# ESTRUCTURAS DE DATOS

2024

Trabajo Práctico N°7

Arbol B


Profesores:

HECTOR REINAGA

FERNANDA DANIELA OYARZO

MIRTHA FABIANA MIRANDA

Alumno:

GONZALO ALEJANDRO ULLOA


# Índice

# Desarrollo

## 3.

Clase Arbol B

```
package tp7;
import java.util.*;
public class ArbolB {
    NodoB raiz;
    int aridad;
    public ArbolB(int aridad){
        this.aridad = aridad + 1;
        raiz = null;
    }
    public void agregarLlave(Object llave){
        if (raiz == null){
            NodoB nuevoNodo = new NodoB(aridad + 1);
            Llaves llaves = new Llaves(aridad);
            llaves.agregarLlave(llave);
            nuevoNodo.cambiarDato(llaves);
            raiz = nuevoNodo;
        }
        NodoB nodoActual = raiz;
        Llaves llavesActuales = raiz.obtenerDato();
        int indice;
        Stack nodos = new Stack();
        while (nodoActual.grado() > 0){
            if (llave instanceof String){
                indice = llavesActuales.buscar((String) llave);
            }else {
                indice = llavesActuales.buscar((Integer) llave);
            }
            if (indice == -1){
                return;
            }
            nodos.push(nodoActual);
            nodoActual = nodoActual.obtenerHijo(indice);
            llavesActuales = nodoActual.obtenerDato();
        }
        if (llave instanceof String){
            indice = llavesActuales.buscar((String) llave);
```

```
        }else {
            indice = llavesActuales.buscar((Integer) llave);
        }
        if (indice == -1){
            return;
        }
        llavesActuales.agregarLlave(llave);
        while (llavesActuales.size == aridad){
            int medio = (aridad + 1) / 2 - 1;
            llave = llavesActuales.llaveEn(medio);
            llavesActuales.borrarLlave(llave);
            NodoB nuevoNodo = new NodoB(aridad + 1);
            Llaves nuevasLlaves = new Llaves(aridad);
            nuevoNodo.cambiarDato(nuevasLlaves);
            for (int i = 0; i < (aridad + 1) / 2 - 1; i++){
                Object llaveTemp = llavesActuales.llaveEn(0);
                nuevasLlaves.agregarLlave(llaveTemp);
                llavesActuales.borrarLlave(llaveTemp);
            }
            if (nodoActual.grado() > 0){
                for (int i = 0; i <= (aridad + 1) / 2 - 1; i++){
                    NodoB hijo = nodoActual.obtenerHijo(0);
                    nuevoNodo.agregarHijo(hijo, i);
                    nodoActual.borrarHijo(0);
                }
            }
            if (nodoActual == raiz){
                raiz = new NodoB(aridad + 1);
                Llaves llaveTemps = new Llaves(aridad);
                llaveTemps.agregarLlave(llave);
                raiz.cambiarDato(llaveTemps);
                raiz.agregarHijo(nuevoNodo, 0);
                raiz.agregarHijo(nodoActual, 1);
                return;

            }

            nodoActual = (NodoB) nodos.pop();

            llavesActuales = nodoActual.obtenerDato();
            indice = llavesActuales.agregarLlave(llave);
```

```
            nodoActual.agregarHijo(nuevoNodo, indice);
        }
    }

    public boolean buscarLlave(Object llave){
        int comparaciones = 0;
        if (raiz == null){
            return false;
        }
        NodoB nodoActual = raiz;
        Llaves llavesActuales = raiz.obtenerDato();
        int indice;
        if (llave instanceof String){
            indice = llavesActuales.buscar((String) llave);
        }else {
            indice = llavesActuales.buscar((Integer) llave);
        }
        while (nodoActual.grado() > 0){
            if (indice == -1){
                if (llave instanceof String){
                    comparaciones = comparaciones +
llavesActuales.buscar((String) llave) + 1;
                } else {
                    comparaciones = comparaciones +
llavesActuales.buscar((Integer) llave) + 1;
                    return true;
                }
            }
            comparaciones = comparaciones + llavesActuales.size;
            nodoActual = nodoActual.obtenerHijo(indice);
            llavesActuales = nodoActual.obtenerDato();
            if (llave instanceof String){
                indice = llavesActuales.buscar((String) llave);
            }else {
                indice = llavesActuales.buscar((Integer) llave);
            }
        }
        if (indice != -1){
            comparaciones = comparaciones + llavesActuales.size;
            return false;
        } else {
```

```
            if (llave instanceof String){
                comparaciones = comparaciones
                        + llavesActuales.buscar((String) llave) +
1;
            }else {
                comparaciones = comparaciones
                        + llavesActuales.buscar((Integer) llave) +
1;
            }
            return true;
        }
    }

    public boolean borrarLlave(Object llave){
        if (raiz == null){
            return false;
        }
        NodoB nodoActual = raiz;
        Llaves llavesActuales = raiz.obtenerDato();
        int indice;
        if (llave instanceof String){
            indice = llavesActuales.buscar((String) llave);
        }else {
            indice = llavesActuales.buscar((Integer) llave);
        }
        Stack nodos = new Stack();
        while (nodoActual.grado() > 0){
            if (indice == -1){
                break;
            }
            nodos.push(nodoActual);
            nodoActual = nodoActual.obtenerHijo(indice);
            llavesActuales = nodoActual.obtenerDato();
            if (llave instanceof String){
                indice = llavesActuales.buscar((String) llave);
            }else {
                indice = llavesActuales.buscar((Integer) llave);
            }
        }
        if (indice == -1 && nodoActual.grado() > 0){
            llavesActuales.borrarLlave(llave);
```

```
            if (llave instanceof String){
                indice = llavesActuales.buscar((String) llave);
            }else {
                indice = llavesActuales.buscar((Integer) llave);
            }
            nodos.push(nodoActual);
            Llaves llaveTemps;
            if (indice < nodoActual.grado() - 1){
                nodoActual = nodoActual.obtenerHijo(indice + 1);
                while (nodoActual.grado() > 0){
                    nodos.push(nodoActual);
                    nodoActual.obtenerHijo(0);
                    nodoActual = nodoActual.obtenerHijo(0);
                }
                llaveTemps = nodoActual.obtenerDato();
                llave = llaveTemps.llaveEn(0);
            } else {
                nodoActual = nodoActual.obtenerHijo(indice);
                while (nodoActual.grado() > 0){
                    nodos.push(nodoActual);
                    nodoActual =
nodoActual.obtenerHijo(nodoActual.grado() - 1);
                }
                llaveTemps = nodoActual.obtenerDato();
                llave = llaveTemps.llaveEn(llaveTemps.size - 1);
            }
            llavesActuales.agregarLlave(llave);
            llaveTemps.borrarLlave(llave);
            llavesActuales = llaveTemps;
        } else {
            if (indice == -1){
                llavesActuales.borrarLlave(llave);
            }else {
                return false;

                    }}
        while (nodoActual != raiz
                && llavesActuales.size < (aridad + 1) / 2 - 1){
            NodoB padre = (NodoB) nodos.peek();
            nodos.pop();
            Llaves llavesPadre = padre.obtenerDato();
```

```
            for (int i = 0; i < padre.grado(); i++){
                if (nodoActual == padre.obtenerHijo(i)){
                    indice = i;
                    break;
                }
            }
            if (indice > 0){
                NodoB hermanoIzq = padre.obtenerHijo(indice - 1);
                Llaves llavesIzq = hermanoIzq.obtenerDato();
                Object llaveTemp = llavesPadre.llaveEn(indice -
1);
                llavesPadre.borrarLlave(llaveTemp);
                if (llavesIzq.size >= (aridad + 1) / 2){ //right
rotation
                    Object llaveMovida = llavesIzq
                            .llaveEn(llavesIzq.size - 1);
                    llavesIzq.borrarLlave(llaveMovida);
                    llavesPadre.agregarLlave(llaveMovida);
                    llavesActuales.agregarLlave(llaveTemp);
                    if (nodoActual.grado() > 0){
                        NodoB nodoMovido =
hermanoIzq.obtenerHijo(hermanoIzq
                                .grado() - 1);
                        hermanoIzq.borrarHijo(nodoMovido);
                        nodoActual.agregarHijo(nodoMovido, 0);
                    }
                    return true;
                } else { //node merge
                    llavesIzq.agregarLlave(llaveTemp);
                    for (int i = 0; i < llavesActuales.size; i++){

llavesIzq.agregarLlave(llavesActuales.llaveEn(i));
                        if (nodoActual.grado() > 0){
                            NodoB nodoTemp =
nodoActual.obtenerHijo(i);
                            hermanoIzq
                                    .agregarHijo(nodoTemp,
hermanoIzq.grado());
                        }
                    }
                    if (nodoActual.grado() > 0){
```

```
                        hermanoIzq.agregarHijo(nodoActual
                                .obtenerHijo(nodoActual.grado() -
1));
                    }
                    padre.borrarHijo(indice);
                    nodoActual = padre;
                    llavesActuales = llavesPadre;
                    continue;
                }
            } else {
                NodoB hermanoDer = padre.obtenerHijo(indice + 1);
                Llaves llaveDer = hermanoDer.obtenerDato();
                Object llaveTemp = llavesPadre.llaveEn(indice);
                llavesPadre.borrarLlave(llaveTemp);
                if (llaveDer.size >= (aridad + 1) / 2){
                    Object llaveMovida = llaveDer.llaveEn(0);
                    llaveDer.borrarLlave(llaveMovida);
                    llavesPadre.agregarLlave(llaveMovida);
                    llavesActuales.agregarLlave(llaveTemp);
                    if (nodoActual.grado() > 0){
                        NodoB nodoMovido =
hermanoDer.obtenerHijo(0);
                        hermanoDer.borrarHijo(nodoMovido);
                        nodoActual.agregarHijo(nodoMovido,
nodoActual.grado());
                    }
                    return true;
                } else {
                    llavesActuales.agregarLlave(llaveTemp);
                    for (int i = 0; i < llaveDer.size; i++){

llavesActuales.agregarLlave(llaveDer.llaveEn(i));
                        if (nodoActual.grado() > 0){
                            NodoB nodoTemp =
hermanoDer.obtenerHijo(i);
                            nodoActual.agregarHijo(nodoTemp);
                        }
                    }
                    if (nodoActual.grado() > 0){
                        nodoActual.agregarHijo(hermanoDer
```

```
                                .obtenerHijo(hermanoDer.grado() -
1));
                    }
                    padre.borrarHijo(indice + 1);
                    nodoActual = padre;
                    llavesActuales = llavesPadre;
                    continue;
                }
            }
        }//while
        if (nodoActual == raiz){
            if (raiz.obtenerDato().size == 0){
                if (raiz.grado() > 0){
                    raiz = raiz.obtenerHijo(0);
                }else {
                    raiz = null;
                }
            }
        }
        return true;
    }
    NodoB obtenerRaiz(){
        return raiz;
    }

    void elementoRaiz(){
        Llaves llave = raiz.obtenerDato();
        for (int i = 0; i < llave.size; i++){
            System.out.print(llave.llaveEn(i) + " ");
        }
    }

    public void muestra(){
        Transversal nodos = new Transversal(this);

        System.out.println("\nArbolB:");
        while (nodos.hasMoreElements()){
            NodoB tempNode = (NodoB) nodos.nextElement();
            Llaves llave = tempNode.obtenerDato();

            for (int i = 0; i < llave.size; i++){
```

```
                System.out.print(llave.llaveEn(i) + " ");
            }

        }
    }

    public void muestraValor(int a, int c){
        Transversal nodos = new Transversal(this);

        System.out.println("\nArbolB:");
        while (nodos.hasMoreElements()){
            NodoB tempNode = (NodoB) nodos.nextElement();
            Llaves llave = tempNode.obtenerDato();

            for (int i = 0; i < llave.size; i++){
                if ((Integer) llave.llaveEn(i) < c && (Integer)
llave.llaveEn(i) > a){
                    System.out.print(llave.llaveEn(i) + " ");
                }
            }
        }
    }

    public void muestraHojas(){
        Transversal nodos = new Transversal(this);

        System.out.println("\nArbolB: Hojas");
        while (nodos.hasMoreElements()){
            NodoB tempNode = (NodoB) nodos.nextElement();
            //
            if (tempNode.esHoja()){
                Llaves llave = tempNode.obtenerDato();
                for (int i = 0; i < llave.size; i++){
                    System.out.print(llave.llaveEn(i) + " ");
                }
            }

        }
    }
    public void muestraNoHojas(){
        Transversal nodos = new Transversal(this);
```

```
        System.out.println("\nArbolB: No hojas");
        while (nodos.hasMoreElements()){
            NodoB tempNode = (NodoB) nodos.nextElement();
            //
            if (!tempNode.esHoja()){
                Llaves llave = tempNode.obtenerDato();
                for (int i = 0; i < llave.size; i++){
                    System.out.print(llave.llaveEn(i) + " ");
                }
            }

        }
    }
    public int cantidadLlaves(){
        Transversal nodos = new Transversal(this);
        int cont = 0;
        while (nodos.hasMoreElements()){
            NodoB tempNode = (NodoB) nodos.nextElement();
            cont = cont +
tempNode.obtenerDato().cantidadElementos();
        }
        return cont;
    }
    public int cantidadNodos(){
        Transversal nodos = new Transversal(this);

        int cont = 0;
        while (nodos.hasMoreElements()){
            cont++;
            nodos.nextElement();
        }
        return cont;
    }
}
```

Clase NodoB

```
package tp7;
public class NodoB {
    Llaves dato;
    NodoB[] hijos;
```

```
    int aridad;
    int size;
    NodoB(int aridad) {
        this.aridad = aridad;
        size = 0;
        hijos = new NodoB[aridad];
    }
    NodoB(int aridad, Llaves dato) {
        this.aridad = aridad;
        size = 0;
        hijos = new NodoB[aridad];
        this.dato = dato;
    }
    void cambiarDato(Llaves dato) {
        this.dato = dato;
    }
    Llaves obtenerDato() {
        return dato;
    }
    NodoB obtenerHijo(int indice) {
        if (indice >= size || indice < 0) {
            return null;
        }
        return (NodoB) hijos[indice];
    }
    NodoB agregarHijo(NodoB dato) {
        if (size == aridad) {
            return null;
        }else {
            Llaves temp = dato.obtenerDato();
            NodoB nodoTemp = new NodoB(aridad, temp);
            hijos[size++] = dato;
            return nodoTemp;
        }
    }
    NodoB agregarHijo(NodoB dato, int indice) {
        Llaves temp = dato.obtenerDato();
        if (indice < 0 || indice >= size) {
            return agregarHijo(dato);
        }
        for (int i = size; i > indice; i--) {
```

```
            hijos[i] = hijos[i - 1];
        }
        NodoB nodoTemp = new NodoB(aridad, temp);
        //hijos[indice]=nodoTemp;
        hijos[indice] = dato;
        size++;
        return nodoTemp;
    }
    NodoB borrarHijo(int indice) {
        if (indice < 0 || indice >= size) {
            return null;
        }
        NodoB nodoTemp = (NodoB) hijos[indice];
        for (int i = indice + 1; i <= size - 1; i++) {
            hijos[i - 1] = hijos[i];
        }
        size--;
        return nodoTemp;
    }
    void borrarHijo(NodoB dato) {
        if (size > 0) {
            int j;
            for (int i = 0; i < size; i++) {
                if ((NodoB) hijos[i] == dato) {
                    borrarHijo(i);
                }
            }
        }
    }
    int grado() {
        return size;
    }
    public boolean esHoja() {
        return (hijos[0] == null);
    }
}
```

Clase Llaves

```
package tp7;

public class Llaves {
```

```
    int aridad, size;
    Object[] llaves;
    Llaves(int aridad) {
        this.aridad = aridad;
        size = 0;
        llaves = new Object[aridad * 2];
    }
    Object llaveEn(int indice) {
        if (indice < 0 || indice >= size) {
            return null;
        }
        return llaves[indice];
    }
    int buscar(String llave) {
        for (int i = 0; i < size; i++) {
            //int c=llave.compareTo(llaves[i]);
            int c = llave.compareTo(llaves[i].toString());
            if (c < 0) {
                return i;
            }
            if (c == 0) {
                return -1;
            }
        }
        return size;
    }
    int buscar(Integer llave) {
        for (int i = 0; i < size; i++) {
            //int c=llave.compareTo(llaves[i]);
            int c = llave.compareTo((Integer) llaves[i]);
            if (c < 0) {
                return i;
            }
            if (c == 0) {
                return -1;
            }
        }
        return size;
    }
    int agregarLlave(Object llave) {
        if (size < aridad) {
```

```
            int indice;
            if (llave instanceof String) {
                indice = buscar((String) llave);
            } else {
                indice = buscar((Integer) llave);
            }
            if (indice == -1) {
                return -1;
            }
            for (int k = size; k > indice; k--) {
                llaves[k] = llaves[k - 1];
            }
            llaves[indice] = llave;
            size++;
            return indice;
        } else {
            return -1;
        }
    }
    boolean borrarLlave(Object llave) {
        int indice = -1;
        for (int i = 0; i < size; i++) {
            int c;
            if (llave instanceof String) {
                c = ((String) llave).compareTo((String)
llaves[i]);
            }else {
                c = ((Integer) llave).compareTo((Integer)
llaves[i]);
            }
            if (c < 0) {
                return false;
            }
            if (c == 0) {
                indice = i;
                break;
            }
        }
        if (indice == -1) {
            return false;
        }
```

```
        for (int j = indice + 1; j <= size; j++) {
            llaves[j - 1] = llaves[j];
        }
        size--;
        return true;
    }
    public void mostrarLlaves() {
        int i = 0;
        if (llaves[i] instanceof Integer) {
            Integer n;
            while (llaves[i] != null) {
                n = (Integer) llaves[i];
                System.out.print(n.intValue() + " ");
                i++;
            }
        } else if (llaves[i] instanceof String) {
            String l;
            while (llaves[i] != null) {
                l = (String) llaves[i];
                System.out.print(l + " ");
                i++;
            }
        }
    }
    public int cantidadElementos() {
        int i = 0;
        while (llaves[i] != null) {
            i++;
        }
        return i;
    }
}
```

Clase Transversal

```
package tp7;
import java.util.*;
class Transversal implements Enumeration {
    private Vector nodes;
    public Transversal(ArbolB AB) {
        nodes=new Vector();
        if(AB.raiz!=null){
```

```
            nodes.addElement(AB.raiz);
        }
        }
    public boolean hasMoreElements() {
        return (nodes.size()!=0);
    }
    public Object nextElement() {
        NodoB tempNode=(NodoB)nodes.elementAt(0);
        nodes.removeElementAt(0);
        for(int i=0; i<tempNode.grado(); i++){
            nodes.addElement(tempNode.obtenerHijo(i));
        }
        return tempNode;
    }
}
```

Main

```
package tp7;

public class Main {
    public static void main(String[] args) {
        ArbolB arbol=new ArbolB(5);
        boolean creado=false;
        int opcion;
        do{
            System.out.println();
            System.out.println("============MENU============");
            System.out.println("1-Crear Arbol");
            System.out.println("0-Finalizar");
            opcion=Console.readInt("Ingrese una opcion: ");
            switch (opcion) {
                case 1 -> {
                    System.out.println("----------Nuevo
arbol---------");
                    int aridad=Console.readInt("Ingrese el
orden(aridad) del arbol: ");
                    arbol=new ArbolB(aridad);
                    System.out.println("\n======Arbol predefinido
creado======");
                    creado=true;
                    arbol.agregarLlave(100);
```

```
                    arbol.agregarLlave(20);
                    arbol.agregarLlave(67);
                    arbol.agregarLlave(150);
                    arbol.agregarLlave(15);
                    arbol.agregarLlave(27);
                    arbol.agregarLlave(45);
                    arbol.agregarLlave(120);
                    arbol.agregarLlave(142);
                    arbol.agregarLlave(25);
                    arbol.agregarLlave(30);
                    arbol.agregarLlave(90);
                    arbol.agregarLlave(101);
                    arbol.agregarLlave(16);
                    arbol.agregarLlave(46);
                    System.out.println();
                }

                default -> {
                    if(opcion!=0){
                        System.out.println("Error, Ingrese un
numero correcto");
                    }
                    System.out.println();
                }
            }
        }while(!creado && opcion!=0);
        if (creado)
            menus(opcion, arbol);

    }
    public static void menus(int opcion, ArbolB arbol){
        do{
            System.out.println();
            System.out.println("============MENU============");
            System.out.println("1-Nuevo Arbol");
            System.out.println("2-Insertar");
            System.out.println("3-Borrar");
            System.out.println("4-Buscar");
            System.out.println("5-Mostrar");
            System.out.println("0-Finalizar");
            opcion=Console.readInt("Ingrese una opcion: ");
```

```
            System.out.println();
            switch(opcion){
            case 1 -> {
                System.out.println("----------Nuevo
arbol---------");
                int aridad=Console.readInt("Ingrese el
orden(aridad) del arbol: ");
                arbol=new ArbolB(aridad);
                int op;
                do{
                    System.out.println("1-Insertar un elemento");
                    System.out.println("0-Finalizar");
                    op=Console.readInt("Ingrese una opcion: ");
                    if(op==1){
                        int num=Console.readInt("Ingrese numero a
agregar en el nuevo arbol: ");
                        arbol.agregarLlave(num);
                    }
                }while(op!=0);
                System.out.println("\n======Arbol creado======");
                System.out.println();
            }
            case 2 -> {
                System.out.println("----------Insertar
Numero---------");
                arbol.agregarLlave(Console.readInt("Ingrese el
numero que quiera agregar: "));
                System.out.println("Numero agregado\n");
            }
            case 3 -> {
                System.out.println("----------Borrar
Numero---------");
                int num=Console.readInt("Ingrese el numero a
borrar:    ");
                if(arbol.borrarLlave(num)){
                    System.out.println("Numero: "+num+" borrado
exitosamente!");
                }else{
                    System.out.println("El numero: "+num+" no se
encuentra en el arbol");
                }
```

```
                System.out.println();
            }
            case 4 -> {
                System.out.println("----------Buscar
Numero---------");
                int num=Console.readInt("Ingrese el numero a
buscar:     ");
                if(arbol.buscarLlave(num)){
                    System.out.println("El numero: "+num+" se
encuentra en el arbol.");
                }else{
                    System.out.println("El numero: "+num+" no se
encuentra en el arbol");
                }
                System.out.println();
            }
            case 5 -> {
                System.out.println("----------Mostrar
Arbol---------");
                arbol.muestra();
                System.out.println();
            }
            default -> {
                if(opcion!=0){
                    System.out.println("Error, Ingrese un numero
correcto");
                }
                System.out.println();
            }
            }
        }while(opcion!=0);
    }
}
```

Resultado

```
0c7ad\bin' 'tp7.Main'

============MENU============
1-Crear Arbol
0-Finalizar
Ingrese una opcion:  1
----------Nuevo arbol---------
Ingrese el orden(aridad) del arbol:  5

======Arbol predefinido creado======


============MENU============
1-Nuevo Arbol
2-Insertar
3-Borrar
4-Buscar
5-Mostrar
0-Finalizar
Ingrese una opcion:  5

----------Mostrar Arbol---------

ArbolB:
27 100 15 16 20 25 30 45 46 67 90 101 120 142 150

============MENU============
1-Nuevo Arbol
2-Insertar
3-Borrar
4-Buscar
5-Mostrar
0-Finalizar
Ingrese una opcion:
```

```
===========MENU===========
1-Nuevo Arbol
2-Insertar
3-Borrar
4-Buscar
5-Mostrar
0-Finalizar
Ingrese una opcion:  1

----------Nuevo arbol---------
Ingrese el orden(aridad) del arbol:  5
1-Insertar un elemento
0-Finalizar
Ingrese una opcion:  1
Ingrese numero a agregar en el nuevo arbol:  10
1-Insertar un elemento
0-Finalizar
Ingrese una opcion:  1
Ingrese numero a agregar en el nuevo arbol:  20
1-Insertar un elemento
0-Finalizar
Ingrese una opcion:  1
Ingrese numero a agregar en el nuevo arbol:  30
1-Insertar un elemento
0-Finalizar
Ingrese una opcion:  1
Ingrese numero a agregar en el nuevo arbol:  40
1-Insertar un elemento
0-Finalizar
Ingrese una opcion:  1
Ingrese numero a agregar en el nuevo arbol:  25
1-Insertar un elemento
0-Finalizar
Ingrese una opcion:  0

======Arbol creado======
```