UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE
INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TRABAJO ENCARGADO

PROYECTO FINAL DE SIMULACION

CURSO:

ESTRUCTURA DE DATOS AVANZADAS

DOCENTE:

ING. COLLANQUI MARTINEZ FREDY

PRESENTADO POR:

BELTRAN EDWIN MAMANI MAMANI

PUNO- PERU

2024

Árbol Binario con Eliminación y Búsqueda Visual

El código proporcionado crea una interfaz web para visualizar y manipular un árbol binario. Los usuarios pueden insertar, buscar, eliminar valores y recorrer el árbol en diferentes órdenes. También se visualiza gráficamente el árbol en un canvas HTML.

Declaración del Tipo de Documento y Lenguaje

```
<head>
    <meta  charset= "UTF-8" >
    <meta  name= "viewport"  content= "width=device-width, inicial-scale=1.0"
>
    <title> Árbol Binario con Eliminación y Búsqueda Visual </title>
    <estilo>
        /* Estilos CSS */
    </estilo>
</head>
```

En el <head>, se configura la codificación de caracteres como UTF-8 y se establece el diseño adaptable para dispositivos móviles. También se define el título de la página y se incluyen estilos CSS.

Estilos CSS

```
        cuerpo {
            familia de fuentes: Arial, sans-serif;
            color de fondo: #f5f5f5;
            pantalla:flexible;
            flex-direccion: columna;
            alinear-elementos: centro;
            relleno: 20px;
        }
        #envase {
            color de fondo: #fff;
            relleno: 20px;
            caja-sombra: 0 0 10px rgba(0, 0, 0, 0.1);
            margen inferior: 20px;
        }
        botón {
    margen: 5px;
        }
        #lienzo {
            borde: 1px sólido #000;
        }
```

Los estilos CSS definen el aspecto visual de la página:

- Se utiliza la fuente Arial y se establece un fondo gris claro.

- El contenedor del formulario tiene un fondo blanco y sombra para mejorar la visibilidad.
- Los botones tienen un margen para separarlos.
- El canvas tiene un borde negro para definir claramente el área de dibujo.

Código JavaScript

Definición de la Clase Node

```
1 clase Nodo {
2     constructor(valor) {
3         este .valor = valor;
4          este .izquierda =  nulo ;
5          este .derecha =  nulo ;
6     }
7 }
```

Define la estructura básica del nodo del árbol binario, que incluye un valor y referencias a los hijos izquierdo y derecho.

Definición de la Clase BinaryTree

```
1 clase ÁrbolBinario {
2     constructor() {
3         este .root =  null ;
4     }
5
6     // Métodos de la clase
7 }
```

Define la estructura y métodos para manejar el árbol binario. Incluye métodos para insertar, buscar, eliminar nodos y realizar recorridos.

Métodos de Inserción

```
1           insertar(valor) {
2               const nuevoNodo =  nuevo Nodo(valor);
3               si ( este .raíz ===  null ) {
4                   este .raíz = nuevoNodo;
5               } de lo contrario {
6                   este .insertNode( este .root, newNode);
7               }
8           }
9
10          insertNode(nodo, nuevoNodo) {
11              si (nuevoNodo.valor < nodo.valor) {
12                  si (nodo.izquierda ===  null ) {
13                      nodo.izquierda = nuevoNodo;
14                  } de lo contrario {
```

```
15                  este .insertNode(nodo.left, nuevoNodo);
dieciséis       }
17            } de lo contrario {
18                si (nodo.derecho ===  nulo ) {
19                  nodo.derecho = nuevoNodo;
20                } de lo contrario {
21                  este .insertNode(nodo.derecha, nuevoNodo);
22                }
23            }
24        }
```

Permite insertar un nuevo nodo en el árbol. El método insert se encarga de insertar un nuevo nodo, mientras que insertNode encuentra la ubicación correcta para el nuevo nodo.

Métodos de Búsqueda

```
1         buscar(valor) {
2             devolver  este .searchNode( este .root, valor);
3         }
4
5         searchNode(nodo, valor) {
6             si (nodo ===  nulo ) {
7                 devolver  falso ;
8             }
9             si (valor < nodo.valor) {
10                devuelve  este .searchNode(nodo.izquierda, valor);
11            } de lo contrario  si (valor > nodo.valor) {
12                devuelve  este .searchNode(nodo.derecha, valor);
13            } de lo contrario {
14                devuelve  verdadero ;
15            }
dieciséis }
```

Busca un valor en el árbol. searchNode recursivamente busca el valor en el subárbol correspondiente.

Métodos de Eliminación

```
1         eliminar (valor) {
2             este .root =  este .deleteNode( este .root, valor);
3         }
4
5         deleteNode(nodo, valor) {
6             si (nodo ===  nulo ) {
7                 devolver  nulo ;
8             }
9
10            si (valor < nodo.valor) {
```

```
11              nodo.izquierda =  this.deleteNode (nodo.izquierda, valor);
12               devolver nodo;
13           } de lo contrario  si (valor > nodo.valor) {
14              nodo.derecha =  this.deleteNode (nodo.derecha, valor);
15               return nodo;
dieciséis    } else {
17               // Caso 1: Nodo sin hijos
18              if (node.left ===  null  && node.right ===  null ) {
19                 nodo =  null ;
20                  retorna nodo;
21              }
22
23              // Caso 2: Nodo con un hijo
24              if (node.left ===  null ) {
25                 nodo = nodo.derecho;
26                  devolver nodo;
27              } de lo contrario  si (nodo.derecho ===  nulo ) {
28                 nodo = nodo.izquierda;
29                  devolver nodo;
30              }
31
32              // Caso 3: Nodo con dos hijos
33              const minNode =  this .findMinNode(node.right);
34              nodo.valor = minNodo.valor;
35              nodo.derecha =  this.deleteNode (nodo.derecha,
36        minNodo.valor);
37               return nodo;
38           }
          }
```

Elimina un nodo del árbol. Se manejan tres casos:

- Nodo sin hijos (se elimina directamente).
- Nodo con un hijo (se reemplaza el nodo por su hijo).
- Nodo con dos hijos (se reemplaza por el nodo mínimo del subárbol derecho).

Métodos de Recorrido

```
1          inOrder(nodo, resultado = []) {
 2             if (nodo !==  null ) {
 3                this .inOrder(nodo.left, resultado);
 4                resultado.push(nodo.valor);
 5                este .inOrder(nodo.derecha, resultado);
 6             }
 7             devolver resultado;
 8          }
 9
10          inOrderReverse(nodo, resultado = []) {
11             if (nodo !==  null ) {
```

```
12                 this .inOrderReverse(nodo.right, resultado);
13                 resultado.push(nodo.valor);
14                 este .inOrderReverse(nodo.izquierda, resultado);
15             }
dieciséis      devolver resultado;
17         }
```

Permiten recorrer el árbol en orden ascendente (inOrder) y en orden descendente (inOrderReverse).

Funciones de Interfaz de Usuario

```
   function insertValue() { /* Inserta un valor en el árbol y actualiza la
   visualización */ }
1  function searchValue() { /* Busca un valor en el árbol y visualiza la
2 búsqueda */ }
3  function deleteValue() { /* Elimina un valor del árbol y actualiza la
4 visualización */ }
5  function traverseTree() { /* Muestra el recorrido en orden del árbol */ }
6  function traverseTreeReverse() { /* Muestra el recorrido en orden inverso
   del árbol */ }
    function showMinMax() { / * Muestra el valor mínimo y máximo del árbol */ }
```

Estas funciones manejan las interacciones del usuario, como insertar, buscar y eliminar valores, y muestran los resultados en el elemento <p> con id output.

Funciones de Visualización del Árbol

```
1
2
3
4
5
function drawTree(nodo, x =  400 , y =  40 , nivel =  0 ) { /* Dibuja el árbol en el
lienzo */ }
 function getNodeColor(nodo, nivel) { /* Devuelve el color del nodo basado en
su valor */ }
 function visualizeSearch(node, value, path) { /* Visualiza el proceso de
búsqueda en el árbol */ }
 function HighlightNode(node) { /* Destaca un nodo específico en el lienzo */ }
 function findNodePosition(node, value, x =  400 , y =  40 , nivel =  0 ) { /*
Encuentra la posición del nodo en el lienzo */ }
```

Estas funciones se encargan de dibujar el árbol en el canvas, resaltar nodos durante la búsqueda y encontrar la posición de los nodos en el canvas para la visualización.

Inicialización del Árbol

```
const valoresIniciales = [ 50 , 30 , 70 , 20 , 40 , 60 , 80 ];
initialValues.forEach(valor => árbol.insert(valor));
drawTree(árbol.raíz);
```

Este código proporciona una interfaz completa para manipular y visualizar un árbol binario en una página web. Permite insertar, buscar, eliminar y recorrer el árbol, con una visualización gráfica en un canvas HTML. La implementación muestra claramente cómo los árboles binarios pueden ser manipulados y representados visualmente usando JavaScript