INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	**Programacion Web 2**				
TÍTULO DE LA PRÁCTICA:	***Laboratorio 05***				
NÚMERO DE PRÁCTICA:	***5***	**AÑO LECTIVO:**	***2025***	**NRO. SEMESTRE:**	***1***
FECHA DE PRESENTACIÓN	***21/05/2024***	**Repositorio**	*https://github.com/code50/82924112.git*		
INTEGRANTE (s): **Silva Pino Jesus Francisco**				**NOTA:**	
DOCENTE(s): **Edson Luque Mamani**					

SOLUCIÓN Y RESULTADOS

I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS

EJERCICIOS PROPUESTOS

Parte 01 (clase)

- Elabore un informe implementando Arboles de Búsqueda Binarios con toda la lista de operaciones:
  - search(), getMin(), getMax(), parent(), son(), insert().
  - INPUT: Una sóla palabra en mayúsculas.
  - OUTPUT: Se debe contruir el BST considerando el valor decimal de su código ascii.
  - Luego, pruebe todas sus operaciones implementadas.

- Codigo Utilizado

Clase Nodo

```
package Laboratorio4;

/**
* Nodo genérico para el Árbol Binario de Búsqueda.
* @param <T> El tipo de dato que almacenará el nodo.
*/
public class Nodo<T> {
T dato;
Nodo<T> izquierdo;
Nodo<T> derecho;

public Nodo(T dato) {
this.dato = dato;
this.izquierdo = null;
this.derecho = null;
}
}
```

Clase BST

```
package Laboratorio4;

import java.util.ArrayList;
import java.util.List;

/**
* Implementación de un Árbol Binario de Búsqueda (BST) genérico.
* @param <T> El tipo de dato, que debe ser comparable.
*/
public class BST<T extends Comparable<T>> {

private Nodo<T> raiz;

public BST() {
this.raiz = null;
}

public void insert(T dato) {
```

```
this.raiz = insertRec(this.raiz, dato);
}

private Nodo<T> insertRec(Nodo<T> actual, T dato) {
if (actual == null) {
return new Nodo<>(dato);
}

// Si el dato es menor, va al subárbol izquierdo.
if (dato.compareTo(actual.dato) < 0) {
actual.izquierdo = insertRec(actual.izquierdo, dato);
// Si el dato es mayor, va al subárbol derecho.
} else if (dato.compareTo(actual.dato) > 0) {
actual.derecho = insertRec(actual.derecho, dato);
}

return actual;
}

public boolean search(T dato) {
return searchRec(this.raiz, dato);
}

private boolean searchRec(Nodo<T> actual, T dato) {
if (actual == null) {
return false;
}
if (dato.equals(actual.dato)) {
return true;
}
return dato.compareTo(actual.dato) < 0
? searchRec(actual.izquierdo, dato)
: searchRec(actual.derecho, dato);
}

public T getMin() {
if (raiz == null) {
return null;
}
Nodo<T> actual = raiz;
while (actual.izquierdo != null) {
actual = actual.izquierdo;
```

```
}
return actual.dato;
}

public T getMax() {
if (raiz == null) {
return null;
}
Nodo<T> actual = raiz;
while (actual.derecho != null) {
actual = actual.derecho;
}
return actual.dato;
}

public T parent(T dato) {
return parentRec(raiz, dato, null);
}

private T parentRec(Nodo<T> actual, T dato, Nodo<T> padre) {
if (actual == null) {
return null; // El dato no está en el árbol
}

// Si encontramos el dato, retornamos el dato del padre.
if (dato.equals(actual.dato)) {
return (padre != null) ? padre.dato : null; // La raíz no tiene padre.
}

// Búsqueda recursiva
if (dato.compareTo(actual.dato) < 0) {
return parentRec(actual.izquierdo, dato, actual);
} else {
return parentRec(actual.derecho, dato, actual);
}
}

public List<T> getChildren(T dato) {
Nodo<T> nodo = findNode(this.raiz, dato);
List<T> hijos = new ArrayList<>();

if (nodo != null) {
if (nodo.izquierdo != null) {
```

```
hijos.add(nodo.izquierdo.dato);
}
if (nodo.derecho != null) {
hijos.add(nodo.derecho.dato);
}
}
return hijos;
}

private Nodo<T> findNode(Nodo<T> actual, T dato) {
if (actual == null || dato.equals(actual.dato)) {
return actual;
}

if (dato.compareTo(actual.dato) < 0) {
return findNode(actual.izquierdo, dato);
} else {
return findNode(actual.derecho, dato);
}
}
}
```

Clase prueba

```
package Laboratorio4;

import java.util.List;

public class LaboratorioBST {

public static void main(String[] args) {

String palabra = "ESTRUCTURA";
BST<Integer> arbol = new BST<>();

System.out.println("Construyendo el árbol con la palabra: " + palabra);
System.out.print("Valores ASCII insertados: ");

for (char c : palabra.toCharArray()) {
int ascii = (int) c;
System.out.print(ascii + " ");
```

```
arbol.insert(ascii);
}
System.out.println("\nÁrbol construido.\n");

System.out.println("--- Probando Operaciones Implementadas ---");

int valorExistente = (int) 'R';
int valorInexistente = 100; //"D"
System.out.println("1. Búsqueda (search):");
System.out.println(" - ¿Se encuentra el valor " + valorExistente + "? -> " + arbol.search(valorExistente));
System.out.println(" - ¿Se encuentra el valor " + valorInexistente + "? -> " +
arbol.search(valorInexistente));
System.out.println();

System.out.println("2. Mínimo y Máximo:");
System.out.println(" - Valor mínimo (getMin): " + arbol.getMin());
System.out.println(" - Valor máximo (getMax): " + arbol.getMax());
System.out.println();

int valorHijo = (int) 'C';
int valorRaiz = (int) 'E';
System.out.println("3. Padre (parent):");
System.out.println(" - Padre de " + valorHijo + ": " + arbol.parent(valorHijo));
```

```
System.out.println(" - Padre de la raíz (" + valorRaiz + "): " + arbol.parent(valorRaiz));
System.out.println();

int nodoConDosHijos = (int) 'U';
int nodoHoja = (int) 'A';
System.out.println("4. Hijos (getChildren / son):");
System.out.println(" - Hijos de " + nodoConDosHijos + ": " + arbol.getChildren(nodoConDosHijos));
System.out.println(" - Hijos de " + valorRaiz + ": " + arbol.getChildren(valorRaiz));
System.out.println(" - Hijos de una hoja (" + nodoHoja + "): " + arbol.getChildren(nodoHoja));
}
}
```

Salida

```
$  /usr/bin/env /opt/jdk/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /home/ubuntu/.vscode-remote/data/User/workspaceStorage/-2f00f915/redhat.java/jdt_ws/829241
12_3dd81b53/bin Laboratorio4.LaboratorioBST
Construyendo el árbol con la palabra: ESTRUCTURA
Valores ASCII insertados: 69 83 84 82 85 67 84 85 82 65
Árbol construido.

--- Probando Operaciones Implementadas ---
1. Búsqueda (search):
   - ¿Se encuentra el valor 82? -> true
   - ¿Se encuentra el valor 100? -> false

2. Mínimo y Máximo:
   - Valor mínimo (getMin): 65
   - Valor máximo (getMax): 85

3. Padre (parent):
   - Padre de 67: 69
   - Padre de la raíz (69): null

4. Hijos (getChildren / son):
   - Hijos de 85: []
   - Hijos de 69: [67, 83]
   - Hijos de una hoja (65): []
$
```

II. SOLUCIÓN DEL CUESTIONARIO

III. CONCLUSIONES

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA