



EXAMEN 1

Estructuras de datos no lineales

LEONARDO AGUILAR MARTINEZ

2203025005

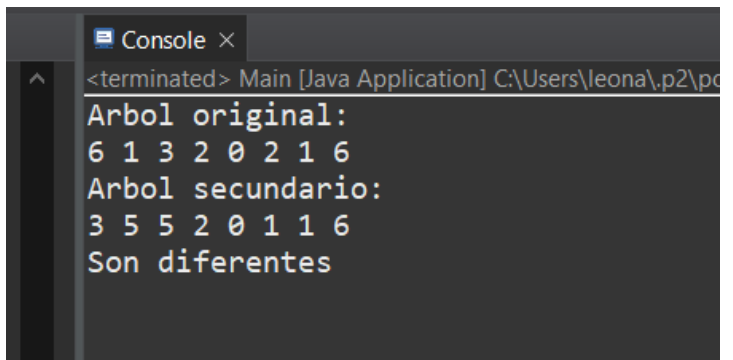
Ejercicio 1: Realiza un programa que dadas dos raíces de dos árboles binarios. Verifiquen si son o no equivalentes. Recuerda que un árbol es equivalente si tienen la misma estructura y datos.

Código:

```
public boolean sonEquivalentes(ArbolBin otroArbol) {  
    return sonEquivalentesRecursivo(this.raiz, otroArbol.raiz);  
}  
  
private boolean sonEquivalentesRecursivo(Nodo nodo1, Nodo nodo2) {  
    if (nodo1 == null && nodo2 == null) {  
        return true;  
    }  
    if (nodo1 == null || nodo2 == null) {  
        return false;  
    }  
    if (nodo1.getDato() != nodo2.getDato()) {  
        return false;  
    }  
    return sonEquivalentesRecursivo(nodo1.getIzq(), nodo2.getIzq()) &&  
        sonEquivalentesRecursivo(nodo1.getDer(), nodo2.getDer());  
}
```

Captura de ejecución:

Lo ejecuté en eclipse dado que Visual Studio tenía problemas para compilar



```
Console x  
<terminated> Main [Java Application] C:\Users\leona\p2\p...  
Arbol original:  
6 1 3 2 0 2 1 6  
Arbol secundario:  
3 5 5 2 0 1 1 6  
Son diferentes
```

Explicación: Tenemos un método sonEquivalentes que recibe como parametro un arbol y ejecuta el método recursivo sonEquivalentesRecursivo. Dentro de éste tenemos 3 condiciones if, éstas sirven para saber si ambos nodos son nulos, equivalentes comparando el nodo 1 con el nodo 2 de los arboles a comparar, si en algún momento se cumple que alguno de los dos nodos no son iguales retornamos falso y salimos.

Ejercicio 2: Escriba una función que reciba como parámetro un árbol A y devuelva una copia de él.

Código:

```
public void imprimirArbol() {
    imprimirArbolRecursivo(raiz, "");
}

private void imprimirArbolRecursivo(Nodo nodo, String prefijo) {
    if (nodo == null) {
        return;
    }
    imprimirArbolRecursivo(nodo.getDer(), prefijo + "    /");
    System.out.println(prefijo + "---" + nodo.getDato());
    imprimirArbolRecursivo(nodo.getIzq(), prefijo + "    /");
}
```

Captura de ejecución:

Se imprime el arbol, pero con base al método de inserción antes implementado, no quedé satisfecho e implementé otro método de inserción para continuar con el ejercicio.

```

Console ×
<terminated> Main [Java Application] C:\Users\leona\p2\p
Arbol original:
6 1 3 2 0 2 1 6
Arbol secundario:
3 5 5 2 0 1 1 6
|           |           |---6
|           |---1
|---1
---3
|---5
|           |---5
|           |           |---2
|           |           |           |---0

```

Explicación:

Tenemos un método llamado `imprimirArbol` el cual ejecuta a su respectivo método recursivo, el cual recibe como parametro un nodo y un prefijo, el cual es un string el cual podrá imprimirse según sea su caso.

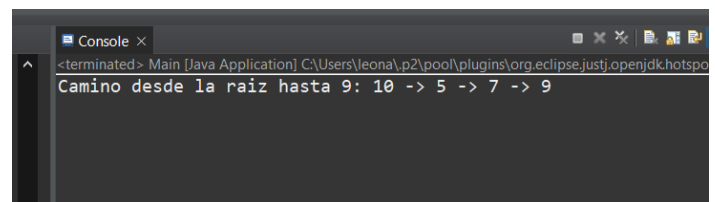
Después de eso tenemos una condicional if la cual desde un inicio si sabemos que el nodo es nulo, retornamos, despues de eso mandamos a llamar al método recursivo para imprimir la parte izquierda y después usamos un print para imprimir dicho prefijo seguido del contenido del nodo con el que vamos transitando por el arbol.

Ejercicio 3:

```
public void imprimirCamino(int valor) {
    List<Integer> camino = new ArrayList<>();
    if (encontrarCamino(raiz, valor, camino)) {
        System.out.print("Camino desde la raíz hasta " + valor + ": ");
        for (int i = 0; i < camino.size() - 1; i++) {
            System.out.print(camino.get(i) + " -> ");
        }
        System.out.println(camino.get(camino.size() - 1));
    } else {
        System.out.println("No se encontró el nodo con el valor " + valor);
    }
}

private boolean encontrarCamino(Nodo nodo, int valor, List<Integer> camino) {
    if (nodo == null) {
        return false;
    }
    camino.add(nodo.getDato());
    if (nodo.getDato() == valor) {
        return true;
    }
    if (encontrarCamino(nodo.getIzq(), valor, camino) ||
    encontrarCamino(nodo.getDer(), valor, camino)) {
        return true;
    }
    camino.remove(camino.size() - 1);
    return false;
}
```

Captura de ejecución:



Explicación: Tenemos una función `imprimirCamino` la cual recibe como parametro el numero a buscar y con ayuda de un `arraylist` podemos almacenar en la lista el camino recorrido de ese nodo en el arbol.

Es importante que la función recursiva retorne `true` o `false` ya que con esto sabremos si podemos almacenar este camino o no.

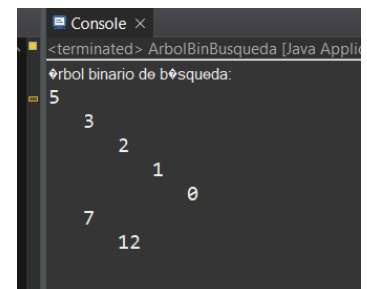
Ejercicio 4: Implementar un árbol binario de búsqueda con valores múltiples:

```
public void insertar(int valor) {
    raiz = insertarRecursivo(raiz, valor);
}

private NodoABB insertarRecursivo(NodoABB nodo, int valor) {
    if (nodo == null) {
        num_nodos++;
        return new NodoABB(valor);
    }
    if (valor < nodo.getValor()) {
        nodo.setIzq(insertarRecursivo(nodo.getIzq(), valor));
    } else if (valor > nodo.getValor()) {
        nodo.setDer(insertarRecursivo(nodo.getDer(), valor));
    } else {
        nodo.agregarValor(valor);
    }
    return nodo;
}
```

Nota: considero que la descripción del problema le faltó un poco más de explicación pero por lo entendido, realicé lo siguiente.

Captura de ejecución:



Explicación: A grandes rasgos podemos decir que tenemos dos funciones: insertar y su respectiva recursiva, en insertar solo ejecutamos el metodo recursivo.

En el método recursivo recibimos un nodo y un valor como parametro, si está vacío el arbol, agregamos uno a la raíz, de lo contrario haremos comparaciones entre los valores de los nodos ya que al ser un arbol binario de busqueda, buscamos que siempre los valores menores estén en la izquierda y los mayores a la derecha.