UNIVERSIDAD DE LA FRONTERA

Tarea Semana 13

Integrantes: -Gilio Linfati
-Diego Vera
Profesor: -Pablo Valenzuela
Asignatura: -Estructura de Datos

Introducción

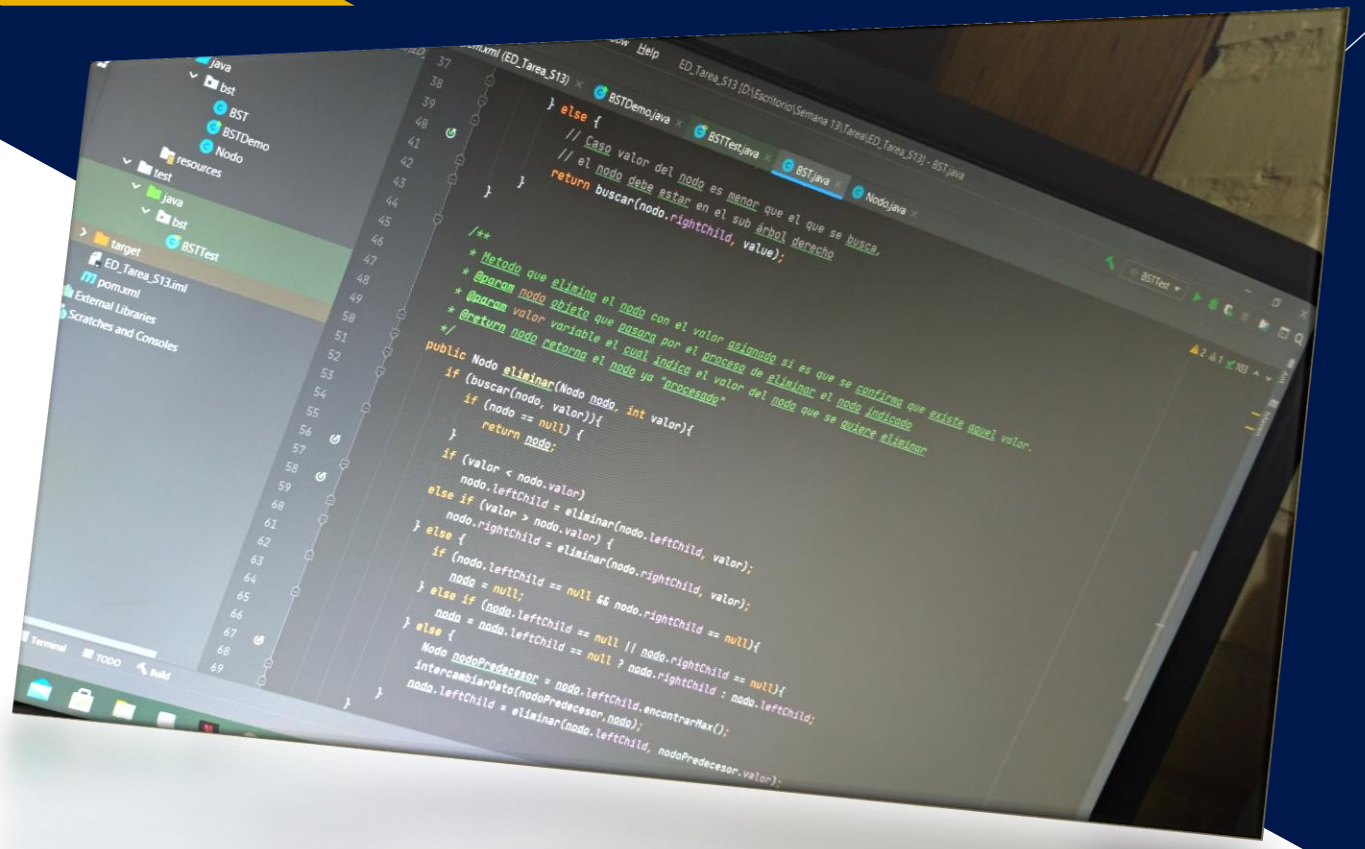
En esta presentación se verán las implementaciones de los métodos eliminar e insertar que se plantearon en ED_Tarea_S1.

Como objetivo se tendrá:

- Revisar el problema dado.
- Mostrar el código de la implementación hecha.
- Explicar el código implementado.

Problema Planteado

Problema 1: este problema debe ser implementado en el proyecto IntelliJ ED_Tarea_S13.zip. La actividad consiste en implementar el código y pruebas unitarias de 2 métodos considerando un árbol binario de búsqueda (BST): 1) eliminar() e insertar().



Código

```java
/**
 * Metodo que consiste en Buscar nodo con determinado valor
 * @param nodo objeto que pasara por el proceso de busqueda
 * @param value variable que se esta buscando
 * @return booleano
 */
public boolean buscar(Nodo nodo, int value) {
    // Caso base: la raíz del BST root es null
    if (nodo == null) {
        return false;
    }
    // Caso base: el valor la raíz del BST root
    // no es null y es igual que valor
    if (nodo.valor == value) {
        return true;
        // Caso valor del nodo es mayor que el que se busca,
        // el nodo debe estar en el sub árbol izquierdo
    } else if (nodo.valor > value) {
        return buscar(nodo.leftChild, value);
    } else {
        // Caso valor del nodo es menor que el que se busca,
        // el nodo debe estar en el sub árbol derecho
        return buscar(nodo.rightChild, value);
    }
}
```

```java
public class Nodo {
    int valor;
    Nodo leftChild;
    Nodo rightChild;

    Nodo(int value) { this.valor = value; }

    /**
     * Metodo que busca el nodo de maximo valor
     * @return el nodo que es maximo en el arbol
     */
    public Nodo encontrarMax(){
        if (rightChild != null){
            return rightChild.encontrarMax();
        }
        return this;
    }
}
```

```java
/**
 * Metodo que elimina el nodo con el valor asignado si es que se confirma que existe aquel valor.
 * @param nodo objeto que pasara por el proceso de eliminar el nodo indicado
 * @param valor variable el cual indica el valor del nodo que se quiere eliminar
 * @return nodo retorna el nodo ya "procesado"
 */
public Nodo eliminar(Nodo nodo, int valor){
    if (buscar(nodo, valor)){
        if (nodo == null) {
            return nodo;
        }
        if (valor < nodo.valor)
            nodo.leftChild = eliminar(nodo.leftChild, valor);
        else if (valor > nodo.valor) {
            nodo.rightChild = eliminar(nodo.rightChild, valor);
        } else {
            if (nodo.leftChild == null && nodo.rightChild == null){
                nodo = null;
            } else if (nodo.leftChild == null || nodo.rightChild == null){
                nodo = nodo.leftChild == null ? nodo.rightChild : nodo.leftChild;
            } else {
                Nodo nodoPredecesor = nodo.leftChild.encontrarMax();
                intercambiarDato(nodoPredecesor,nodo);
                nodo.leftChild = eliminar(nodo.leftChild, nodoPredecesor.valor);
            }
        }
    }
    return nodo;
}
```

```java
/**
 * Metodo que intercambia los valores entre los nodos dados
 * @param nodoA nodo que cambia el valor con el nodoB
 * @param nodoB nodo que cambia el valor con el nodoA
 */
private void intercambiarDato(Nodo nodoA, Nodo nodoB){
    int temp = nodoA.valor;
    nodoA.valor = nodoB.valor;
    nodoB.valor = temp;
}
```

```java
/**
 * Metodo que inserta un nuevo nodo con su respectivo valor, si es que el valor ingresado no existe
 * @param nodo objeto que pasara por el proceso de insertar el nodo dado
 * @param valor varible de tipo int que indica el numero relacionado con el nodo
 * @return nodo retorna el nodo ya "procesado"
 */
public Nodo insertar(Nodo nodo, int valor){
    if (nodo == null) {
        return new Nodo(valor);
    }
    if (!buscar(nodo,valor)) {
        if (valor < nodo.valor)
            nodo.leftChild = insertar(nodo.leftChild, valor);
        else if (valor > nodo.valor)
            nodo.rightChild = insertar(nodo.rightChild, valor);
        return nodo;
    }
    return nodo;
}
```

```java
public class BSTTest {

    BST arbolAgregar;
    BST arbolEliminar;

    @Before
    public void setUp() throws Exception {
        arbolAgregar = new BST();
        arbolEliminar = new BST();

        arbolEliminar.root = arbolEliminar.insertar(arbolEliminar.root, valor: 50);
        arbolEliminar.insertar(arbolEliminar.root, valor: 30);
        arbolEliminar.insertar(arbolEliminar.root, valor: 40);
        arbolEliminar.insertar(arbolEliminar.root, valor: 20);
        arbolEliminar.insertar(arbolEliminar.root, valor: 70);
        arbolEliminar.insertar(arbolEliminar.root, valor: 60);
        arbolEliminar.insertar(arbolEliminar.root, valor: 80);
        arbolEliminar.insertar(arbolEliminar.root, valor: 100);
        arbolEliminar.insertar(arbolEliminar.root, valor: 10);
        arbolEliminar.insertar(arbolEliminar.root, valor: 25);


    }
```

```java
@Test
public void eliminar() {
    Assert.assertTrue(arbolEliminar.buscar(arbolEliminar.root, value: 20));
    Assert.assertTrue(arbolEliminar.buscar(arbolEliminar.root, value: 30));
    Assert.assertTrue(arbolEliminar.buscar(arbolEliminar.root, value: 50));
    arbolEliminar.eliminar(arbolEliminar.root, valor: 20);
    arbolEliminar.eliminar(arbolEliminar.root, valor: 30);
    arbolEliminar.eliminar(arbolEliminar.root, valor: 50);
    Assert.assertFalse(arbolEliminar.buscar(arbolEliminar.root, value: 20));
    Assert.assertFalse(arbolEliminar.buscar(arbolEliminar.root, value: 30));
    Assert.assertFalse(arbolEliminar.buscar(arbolEliminar.root, value: 50));

}
```

```java
@Test
public void insertar() {
    arbolAgregar = new BST();
    Random rng = new Random();
    int numeroAleatorio = rng.nextInt( bound: 150);
    Assert.assertFalse(arbolAgregar.buscar(arbolAgregar.root, numeroAleatorio));
    arbolAgregar.root = arbolAgregar.insertar(arbolAgregar.root, numeroAleatorio);
    Assert.assertTrue(arbolAgregar.buscar(arbolAgregar.root, numeroAleatorio));

    arbolAgregar.insertar(arbolAgregar.root, valor: 200);
    arbolAgregar.insertar(arbolAgregar.root, valor: 500);
    arbolAgregar.insertar(arbolAgregar.root, valor: 300);

    Assert.assertTrue(arbolAgregar.buscar(arbolAgregar.root,  value: 200));
    Assert.assertTrue(arbolAgregar.buscar(arbolAgregar.root,  value: 500));
    Assert.assertTrue(arbolAgregar.buscar(arbolAgregar.root,  value: 300));

}
```

Muchas Gracias por su atención

Referencias

- (2020). Retrieved 3 September 2020, from http://en.ufro.cl/images/news/2018/december/Ufro-institucional-2018.jpg
- (2020). Retrieved 3 September 2020, from https://www.temuco.cl/wp-content/uploads/2018/12/Carpeta-33-A-1er-lugar.jpg
- (2020). Retrieved 3 September 2020, from https://www.pexels.com/photo/black-twist-pen-on-notebook-891059/