**David Upegui Londoño  
Juan Manuel Vera**

**Lógica Y Representación III. Practica II.**

**Manual Técnico: Árboles como Lista Generalizada**

Este manual servirá para entender el funcionamiento de la aplicación Árboles como listas generalizadas creada como solución de la Practica II del curso Lógica y Programación III. En este manual encontrará la información relacionada con los métodos e instrumentos usados en el desarrollo de la aplicación.

**Tecnologías Empleadas:**

* JavaScript
* HTML
* CSS
* GitHub
* VisualStudio Code

**Clases del modelo:**

En este apartado están todas las clases que tienen como objetivo abstraer un objeto de la realidad.

* **Clase NodoLg:** Extiende de la clase **NodoSimple**¸tiene como atributo añadido *(int)sw*. Este atributo puede tener 2 valores, 0 o 1, cuando sw = 0 entonces el campo de dato del nodo contiene un registro, cuando el sw = 1 el campo de dato del nodo contiene un nodo de la clase NodoLg.
* **Clase ArbolLgConstructionError:** Extiende de la clase **Error** nativa JavaScript. La clase contiene un constructor con parámetro msg (String) que será imprimido al saltar la excepción, además, el nombre del error será “ArbolLgConstructionError”.
* **Clase ArbolLg:** Es la clase principal de la aplicación, contiene como atributos *(NodoLg) raiz* y *(String) string* que contiene la hilera de átomos, comas y paréntesis del árbol, posee funciones para construir el árbol y determinar datos como la altura, el grado, número de hojas del árbol, etc.

**Clases Controladoras:**

* **Clase Visual:** Contiene métodos que se encargan de controlar la interfaz visual modificando el DOM por medio de clases o inyecciones de código.

**Funciones:**

* **validarArbolStr(String str):** Es un método estático de la clase ArbolLg que se ejecuta antes de la construcción del árbol por medio de la función construyeArbol. Esta función recorre la Hilera ingresadas en búsqueda de errores que puedan generar inconsistencias durante la construcción del árbol. Alguno de los errores, por ejemplo, encontrar 2 paréntesis abiertos consecutivos o que la raíz del árbol tenga hermanos. La función es de orden O(n).
* **construyeArbol(String str):** Es un método de instancia de la clase **ArbolLg** Recibe como parámetro la hilera ingresada por el usuario luego de comprobar si tiene errores, se encarga de construir la representación de lista generalizada correspondiente al árbol que será analizado por la aplicación. La función es de orden O(n)
* **muestraArbol():** Es un método de instancia de la clase **ArbolLg.** Este método recorre todo el árbol y lo imprime en consola haciendo uso de un Array nativo de JavaScript usado como estructura LIFO y un ciclo. Esta función es prototipo de las demás funciones dado que en todas las funciones hace falta un recorrer el árbol.
* **Altura():** Es un método de instancia de la clase **ArbolLg**. El método recorre el árbol evaluando la altura cada vez que se desplaza hacia un hijo y comparándola con la altura máxima registrada. La función retorna la altura máxima como *int*.
* **grado():** Es un método de instancia de la clase **ArbolLg.** El método recorre el árbol evaluando el grado de cada elemento que contenga hijos, además, el grado evaluado es comparado con el grado máximo registrado. La función retorna el grado máximo como *int.*
* **hojas():** Es un método de instancia de la clase **ArbolLg.** El método recorre el árbol contando los elementos que no tengan hijos. Retorna la cantidad de elementos sin hijos.
* **buscarRegistro(char d):** Es un método de instancia de la clase **ArbolLg.** El método recibe un parámetro *d* que corresponde a un registro en el árbol que se desee encontrar. El método retorna la primera coincidencia de *(NodoLg) nodo* que contenga el dato *d*. En el caso de no encontrar no haber coincidencias retorna *null.*
* **gradoRegistro(char d):** Es un método de instancia de la clase **ArbolLg.** El método recibe un parámetro *d* que corresponde a un registro en el árbol al cual se le desea encontrar el grado. En la función se recorre el árbol hasta encontrar el dato *d,* luego se busca el grado del elemento y lo retorna. En el caso en que no encuentre el elemento retorna -1.
* **nivelRegistro(char d):** Es un método de instancia de la clase **ArbolLg.** El método recibe un parámetro *d* que corresponde a un registro en el árbol. El método recorre el árbol hasta encontrar el registro *d,* pero cada que se pase a un hijo (se apila) el nivel aumenta, y cada vez que se pasa a un padre (se desapila) el nivel disminuye. Cuando encuentra el dato retorna el nivel. Si no encuentra el dato retorna -1
* **ancestrosRegistro(char d):** Es un método de instancia de la clase **ArbolLg.** El método recibe un parámetro *d* que corresponde a un registro en el árbol. El método recorre el árbol en busca del registro *d,* pero mientras lo va recorriendo guarda el rastro de los registros padre en un array con comportamiento LIFO. Una vez encontrado el registro se retorna un String con los ancestros del registro. En caso de no encontrar el registro se retorna ‘El dato no ha sido encontrado’. Cuando el dato es la raíz se retorna ‘El dato no tiene ancestros, es la raíz’

**Nota:** Las funciones nivel, altura y hojas se podrían ejecutar en una sola corrida (en todas las funciones hay que recorrer el árbol) y se podrían guardar los valores en nuevos atributos de clase. En lo personal no sé qué sería más eficiente en términos de escalabilidad de la aplicación ¿Encontrar el nivel, altura y hojas, aunque no se vayan a usar o recorrer el árbol 3 veces cuando se quieran conocer los 3 valores? De igual manera, Las funciones gradoRegistro, nivelRegistro y ancestrosRegistro podrían ser ejecutadas en una sola corrida, aunque esto se salte la “regla” de “única funcionalidad” en los paradigmas de programación. El caso es que funciona 😊.

**Iniciador:**

- **index.js:** Es el archivo Main, es decir, el archivo que ejecuta las funciones. En esta clase se encuentran funciones de **VisualController** y **ArbolLg**. En la clase se adicionan eventos a los elementos HTML para que la página activar sus funciones principales además de tener adicionar eventos que mejoran la experiencia de usuario.