.arreglo[i].listaProductos.primero;
            while(aux!=null){
                cadena += '     * ID: '+aux.id+' Nombre: '+aux.nombre+' Cant: '+aux.precio+' Cant: '+aux.cantidad+'\n';
                aux = aux.siguiente;
            }
        }
    }
    return cadena;
}
```

Función que retorna los datos de aquellas ventas cuyo nombre de vendedor se igual al parámetro nombre recibido, en formato String.

Grafo.js

Clase ListaRutas

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

    insertar(id, nombre, distancia, adyacentes){···
    }

    insertarOrdenado(id, nombre, distancia, adyacentes, longitud, identificador){···
    }
}
```

Esta clase es utilizada para la creación de listas que contendrán los nodos para la formación de la lista de vértices y la lista de adyacentes para cada vértice. Posee los atributos:

- **Primero:** Apuntador al primer nodo de la lista.
- **Ultimo:** Apuntador al ultimo nodo de la lista.

Posee dos métodos o funciones:

Insertar(id, nombre, distancia, adyacentes)

```
insertar(id, nombre, distancia, adyacentes){
    let nuevo = new NodoRuta(id, nombre, distancia, adyacentes);
    if (this.primero == null){
        this.primero = this.ultimo = nuevo
    }
    else{
        this.ultimo.siguiente = nuevo;
        this.ultimo = nuevo;
    }
}
```

Método que se encarga de ingresar un nodo al final de la lista.

insertarOrdenado(id, nombre, distancia, adyacentes, longitud, identificador)

```
insertarOrdenado(id, nombre, distancia, adyacentes, longitud, identificador){
    let this: this NodoRuta(id, nombre, distancia, adyacentes, longitud, identificador);
    if (this.primero == null){
        this.primero=this.ultimo=nuevo;
    }
    else if (this.primero.ruta.longitud <= nuevo.ruta.longitud && this.primero.siguiente == null){
        this.primero.siguiente = nuevo;
        this.ultimo = nuevo;
    }else if (this.primero.ruta.longitud >= nuevo.ruta.longitud){
        nuevo.siguiente = this.primero;
        this.primero = nuevo;
    }else{
        let aux = this.primero;
        while(aux != null){
            if (aux == this.ultimo && aux.ruta.longitud <= nuevo.ruta.longitud){
                aux.siguiente = nuevo;
                this.ultimo = nuevo;
                break;
            }
            else if (aux != this.ultimo && aux.siguiente.ruta.longitud >= nuevo.ruta.longitud && aux.
                nuevo.siguiente = aux.siguiente;
                aux.siguiente = nuevo;
                break;
            }

            aux = aux.siguiente;
        }
    }
}
```

Método encargado de ingresar un nodo de manera ordenada ascendente dentro de la lista.

Clase Vertice

```
class Vertice{
    constructor(id, nombre, distancia, adyacentes, longitud, identificador){
        this.id = id;
        this.nombre = nombre;
        this.distancia = distancia;
        this.longitud = longitud;
        this.adyacentes = adyacentes;
        this.identificador = identificador;
    }
}
```

Clase utilizada para la creación de objetos de tipo vértice los cuales son las intersecciones del grafo. Sus atributos son:

- **Id:** Id del vértice.
- **Nombre:** Nombre del vértice
- **Distancia:** Distancia a la que se encuentra el vértice de otro.
- **Longitud:** Distancia total recorrida para llegar a este nodo.
- **Adyacentes;** Lista de vértices adyacentes.
- **Identificador:** Id de identificación del nodo para la gráfica.

Clase NodoRuta

```
class NodoRuta{
    constructor(id, nombre, distancia, adyacentes, longitud, identificador){
        this.ruta = new Vertice(id, nombre, distancia, adyacentes, longitud, identificador);
        this.siguiente = null;
    }
}
```

Clase utilizada para la creación de nodos los cuales contendrán un vértice y un apuntador llamado siguiente para la formación de una lista.

Clase Grafo

```
class Grafo{
    constructor(){
        this.listaVertices = new ListaRutas();
        this.distanciaActual = 0;
    }

    imprimir(){···
    }

    buscarEnVertices(id){···
    }

    generarDot(){···
    }

    extraer(lista){···
    }

    costoUniforme(idInicio, idFinal){···
    }
}
```

Clase utilizada para inicializar un grafo. Posee los atributos:

- **listaVertices:** Lista de vértices que conforman un grafo.
- **distanciaAcutal:** Distancia o longitud recorrida para llegar de un vértice a otro.

Posee 5 métodos o funciones:

Imprimir()

```
imprimir(){
    let aux = this.listaVertices.primero;

    while(aux!=null){
        let aux2 = aux.ruta.adyacentes.primero;
        console.log(aux.ruta.id+" "+aux.ruta.nombre);
        while(aux2 != null){
            console.log("    *"+aux2.ruta.id+" "+aux2.ruta.nombre+" "+aux2.ruta.distancia);

            aux2 = aux2.siguiente;
        }
        aux = aux.siguiente;
    }
}
```

Método que imprime en consola los vértices del grafo y su lista de vértices adyacentes.

buscarEnVertices(id):

```
buscarEnVertices(id){
    let aux = this.listaVertices.primero;

    while(aux!=null){
        if (aux.ruta.id == id){
            return aux.ruta;
        }
        aux = aux.siguiente
    }
    return null;
}
```

Función que retorna el primer el objeto ruta de la lista de vértices cuyo id sea igual al parámetro id recibido.

generarDot()

```
generarDot(){
    let aux = this.listaVertices.primero;
    let cadena = "";
    let enlaces = "";

    while(aux!=null){
        let aux2 = aux.ruta.adyacentes.primero;
        cadena += aux.ruta.id+'[ label="'+aux.ruta.id+' '+aux.ruta.nombre+'"];\n';
        while(aux2 != null){
            enlaces += aux.ruta.id+'->'+aux2.ruta.id+'[ label="'+aux2.ruta.distancia+'"];\n';;

            aux2 = aux2.siguiente;
        }
        aux = aux.siguiente;
    }

    return cadena+enlaces;
}
```

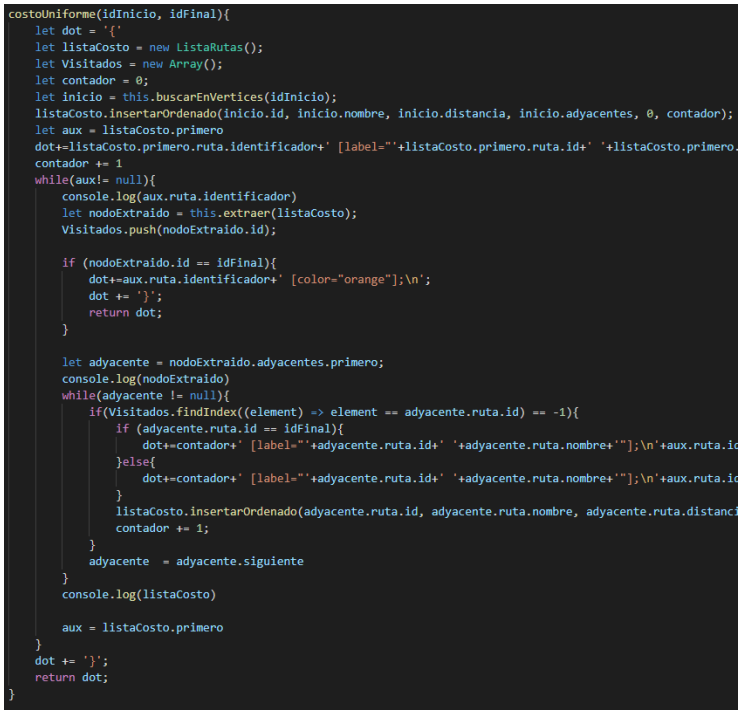
Función que retorna una cadena String que posee formato DOT para la realización de una gráfica, y posee toda la información del grafo.

Extraer(lista)

```
extraer(lista){
    let aux = lista.primero;
    if(lista.primero == lista.ultimo){
        lista.ultimo = lista.ultimo.siguiente;
    }
    lista.primero = lista.primero.siguiente;
    return aux.ruta;
}
```

Función que recibe como parámetro un objeto ListaRuta del cual se extraerá y retornará el primer nodo de la lista.

costoUniforme(idInicio, idFinal)



```
costoUniforme(idInicio, idFinal){
    let dot = '{'
    let listaCosto = new ListaRutas();
    let Visitados = new Array();
    let contador = 0;
    let inicio = this.buscarEnVertices(idInicio);
    listaCosto.insertarOrdenado(inicio.id, inicio.nombre, inicio.distancia, inicio.adyacentes, 0, contador);
    let aux = listaCosto.primero
    dot+=listaCosto.primero.ruta.identificador+' [label="'+listaCosto.primero.ruta.id+' '+listaCosto.primero.
    contador += 1
    while(aux!= null){
        console.log(aux.ruta.identificador)
        let nodoExtraido = this.extraer(listaCosto);
        Visitados.push(nodoExtraido.id);

        if (nodoExtraido.id == idFinal){
            dot+=aux.ruta.identificador+' [color="orange"];\n';
            dot += '}';
            return dot;
        }

        let adyacente = nodoExtraido.adyacentes.primero;
        console.log(nodoExtraido)
        while(adyacente != null){
            if(Visitados.findIndex((element) => element == adyacente.ruta.id) == -1){
                if (adyacente.ruta.id == idFinal){
                    dot+=contador+' [label="'+adyacente.ruta.id+' '+adyacente.ruta.nombre+'"];\n'+aux.ruta.id
                }else{
                    dot+=contador+' [label="'+adyacente.ruta.id+' '+adyacente.ruta.nombre+'"];\n'+aux.ruta.id
                }
                listaCosto.insertarOrdenado(adyacente.ruta.id, adyacente.ruta.nombre, adyacente.ruta.distanci
                contador += 1;
            }
            adyacente  = adyacente.siguiente
        }
        console.log(listaCosto)

        aux = listaCosto.primero
    }
    dot += '}';
    return dot;
}
```

Función que recibe como parámetros un id de inicio y un id de fin para la ejecución del algoritmo de costo uniforme para encontrar la ruta más corta.