TAREA 5#

Estructuras Fase 2.

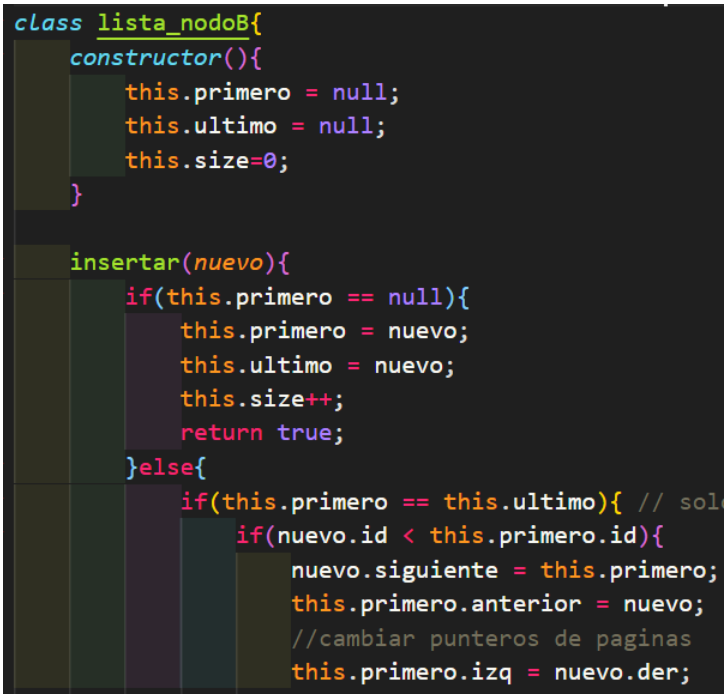
Realizar un reporte donde se descrica cada estructura realizada en la fase 2 del proyecto, añadiendo pantallazos de su codigo y comentando el funcionamiento del mismo.

ARBOL B

El árbol B es una estructura que se desarrolla creando un Nodo donde se almacenan los parámetros a manejar, en este caso los datos del inventario que son el id, nombre, precio y cantidad.

```
class nodoB{
    constructor(id,nombre,precio,cantidad){
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
        //apuntadores de lista - tipo nodoB
        this.siguiente = null;
        this.anterior = null;
        //apuntadores de arbol - tipo pagina
        this.izq = null;
        this.der = null;
    }
}
```

Después se crea otro nodo donde se inicializan los apuntadores, primero, ultimo y el tamaño. Se crea un método para insertar en el orden respectivo los nodos y ahí se crean las validaciones necesarias.

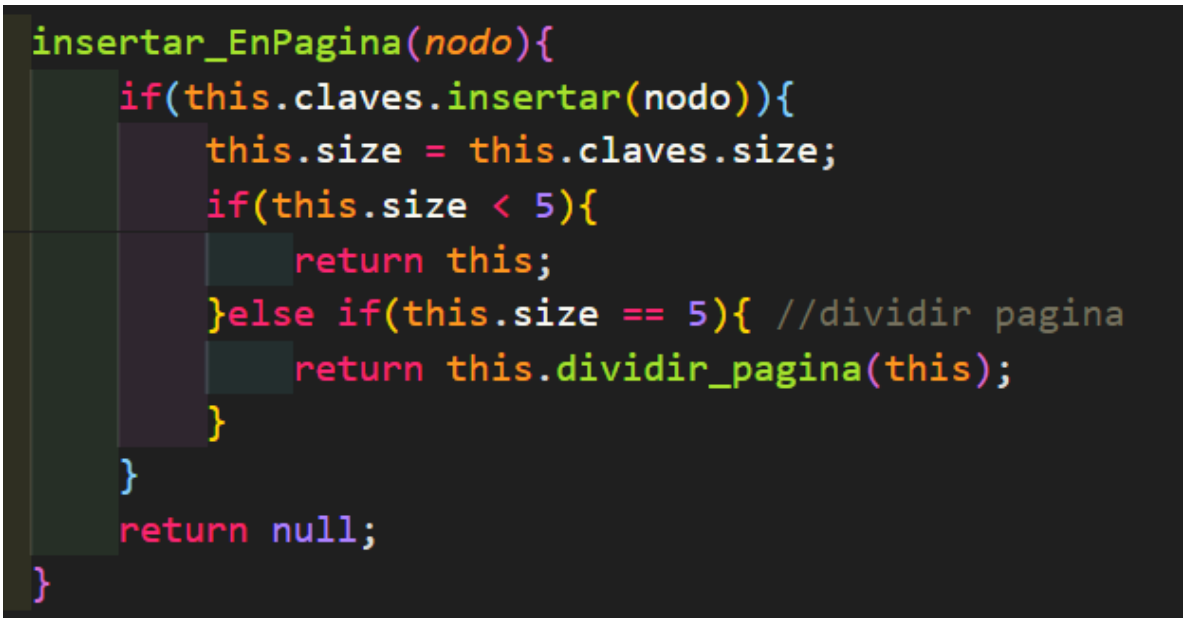


```
class lista_nodoB{
    constructor(){
        this.primero = null;
        this.ultimo = null;
        this.size=0;
    }

    insertar(nuevo){
        if(this.primero == null){
            this.primero = nuevo;
            this.ultimo = nuevo;
            this.size++;
            return true;
        }else{
            if(this.primero == this.ultimo){ // sol
                if(nuevo.id < this.primero.id){
                    nuevo.siguiente = this.primero;
                    this.primero.anterior = nuevo;
                    //cambiar punteros de paginas
                    this.primero.izq = nuevo.der;
```

Se crea la clase de la página donde se inicializan los apuntadores de la raíz, las claves máxima y mínima, el tamaño y las clases que son referencia y apuntan a la lista del nodoB.

Se crea también un método para insertar los nodos y las validaciones correspondientes para poder dividir la pagina.



```
insertar_EnPagina(nodo){
    if(this.claves.insertar(nodo)){
        this.size = this.claves.size;
        if(this.size < 5){
            return this;
        }else if(this.size == 5){ //dividir pagina
            return this.dividir_pagina(this);
        }
    }
    return null;
}
```

Respectivamente se crea la clase del árbol B, donde se desarrolla el constructor, que contiene la raíz, el orden y la altura del árbol, se insertan los nodos, se recorren y se crea el método para graficar.

```
class Arbol_B{
    constructor(){
        this.raiz = null;
        this.orden =5;
        this.altura =0;
    }
    insertar_nodo(id,nombre,precio,cantidad){
        let nuevo = new nodoB(id,nombre,precio,cantidad);
        if(this.raiz == null){
            this.raiz = new pagina();
            this.raiz.raiz = true;
            this.raiz = this.raiz.insertar_EnPagina(nuevo);
            //console.log("se inserto el valor "+this.raiz.c
        }else{
            if(this.altura==0){
                let respuesta = this.raiz.insertar_EnPagina(
                if(respuesta instanceof pagina){// la raiz r
                    this.raiz = respuesta;
```

GRAFO:

Se crea el grafo para almacenar los parámetros que representan el recorrido de las rutas, el id, el nombre, la distancia y los apuntadores de los nodos, así mismo se inicializaron la ponderación y los adyacentes, que hace referencia a la lista de adyacentes.

```
class nodo{
    constructor(id,nombre, distancia){
        this.id = id;
        this.nombre = nombre;
        this.distancia = distancia;
        this.siguiente = null;
        this.anterior = null;
        this.ponderacion=0;
        this.adyasentes = new lista_adyasentes();
    }
}
```

Se crea el método de insertar donde se agregan los nodos principales, las validaciones correspondientes para que funcione. Se construye el grafo con sus métodos de buscar, agregar adyacente, mostrar, para recorrerlos y obtenerlos, también se crea el método de graficar donde se recorren los nodos y se agregar para generar un .dot.

```
graficar(){
    let cadena= "digraph grafo {\n rankdir=\"LR\" \n"
    let aux = this.primero;
    while(aux != null){
        cadena+="n"+aux.id+"[label= \""+aux.id+ " "+aux.nomb
        aux = aux.siguiente;
    }
    // graficar relaciones
    aux = this.primero;
    while(aux != null){
        let aux2 = aux.adyasentes.primero;
        while(aux2 != null){
            cadena+= "n"+aux.id+" -> n"+aux2.id+" [label=\"
            aux2 = aux2.siguiente;
        }
        aux = aux.siguiente;
    }
    cadena += "}"
    console.log(cadena);
}
```

TABLA HASH

La tabla hash se crea para almacenar los parámetros, el identificador de la venta, nombre del vendedor, nombre del cliente y el total de la venta.

Se crea el hash, inicializando las claves, las claves usadas, y el tamaño del hash.

Se crean los métodos de iniciar el arreglo, calcular el hash, solución de colisiones, generar el hash y recorrer.

```
calcular_hash(id_venta){ //,nombre_vendedor,nombre_clien
    //metodo de division
    let resultado=0;
    resultado= id_venta % this.size; // agregar aqui o n
    return resultado;
}

solucion_coliciones(indice){ //metodo de exploracion cua
    let nuevo_indice =0;
    let i=0;
    let disponible = false;

    while(disponible == false){
        nuevo_indice = indice + Math.pow(i,2);
        //validar que nuevo_indice sea menor al tañano d
        if(nuevo_indice>= this.size){
            nuevo_indice = nuevo_indice-this.size;
        }
        //validar que la posicion del nuevo indice este
        if(this.claves[nuevo_indice]==null){
            disponible= true;
        }
        i++;
    }
    return nuevo_indice;
}
```

```
recorrer(){
    for(var i =0;i<this.size;i++){
        if(this.claves[i]!=null){
            console.log("-->"+this.claves[i].id_venta+th
        }else{
            console.log("------------");
        }
    }
}
```

BLOCK CHAIN

Se crea inicialmente un constructor con los parámetros del índice, data, previus Hash, los apuntadores de la fecha que actualiza, el hasho, nonce.

```
class bloque{
    constructor(indice,data,previusHash){
        this.indice = indice;
        this.data = data;
        this.fecha = Date.now();
        this.previusHash = previusHash;
        this.hash = this.crearHash();
        this.nonce =0;

        this.prueba_de_trabajo(3);
    }
```

Se crea el hash y la prueba de trabajo para manejar la dificultad:

```
crearHash(){
  //usar libreria
  return crypto.createHash('sha256').update(this.indice+t
}

prueba_de_trabajo(dificultad){
  while(this.hash.substring(0,dificultad) !== Array(dific
    this.nonce++;
    this.hash = this.crearHash();
    //console.log("->nonce " +this.nonce);
  }
  //console.log(this.hash);
  return this.hash;
}
```

Se crea la clase cadena donde se inicializan los nodos del índice, las cadenas.

```
class cadena{
  constructor(){
    this.indice=0;
    this.cadena =[];
    this.cadena.push(this.Bloque_genesis());
  }
```

Se crea el bloque genesis donde se desarrolla una variable del bloque y se fuerza el índice para retornar el genesis.

```
Bloque_genesis(){
  let genesis = new bloque(this.indice,"bloque Genesis","").
  this.indice++;
  return genesis;
}
```

Se crean los métodos de agregar y recorrer

```
agregar(data){
  let nuevo = new bloque(this.indice,data,this.cadena[this.
  this.indice++;
  this.cadena.push(nuevo);
}

recorrer(){
  for(let item of this.cadena){
    console.log(item);
  }
}
```

Por ultimo se crean unos diccionarios con formato json para pushar la información y revisar su funcionamiento:

```
let blockChain = new cadena();

let info=[]

let nueva_venta = {
  "id":3,
  "vendedor":"vendedor3",
  "cliente":"cliente1",
  "productos":[
    {
    "id":1,
    "cantidad":3
    }
  ]
};
info.push(nueva_venta)

info.push({
  "id":1,
  "vendedor":"vendedor1",
  "cliente":"cliente1",
  "productos":[
    {
    "id":1,
    "cantidad":3
    }
  ]
}
```