PSEUDOCÓDIGO DE ÁRBOL BINARIO DE BÚSQUEDA

Definición de un nodo binario.

```
Clase NodoBinario
EMPIEZA
        Objeto elemento
        NodoBinario izquierdo
        NodoBinario derecho
        //Constructor.
        NodoBinario (Objeto dato)
        EMPIEZA
        elemento ← dato
        izquierdo ← nil
        derecho ← nil
        TERMINA
TERMINA
```

Definición de un árbol binario de búsqueda.

```
Clase ArbolBinarioBusqueda
EMPIEZA
        NodoBinario raiz
        //Constructor
        ArbolBinarioBusqueda()
        COMIENZA
        raiz ← nil
        TERMINA
TERMINA
```

Operación INSERTAR.

```
NodoBinario insertar(Objeto x, NodoBinario t)
COMIENZA
        SI (t = nil)
                t ← nuevo NodoBinario (x)
        OTRO SI (x < t.elemento)
                t.izquierdo ← insertar(x, t.izquierdo)
        OTRO SI (x > t.elemento)
                t.derecho ← insertar(x, t.derecho)
        OTRO
        "El elemento ya se encuentra"
        REGRESA t
TERMINA
```

Operación BUSCAR.

```
NodoBinario buscar(Objeto x, nodoBinario t)
COMIENZA
        SI (t ← nil)
        COMIENZA
                "El elemento no se encuentra"
                REGRESA nil
        TERMINA
        OTRO SI (x < t.elemento)
        REGRESA buscar(x, t.izquierdo)
        OTRO SI (x > t.elemento)
        REGRESA buscar(x, t.derecho)
        OTRO
        COMIENZA
                "El elemento si se encuentra"
                REGRESA t
        TERMINA
TERMINA
```

Operación BORRAR.

```
NodoBinario borrar(Objeto x, NodoBinario t)
COMIENZA
        SI (t = nil)    //El elemento no se encuentra
        REGRESA t
        OTRO SI (x < t.elemento)
        t.izquierdo ← borrar(x, t.izquierdo)
        OTRO SI (x > t.elemento)
        t.derecho ← borrar(x, t.derecho)
        OTRO SI (t.izquierdo =nil && t.derecho =nil)
        t←nil
        OTRO SI (t.izquierdo <> nil && t.derecho <> nil)
        COMIENZA
                //Utilizando el menor de los mayores
                t.elemento ← buscarMinimo(t.derecho).elemento
                t.derecho ← borrar(t.elemento, t.derecho)
                //Utilizando el mayor de los menores
                t.elemento ← buscarMaximo(t.izquierdo).elemento
                t.izquierdo ← borrar(t.elemento, t.izquierdo)
        TERMINA
        OTRO
        COMIENZA
                SI(t.izquierdo <>nil)
                t ← t.izquierdo
                SI(t.derecho <> nil)
                t ← t.derecho
        TERMINA
        REGRESA t
TERMINA
```

Menor de los mayores.

```
NodoBinario buscarMinimo(nodoBinario t)
COMIENZA
        SI (t=nil)
        REGRESA nil
        OTRO (t.izquierda = nil)
        REGRESA t
        REGRESA buscarMinimo(t.izquierdo)
TERMINA
```

Mayor de los menores.

```
NodoBinario buscarMaximo(nodoBinario t)
COMIENZA
        SI (t=nil)
        REGRESA nil
        OTRO (t.derecha = nil)
        REGRESA t
        REGRESA buscarMaximo(t.derecha)
TERMINA
```

Recorrido en Pre-orden.

```
void imprimirPreOrden(nodoBinario t)
COMIENZA
        SI (t <>nil)
        COMIENZA
                "El elemento es" + t.elemento
                imprimirPreOrden(t.izquierdo)
                imprimirPreOrden(t.derecho)
        TERMINA
TERMINA
```

Recorrido en In-orden.

```
void imprimirInOrden(nodoBinario t)
COMIENZA
        SI (t <>nil)
        COMIENZA
                imprimirInOrden(t.izquierdo)
                "El elemento es" + t.elemento
                imprimirInOrden(t.derecho)
        TERMINA
TERMINA
```

Recorrido en Post-orden.

```
void imprimirPostOrden(nodoBinario t)
COMIENZA
        SI (t <>nil)
        COMIENZA
                imprimirPostOrden(t.izquierdo)
                imprimirPostOrden(t.derecho)
                "El elemento es" + t.elemento
        TERMINA
TERMINA
```

Árbol vacío.

```
boolean vacio()
COMIENZA
        SI (raiz = nil)
        COMIENZA
                regresa verdadero
        TERMINA
        OTRO
        COMIENZA
                regresa falso
        TERMINA
TERMINA
```