Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Estructuras de Datos

Estructuras Fase 1

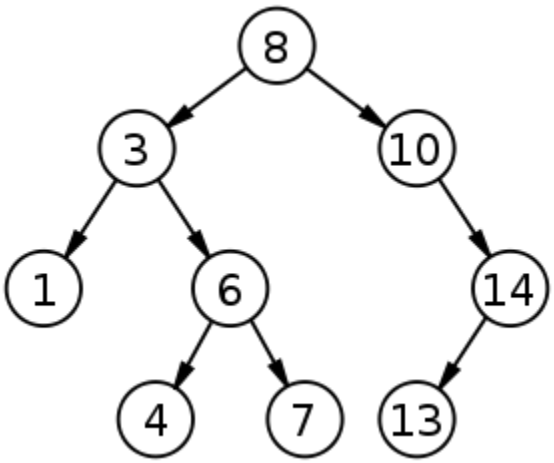
Fabian Esteban Reyna Juárez

Carné: 202003919

Guatemala, diciembre 2021

1. Árbol Binario de Búsqueda (ABB)

Es una estructura de datos no lineal que posee un numero finito de elementos, llamados nodos, unidos por líneas dirigidas llamadas ramas.



- **Implementación**:
  Se construyó un ABB para almacenar a los proveedores, con sus respectivos atributos.
  - Nodo: El nodo posee la información del proveedor y los apuntadores izq y der para poder navegar en el arbol

```
class nodo_proveedor {
    constructor(id, nombre, direccion, telefono, correo) {
        this.id = id
        this.name = nombre
        this.dir = direccion
        this.tel = telefono
        this.correo = correo
        this.izq = null
        this.der = null
    }
}
```

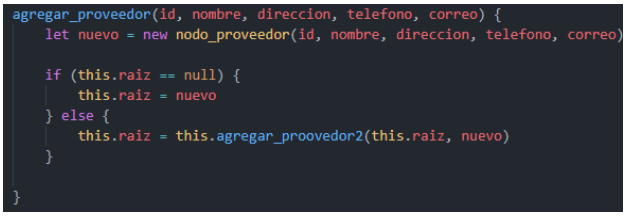
Nodo abb proveedor

  - Árbol: La clase del árbol posee únicamente el apuntador a la raíz del árbol y posee sus respectivos métodos(inserción, eliminación y búsqueda)

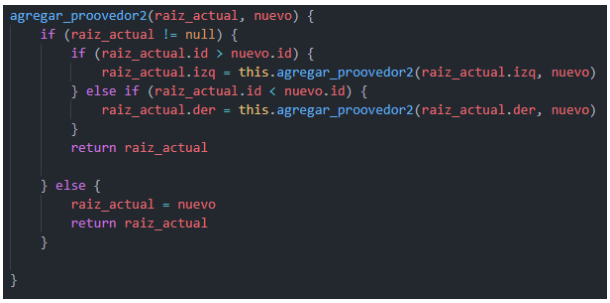
```
class abb_proveedor {
    constructor() {
        this.raiz = null
    }
```

Clase abb proveedor

- Método de inserción: La inserción al árbol se realiza por medio de dos funciones. La primera verifica si la raíz es nula para poder ingresarlo en esta y en caso contrario lo ingresa al segundo método. El segundo método funciona de forma recursiva, navegando en el árbol según sea necesario (valores mayores a la raíz a la derecha y menores a la izquierda) y al encontrar el lugar correcto lo inserta y retorna los nodos raíz utilizados.



Método inicial



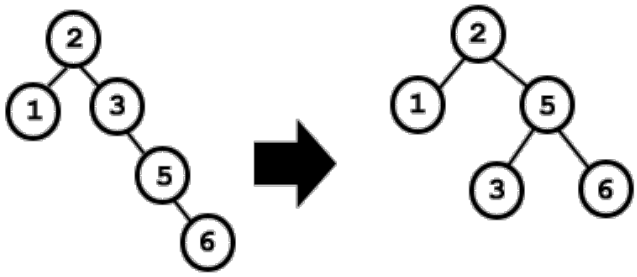
Método recursivo

- Método de eliminación: Recibe el id del proveedor a eliminar y lo va buscando dentro del árbol, al encontrarlo busca el nodo derecho mas a la izquierda para poder colocarlo en su lugar y borrarlo como si fuera una hoja. En caso de ser una hoja lo elimina directamente.

Método de eliminación

2. Árbol AVL

Es una estructura de datos no lineal que posee un numero finito de elementos, llamados nodos, unidos por líneas dirigidas llamadas ramas. Este es una mejora al ABB. Busca balancear sus nodos de forma automática mediante el calculo de alturas en la raíz. El balanceo de sus nodos busca evitar que el árbol se vuelva degenerado.



- **Implementación**:
  Se construyó un AVL para almacenar a los vendedores, con sus respectivos atributos.
  - Nodo: El nodo posee la información del vendedor y los apuntadores izq y der para poder navegar en el árbol.

```
class nodo_vendedor {
    constructor(id, nombre, edad, correo, password) {
        this.id = id
        this.name = nombre
        this.edad = edad
        this.correo = correo
        this.password = password
        this.listaMeses = new lista_meses()
        this.listaClientes = new lista_cliente()
        this.altura = 0
        this.izq = null
        this.der = null
    }
}
```

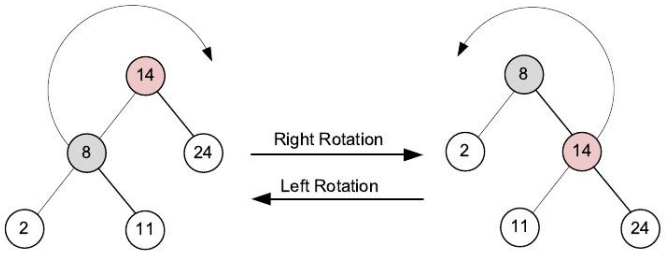
Nodo avl vendedor

  - Árbol: La clase del árbol posee únicamente el apuntador a la raíz del árbol y posee sus respectivos métodos(inserción, eliminación y búsqueda)

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

Clase avl vendedor

- Método de inserción: La inserción se realiza de la misma forma que el árbol ABB, con la diferencia que en el método recursivo se calculan las alturas y se realizan las rotaciones correspondientes. Las rotaciones son: derecha, izquierda, derecha-izquierda, izquierda-derecha.
  La rotación derecha e izquierda se basan en intercambiar uno de los nodos a la raíz del subárbol, para que este quede balanceado. Este proceso se realiza en cada subárbol.
  Las rotaciones dobles suceden cuando una rotación simple no balancea completamente el árbol. Se verifica por medio de comparaciones en las alturas cual rotación realizar.



Rotaciones

```
agregar_vendedor(id, nombre, edad, correo, password) {
    let nuevo = new nodo_vendedor(id, nombre, edad, correo, password)

    if (this.raiz == null) {
        this.raiz = nuevo
    } else {
        this.raiz = this.agregar_vendedor2(this.raiz, nuevo)
    }
}
```

Método inicial

```
agregar_vendedor2(raiz_actual, nuevo) {
    if (raiz_actual != null) {
        if (raiz_actual.id > nuevo.id) {
            raiz_actual.izq = this.agregar_vendedor2(raiz_actual.izq, nuevo)

            if (this.calcular_altura(raiz_actual.der) - this.calcular_altura(raiz_actual.izq) == -2) {
                if (nuevo.id < raiz_actual.izq.id) {
                    raiz_actual = this.rotacion_izq(raiz_actual)

                } else {
                    raiz_actual = this.rotacion_izq_der(raiz_actual)
                }
            }

        } else if (raiz_actual.id < nuevo.id) {
            raiz_actual.der = this.agregar_vendedor2(raiz_actual.der, nuevo)

            if (this.calcular_altura(raiz_actual.der) - this.calcular_altura(raiz_actual.izq) == 2) {
                if (nuevo.id > raiz_actual.der.id) {
                    raiz_actual = this.rotacion_der(raiz_actual)
                } else {
                    raiz_actual = this.rotacion_der_izq(raiz_actual)
                }
            }

        }
        raiz_actual.altura = this.altura_maxima(raiz_actual.der, raiz_actual.izq) + 1
        return raiz_actual
    } else {
        raiz_actual = nuevo
        return raiz_actual
    }
}
```

Método recursivo

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

altura_maxima(nodo1, nodo2) {
    let h1 = this.calcular_altura(nodo1)
    let h2 = this.calcular_altura(nodo2)
    if (h2 >= h1) {
        return h2
    }
    return h1
}
```

Métodos auxiliares para las rotaciones

```
rotacion_izq(nodo) {
    let aux = nodo.izq
    nodo.izq = aux.der
    aux.der = nodo
    nodo.altura = this.altura_maxima(nodo.der, nodo.izq) + 1
    aux.altura = this.altura_maxima(nodo.altura, nodo.izq) + 1
    return aux
}

rotacion_der(nodo) {
    let aux = nodo.der
    nodo.der = aux.izq
    aux.izq = nodo
    nodo.altura = this.altura_maxima(nodo.izq, nodo.der) + 1
    aux.altura = this.altura_maxima(nodo.altura, nodo.der) + 1
    return aux
}

rotacion_izq_der(nodo) {
    nodo.izq = this.rotacion_der(nodo.izq)
    let aux = this.rotacion_izq(nodo)
    return aux
}

rotacion_der_izq(nodo) {
    nodo.der = this.rotacion_izq(nodo.der)
    let aux = this.rotacion_der(nodo)
    return aux
}
```

Rotaciones

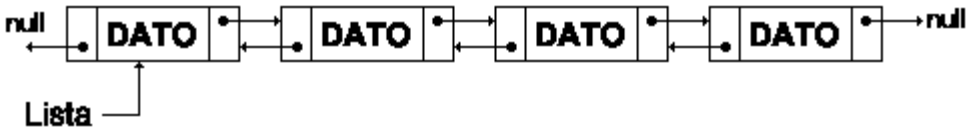
- Método de eliminación: Recibe el id del vendedor a eliminar y lo va buscando dentro del árbol, al encontrarlo busca el nodo derecho más a la izquierda para poder colocarlo en su lugar y borrarlo como si fuera una hoja. En caso de ser una hoja lo elimina directamente. Además, tiene que verificar nuevamente que el árbol este balanceado y lo realiza utilizando los mismos métodos de las rotaciones

```
eliminar_vendedor(raiz_actual, id) {
    if (raiz_actual == null) {
        return raiz_actual
    }

    if (id < raiz_actual.id) {
        raiz_actual.izq = this.eliminar_vendedor(raiz_actual.izq, id)
    } else if (id > raiz_actual.id) {
        raiz_actual.der = this.eliminar_vendedor(raiz_actual.der, id)
    } else {
        if (raiz_actual.izq == null) {
            return raiz_actual.der
        } else if (raiz_actual.der == null) {
            return raiz_actual.izq
        }
        var temp = this.obtener_nodo_min(raiz_actual.izq)
        raiz_actual.id = temp.id
        raiz_actual.edad = temp.edad
        raiz_actual.correo = temp.correo
        raiz_actual.password = temp.password
        raiz_actual.listaMeses = temp.listaMeses
        raiz_actual.listaClientes = temp.listaClientes

        raiz_actual.izq = this.eliminar_vendedor(raiz_actual.izq, temp.id)
    }
    if (raiz_actual == null) {
        return raiz_actual
    }
    raiz_actual.altura = this.altura_maxima(raiz_actual.izq, raiz_actual.der) + 1

    let balanceo = this.calcular_altura(raiz_actual.izq) - this.calcular_altura(raiz_actual.der)
    let balanceo_izq = this.balanceo_nodos(raiz_actual.izq)
    let balanceo_der = this.balanceo_nodos(raiz_actual.der)


    if (balanceo > 1 && balanceo_izq >= 0) {
        return this.rotacion_der(raiz_actual)
    }

    if (balanceo > 1 && balanceo_izq < 0) {
        raiz_actual.izq = this.rotacion_izq(raiz_actual.izq)
        return this.rotacion_der(raiz)
    }


    if (balanceo < -1 && balanceo_der <= 0) {
        return this.rotacion_izq(raiz_actual)
    }

    if (balanceo < -1 && balanceo_der > 0) {
        raiz_actual.der = this.rotacion_der(raiz_actual.der)
        return this.rotacion_der(raiz_actual)
    }
    return raiz_actual


}
```

Método de eliminación

3. Lista Doblemente enlazada

Es una estructura de datos lineal que posee un numero finito de elementos, llamados nodos, unidos por dos apuntadores, al predecesor y al sucesor.



- **Implementación**:
  Se construyó una lista doblemente enlazada para almacenar los clientes de cada proveedor y los meses en los cuales tendrá su calendario. Ambas listas funcionan de la misma manera.
  - Nodo: El nodo posee la información del cliente y los apuntadores ant y sig para poder navegar en la lista.

```
class nodo_cliente {
    constructor(id, nombre, correo) {
        this.id = id
        this.name = nombre
        this.correo = correo
        this.ant = null
        this.sig = null
    }
}
```

Nodo cliente

```
class nodo_mes {
    constructor(mes) {
        this.mes = mes
        this.calendario = new matriz_calendario()
        this.ant = null
        this.sig = null
    }
}
```

Nodo mes

  - Lista: La clase lista posee dos apuntadores, hacia el inicio de la lista y hacia el final; además posee sus respectivos métodos (inserción con sus respectivas validaciones y búsquedas).

```
class lista_cliente {
    constructor() {
        this.inicio = null
        this.final = null
    }
```

Clase lista cliente

```
class lista_meses {
    constructor() {
        this.inicio = null
        this.final = null
    }
```

Clase lista meses

- Método de inserción: La inserción en la lista de clientes es inserción al final, mientras que para los meses es una inserción en orden que valida que no se repitan los elementos en esta.

```
agregar_cliente(id, nombre, correo) {
    let nuevo = new nodo_cliente(id, nombre, correo)
    if (this.inicio == null) {
        this.inicio = nuevo
    } else {
        this.final.sig = nuevo
        nuevo.ant = this.final
    }
    this.final = nuevo
}
```

Método inserción cliente

```
agregar_mes(mes) {
    if (mes <= 12) {
        let nuevo = new nodo_mes(mes)
        if (this.inicio == null) {
            this.inicio = nuevo
            this.final = nuevo
        } else {
            let estado = this.verifica_mes(nuevo)
            if (estado) {
                this.final.sig = nuevo
                nuevo.ant = this.final
                this.final = nuevo
            }
        }
    }
}
```

Método inserción mes

```
verifica_mes(nuevo) {
    let aux = this.inicio
    while (aux != null) {
        if (aux.mes == nuevo.mes) {
            return false
        }
        aux = aux.sig
    }
    return true
}
```

Método verificación de mes

- Método de búsqueda: Ambas listas realizan la búsqueda de la misma manera. Reciben el id o mes según sea el caso y lo busca en la lista. Al encontrar lo solicitado retorna el nodo correspondiente y en caso de no encontrar coincidencias retorna null.

```
buscar_cliente(id) {
    let aux = this.inicio
    while (aux != null) {
        if (aux.id == id) {
            return aux
        }
        aux = aux.sig
    }
    return null
}
```

Método de búsqueda cliente

```
obtener_nodo_mes(mes) {
    let aux = this.inicio
    while (aux != null) {
        if (aux.mes == mes) {
            return aux
        }
        aux = aux.sig
    }
    return null
}
```

Método de búsqueda mes

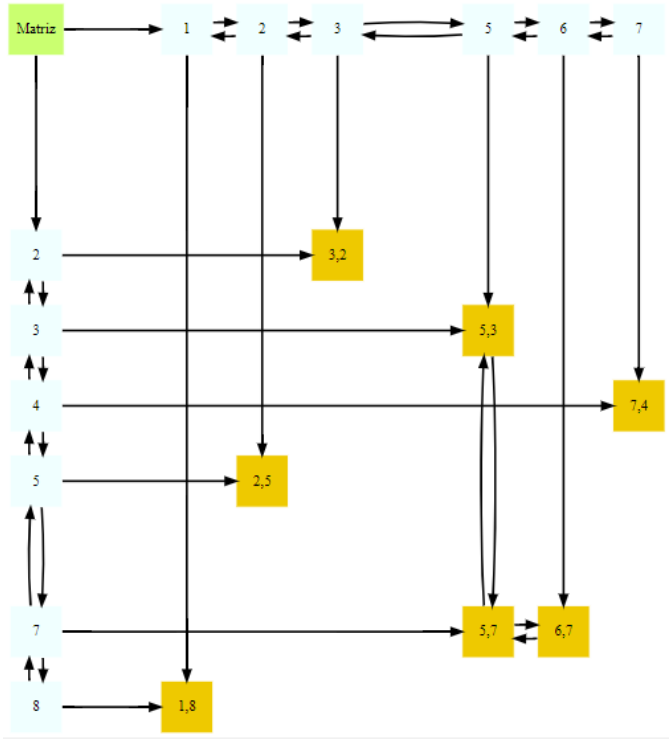
- Método de eliminación: Este lo posee la lista de clientes. Utiliza el método de buscar para obtener el nodo. Si el nodo es distinto a null se realizan las siguientes validaciones:
  - Si inicio es igual a final e igual al nodo, significa que es el único elemento de la lista y la dejara vacía.
  - Si el inicio es igual al nodo, se elimina el nodo y el puntero al nodo inicio pasa a ser el siguiente del nodo a eliminar.
  - Si el final es igual al nodo, se elimina el nodo y el puntero al nodo final pasa a ser el anterior del nodo a eliminar.
  - Si no se cumple ninguna de las anteriores, es un nodo intermedio por lo cual se elimina intercambiando los punteros del nodo anterior y nodo siguiente.

```
eliminar_cliente(id) {
    var nodo = this.buscar_cliente(id)
    if (nodo != null) {
        if (this.inicio == this.final == nodo) {
            this.inicio = this.final = null
        } else if (nodo == this.inicio) {
            this.inicio = nodo.sig
            nodo.sig.ant = null
        } else if (nodo == this.final) {
            this.final = nodo.ant
            nodo.ant.sig = null
        } else {
            nodo.ant.sig = nodo.sig
            nodo.sig.ant = nodo.ant
        }
        nodo.sig = null
        nodo.ant = null
    }

}
```

Método de eliminación cliente

4. Matriz dinámica

Es una estructura de datos lineal que posee dos listas dobles de cabeceras que se conectan a nodos internos.



- **Implementación**:
  Se construyó una matriz dinámica para almacenar el calendario de eventos de cada mes de los vendedores. Este se construye utilizando dos listas dobles que funcionan de encabezados, siendo en este caso particular x-días y y-horas. Dentro de los atributos de los nodos de las listas se encuentra el acceso el cual permite conectarse con los nodos internos o celdas de la matriz

    - Listas cabeceras
        - Nodo: Nodo que almacena el día u hora y sus apuntadores ant y sig para navegar en la lista.

```
class nodo_dia {
    constructor(dia) {
        this.dia = dia
        this.sig = null
        this.ant = null
        this.acceso = null
    }
}
```

Nodo dia

```
class nodo_hora {
    constructor(hora) {
        this.hora = hora
        this.sig = null
        this.ant = null
        this.acceso = null
    }
}
```

Nodo dia

- Lista: posee dos apuntadores, hacia el inicio de la lista y hacia el final; además posee sus respectivos métodos. Ambas listas insertan los elementos en orden y verifican que no se repitan al igual que la lista de meses mencionada anteriormente.

```
class lista_dia {
    constructor() {
        this.inicio = null
        this.final = null
    }
```

Lista día

```
class lista_hora {
    constructor() {
        this.inicio = null
        this.final = null
    }
```

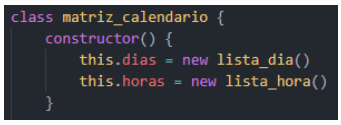
Lista hora

- Nodo Evento: Posee la información del evento (dia, hora y descripción) y los apuntadores arriba, abajo, der e izq.

```
class evento {
    constructor(dia, hora, desc) {
        this.dia = dia
        this.hora = hora
        this.desc = desc
        this.arriba = null
        this.abajo = null
        this.izq = null
        this.der = null
    }
}
```

Nodo evento

- Matriz: La clase matriz posee dos atributos los cuales son instancias de las listas cabeceras (dias y horas). Esta posee dos métodos, los cuales son la inserción y el graficar neato.

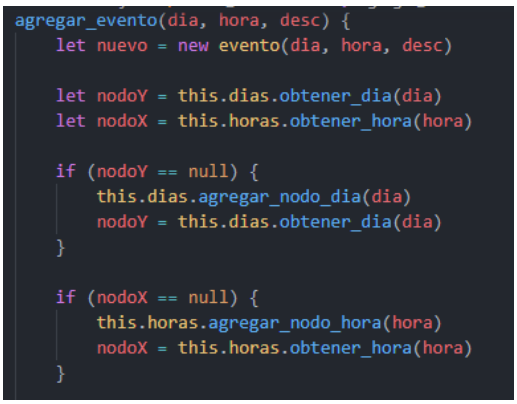


Matriz calendario

- Inserción: La inserción en la matriz se realiza en tres etapas.
  La primera etapa verifica que en las listas encabezados existan los días u horas a agregar, en caso de existir se almacenara en una variable el nodo correspondiente y en caso contrario se creara y se almacenara el nodo día u hora recién creado.

  La segunda etapa es la inserción en hora, la cual se verifica si el acceso del nodo existe, si no existe se coloca en este y caso contrario se busca la posición que debe tener, recorriendo los nodos evento hacia la derecha hasta poder ingresar el nodo evento.

  La tercera etapa es la inserción en dia, que a diferencia de la inserción de hora, se recorre el nodo hacia abajo para poder encontrar su posición.



Inserción etapa 1

```
//insercion en hora
if (nodoX.acceso == null) {
    nodoX.acceso = nuevo
} else {
    if (nuevo.dia < nodoX.acceso.dia) {
        nodoX.acceso.izq = nuevo
        nuevo.der = nodoX.acceso
        nodoX.acceso = nuevo
    } else {
        let aux = nodoX.acceso
        while (aux != null) {
            if (nuevo.dia < aux.dia) {
                nuevo.der = aux
                nuevo.izq = aux.izq
                aux.izq.der = nuevo
                aux.izq = nuevo
                break
            } else if (nuevo.dia == aux.dia && nuevo.hora == aux.hora) {
                break
            } else {
                if (aux.der == null) {
                    nuevo.izq = aux
                    aux.der = nuevo
                    break
                } else {
                    aux = aux.der
                }
            }
        }
    }
}
```

Inserción etapa 2

```
//insercion en dia
if (nodoY.acceso == null) {
    nodoY.acceso = nuevo
} else {
    if (nuevo.hora < nodoY.acceso.hora) {
        nuevo.abajo = nodoY.acceso
        nodoY.acceso.arriba = nuevo
        nodoY.acceso = nuevo
    } else {
        let aux = nodoY.acceso
        while (aux != null) {
            if (nuevo.hora < aux.hora) {
                nuevo.abajo = aux
                nuevo.arriba = aux.arriba
                aux.arriba.abajo = nuevo
                aux.arriba = nuevo
                break
            } else if (nuevo.hora == aux.hora && nuevo.dia == aux.dia) {
                break
            } else {
                if (aux.abajo == null) {
                    aux.abajo = nuevo
                    nuevo.arriba = aux
                    break
                } else {
                    aux = aux.abajo
                }
            }
        }
    }
}
```

Inserción etapa 3

- Método de graficar neato: Este metodo genera la cadena que genera la grafica de la matriz utilizando graphviz y neato.
  Inicialmente se genera un nodo principal, el cual servirá para darle mejor efecto visual y luego se recorrerán las listas encabezado, generando sus nodos y conexiones entre si. Posteriormente se recorrerán los nodos eventos recorriendo desde el acceso la lista de dias y finalmente se generarán las conexiones entre nodos y con las cabeceras de la matriz.

```
graficar_neato() {
    let cadena = ""
        //definicion nodo inicial
    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"
    cadena += "node[label = Calendario fillcolor=\" darkolivegreen1\" pos = \"-1,1!\"]principal; \n"

    //nodos cabecera x (dias)
    let aux = this.dias.inicio
    while (aux != null) {
        cadena += "node[label=" + aux.dia + " fillcolor=\" azure1\" pos = \"" + aux.dia + ",1!\"]x" + aux.dia + ";\n"
        aux = aux.sig
    }

    //conexion nodos x (dias)
    aux = this.dias.inicio
    while (aux.sig != null) {
        cadena += "x" + aux.dia + "->x" + aux.sig.dia + ";\n"
        cadena += "x" + aux.sig.dia + "->x" + aux.dia + ";\n"
        aux = aux.sig
    }

    //conexion nodo principal a dias
    if (this.dias.inicio != null) {
        cadena += "principal->x" + this.dias.inicio.dia + ";\n"
    }
```

```
//creacion nodos y (horas)
let aux2 = this.horas.inicio
while (aux2 != null) {
    cadena += "node[label=" + aux2.hora + " fillcolor=\" azure1\" pos = \"-1,-" + aux2.hora + "!\"]y" + aux2.hora + ";\n"
    aux2 = aux2.sig
}

//conexion nodos y (horas)
aux2 = this.horas.inicio
while (aux2.sig != null) {
    cadena += "y" + aux2.hora + "->y" + aux2.sig.hora + ";\n"
    cadena += "y" + aux2.sig.hora + "->y" + aux2.hora + ";\n"
    aux2 = aux2.sig
}

//conexio (property) matriz_calendario.horas: lista_hora
if (this.horas.inicio != null) {
    cadena += "principal->y" + this.horas.inicio.hora + ";\n"
}

//creacion de nodos evento y conexion con cabeceras dias
aux = this.dias.inicio
while (aux != null) {
    let auxiliar = aux.acceso
    while (auxiliar != null) {
        cadena += "node[label=" + auxiliar.desc + " fillcolor=\" gold2\" pos = \"" + auxiliar.dia + ",-" + auxiliar.hora + "!\"]x" + aux
        auxiliar = auxiliar.abajo
    }

    auxiliar = aux.acceso
```

```
  while (aux != null) {
      let auxiliar = aux.acceso
      while (auxiliar != null) {
          cadena += "node[label=" + auxiliar.desc + " fillcolor=\" gold2\" pos = \"" + auxiliar.dia + ",-" + auxiliar.hora + "!\"]x" + auxi
          auxiliar = auxiliar.abajo
      }

      auxiliar = aux.acceso
      while (auxiliar.abajo != null) {
          cadena += "x" + auxiliar.dia + "y" + auxiliar.hora + "->x" + auxiliar.abajo.dia + "y" + auxiliar.abajo.hora + ";\n"
          cadena += "x" + auxiliar.abajo.dia + "y" + auxiliar.abajo.hora + "->x" + auxiliar.dia + "y" + auxiliar.hora + ";\n"
          auxiliar = auxiliar.abajo
      }

      if (aux.acceso != null) {
          cadena += "x" + aux.dia + "->x" + aux.acceso.dia + "y" + aux.acceso.hora + ";\n"
      }
      aux = aux.sig
  }

  //conexion nodos con horas
  aux2 = this.horas.inicio
  while (aux2 != null) {
      let auxiliar2 = aux2.acceso
      while (auxiliar2.der != null) {
          cadena += "x" + auxiliar2.dia + "y" + auxiliar2.hora + "->x" + auxiliar2.der.dia + "y" + auxiliar2.der.hora + ";\n"
          cadena += "x" + auxiliar2.der.dia + "y" + auxiliar2.der.hora + "->x" + auxiliar2.dia + "y" + auxiliar2.hora + ";\n"
          auxiliar2 = auxiliar2.der
      }
```

```
      if (aux2.acceso != null) {
          cadena += "y" + aux2.hora + "->x" + aux2.acceso.dia + "y" + aux2.acceso.hora + ";\n"
      }
      aux2 = aux2.sig
  }
  cadena += "\n}"
  return cadena
}
```