

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p align="center">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p align="center">Código: GUIA-PRLE-001</p>	<p align="right">Página: 1</p>

INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	ÁRBOL BINARIO B y B+				
NÚMERO DE PRÁCTICA:	08	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	Tercero III
FECHA DE PRESENTACIÓN	08/07/2025	HORA DE PRESENTACIÓN	23:59		
INTEGRANTE (s): Davila Flores Mathias Dario				NOTA:	
DOCENTE(s): <ul style="list-style-type: none"> Mg. Ing. Rene Alonso Nieto Valencia. ENLACE GITHUB: https://github.com/mathiasddf/LabsEDA 					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <p align="center">a. Ejercicios Resueltos:</p> <p align="center">i. Ejercicio 1: Implementación BNode Genérico</p>

```
Lab8 / Ej. Resueltos / BNode.java / ...
5 public class BNode<E extends Comparable<E>> {
7     protected ArrayList<BNode<E>> childs;
8     protected int count;
9
10    public BNode(int n) {
11        this.keys = new ArrayList<E>(n);
12        this.childs = new ArrayList<BNode<E>>(n + 1);
13        this.count = 0;
14        for (int i = 0; i < n; i++) {
15            this.keys.add(e:null);
16        }
17        for (int i = 0; i < n + 1; i++) {
18            this.childs.add(e:null);
19        }
20    }
21
22    // Verifica si el nodo está lleno
23    public boolean nodeFull(int n) {
24        return count == n;
25    }
26
27    // Verifica si el nodo está vacío
28    public boolean nodeEmpty() {
29        return count == 0;
30    }
31
32    // Busca una clave dentro del nodo y retorna si existe, junto con su posición
33    public boolean searchNode(E key, int[] pos) {
34        int i = 0;
35        while (i < count && keys.get(i).compareTo(key) < 0) {
36            i++;
37        }
38
39        pos[0] = i;
40        if (i < count && keys.get(i).compareTo(key) == 0) {
41            return true; // encontrada
42        }
43    }
44 }
```

ii. Ejercicio 2: Implementación Clase BTree Genérico

```
18     public void insert(E cl) {
19         up = false;
20         E mediana;
21         BNode<E> pnew;
22         mediana = push(this.root, cl);
23         if (up) {
24             pnew = new BNode<E>(this.orden);
25             pnew.count = 1;
26             pnew.keys.set(index:0, mediana);
27             pnew.chlds.set(index:0, this.root);
28             pnew.chlds.set(index:1, nDes);
29             this.root = pnew;
30         }
31     }
32
33     private E push(BNode<E> current, E cl) {
34         int pos[] = new int[1];
35         E mediana;
36         if (current == null) {
37             up = true;
38             nDes = null;
39             return cl;
40         } else {
41             boolean fl = current.searchNode(cl, pos);
42             if (fl) {
43                 System.out.println(x:"Item duplicado\n");
44                 up = false;
45                 return null;
46             }
47             mediana = push(current.chlds.get(pos[0]), cl);
48             if (up) {
49                 if (current.nodeFull(this.orden - 1)) {
50                     mediana = dividedNode(current, mediana, pos[0]);
```

```
51
52     private void putNode(BNode<E> current, E cl, BNode<E> rd, int k) {
53         int i;
54         for (i = current.count - 1; i >= k; i--) {
55             current.keys.set(i + 1, current.keys.get(i));
56             current.chlds.set(i + 2, current.chlds.get(i + 1));
57         }
58         current.keys.set(k, cl);
59         current.chlds.set(k + 1, rd);
60         current.count++;
61     }
62
63     private E dividedNode(BNode<E> current, E cl, int k) {
64         BNode<E> rd = nDes;
65         int i, posMdna;
66         posMdna = (k <= this.orden / 2) ? this.orden / 2 : this.orden / 2 + 1;
67         nDes = new BNode<E>(this.orden);
68         for (i = posMdna; i < this.orden - 1; i++) {
69             nDes.keys.set(i - posMdna, current.keys.get(i));
70             nDes.chlds.set(i - posMdna + 1, current.chlds.get(i + 1));
71         }
72         nDes.count = (this.orden - 1) - posMdna;
73         current.count = posMdna;
74         if (k <= this.orden / 2)
75             putNode(current, cl, rd, k);
76         else
77             putNode(nDes, cl, rd, k - posMdna);
78         E median = current.keys.get(current.count - 1);
79         nDes.chlds.set(index:0, current.chlds.get(current.count));
80         current.count--;
81         return median;
82     }
83 }
```

iii. Ejercicio 3 : Implementación método toString Genérico

```
105     @Override
106     public String toString() {
107         String s = "";
108         if (isEmpty()) {
109             s += "BTree is empty...";
110         } else {
111             s = writeTree(this.root, level:0);
112         }
113         return s;
114     }
115
116     private String writeTree(BNode<E> current, int level) {
117         StringBuilder sb = new StringBuilder();
118
119         if (current != null) {
120             for (int i = current.count - 1; i >= 0; i--) {
121                 sb.append(writeTree(current.childs.get(i + 1), level + 1));
122                 sb.append(" ".repeat(level * 4)); // sangría según nivel
123                 sb.append(current.keys.get(i)).append(str:"\n");
124             }
125             sb.append(writeTree(current.childs.get(index:0), level + 1));
126         }
127
128         return sb.toString();
129     }
130
131 }
132
```

b. Ejercicios Propuestos:

- i. **EJERCICIO 1:** Mediante la dinámica de un árbol B, sea de grado: 5 realizar lo siguiente:
 - Inserción de los siguientes nodos: 100 – 200 – 300 – 400 – 500 - 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190
 - Mostrar el árbol resultante de eliminar los nodos: 100 – 200 – 300 – 400 – 500 – 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190
 - Mostrar el paso a paso de la inserción y eliminación de cada nodo
 - Mostrar el árbol resultante

INSERCIÓN

• 1°

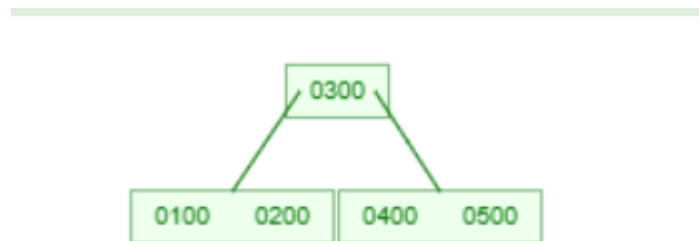
B-Trees

☐ Max. Degree = 3
☐ Max. Degree = 4
☒ Max. Degree = 5
☐ Max. Degree = 6
☐ Max. Degree = 7

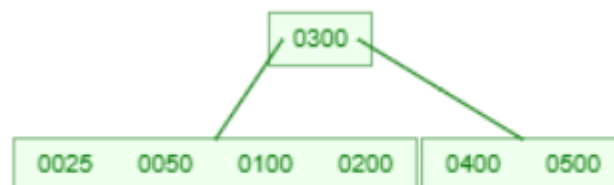
☐ Preemptive Split / Merge (Even max degree only)

0100
0200
0300
0400

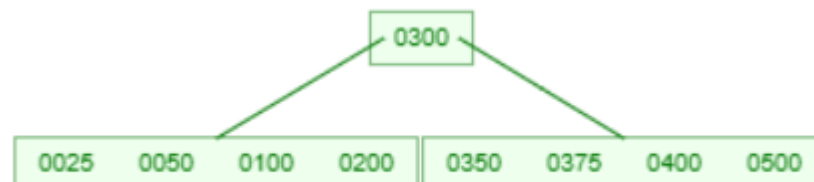
• 2°



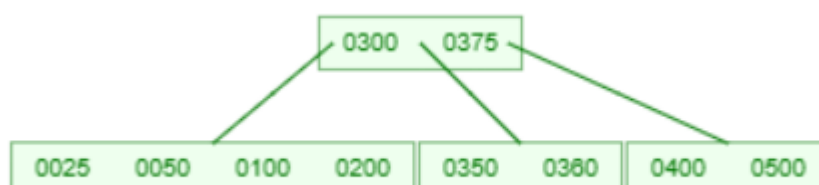
• 3°



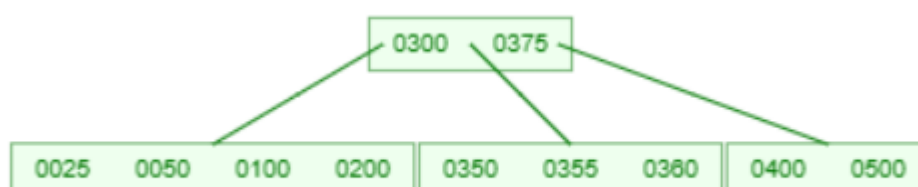
• 4°



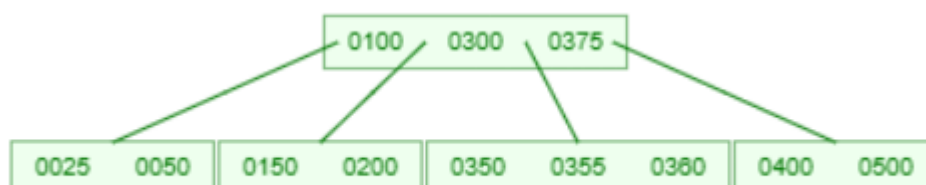
• 5°



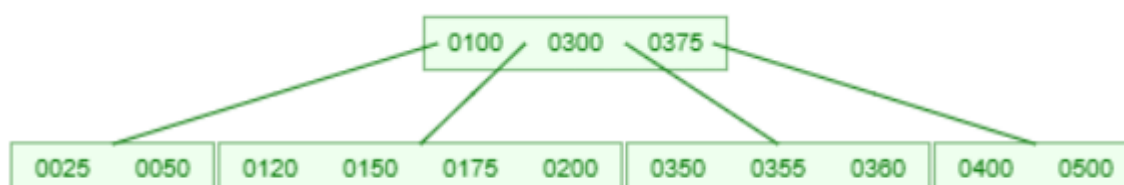
• 6°



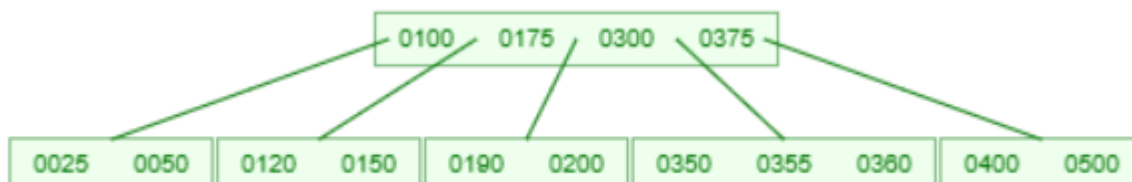
• 7°



• 8°

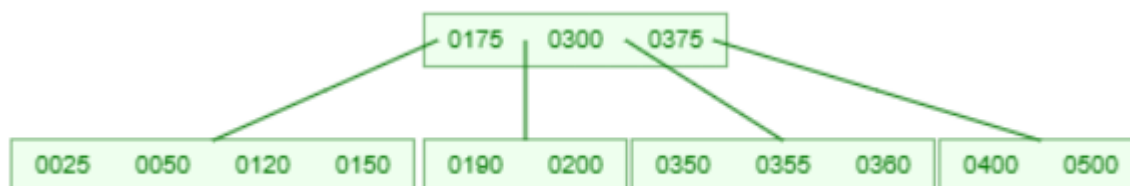


- 9° (Árbol resultante)

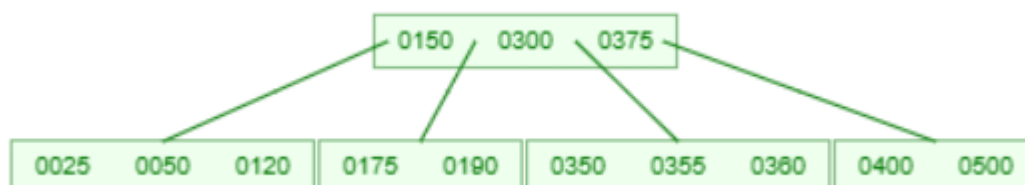


ELIMINACIÓN

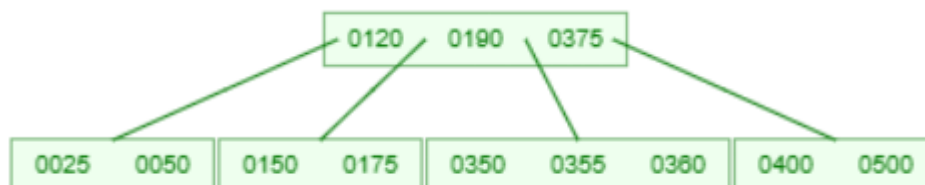
- 1°



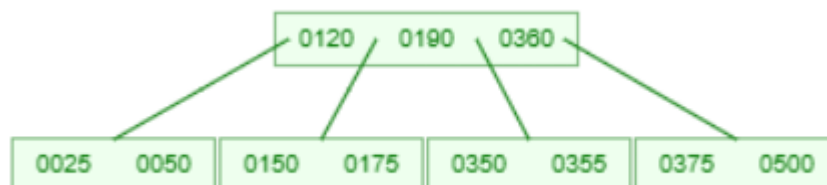
- 2°



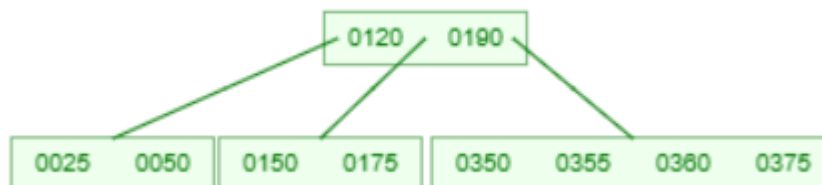
- 3°



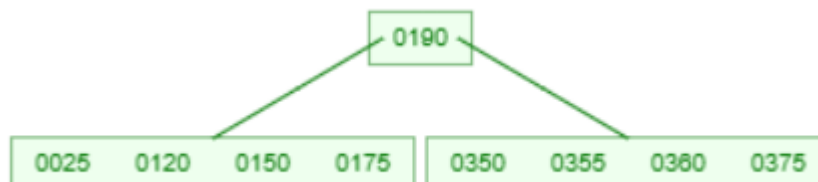
• 4°



• 5°



• 6°



• 7°



• 8°

0120 0150 0175 0190

ii. **EJERCICIO 2:** Ejercicio 2: Mediante la dinámica de un árbol B, sea de grado: 4 realizar lo siguiente:

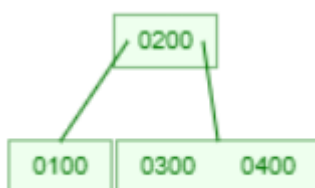
- Inserción de los siguientes nodos: 100 – 200 – 300 – 400 – 500 - 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el árbol resultante de eliminar los nodos: 100 – 200 – 300 – 400 – 500 – 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el paso a paso de la inserción y eliminación de cada nodo.
- Mostrar el árbol resultante.

INSERCIÓN

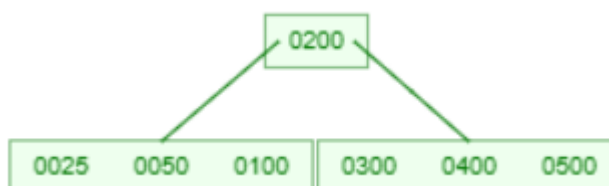
• 1°

0100 0200 0300

• 2°



• 3°



• 4°



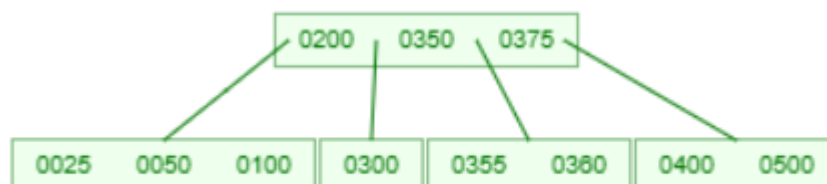
• 5°



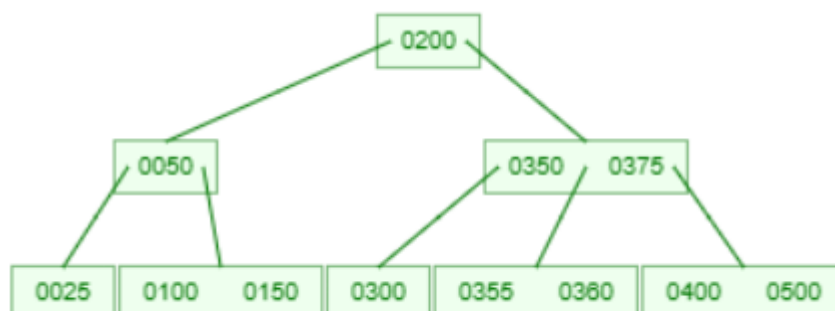
• 6°



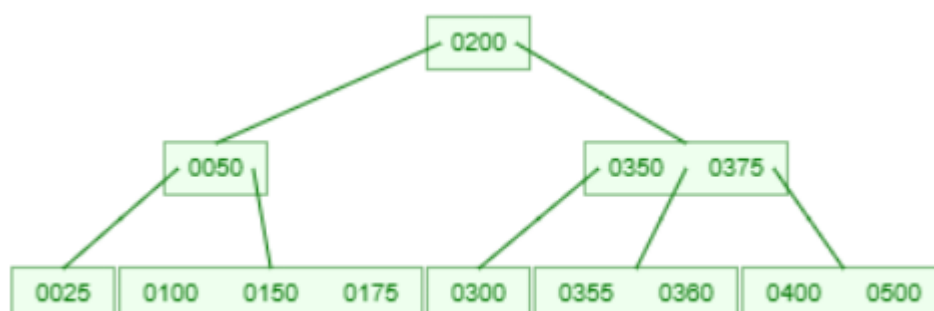
• 7°



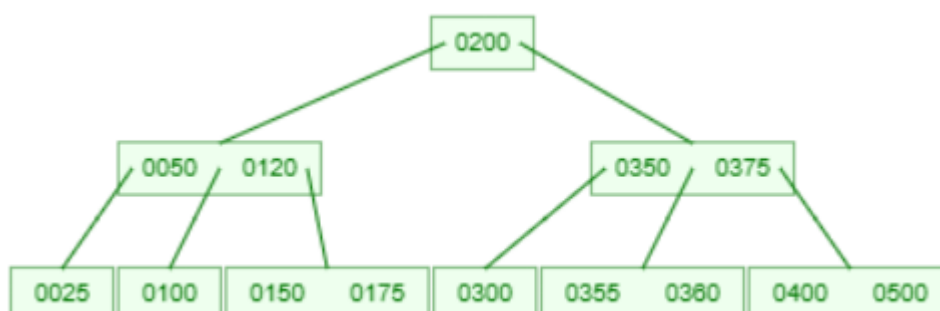
• 8°



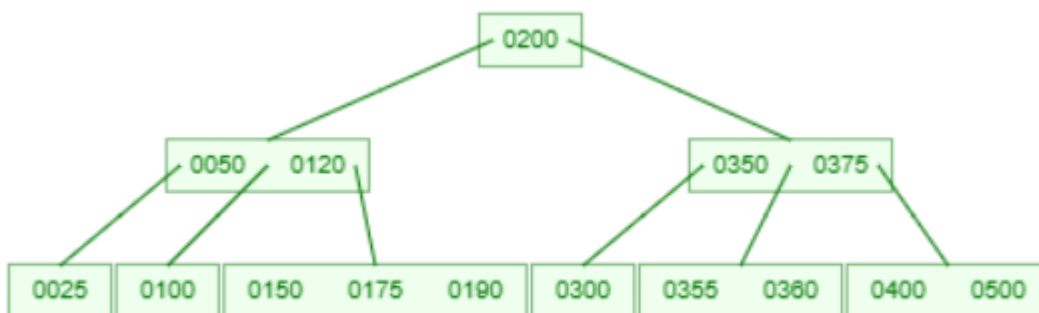
• 9°



• 10°

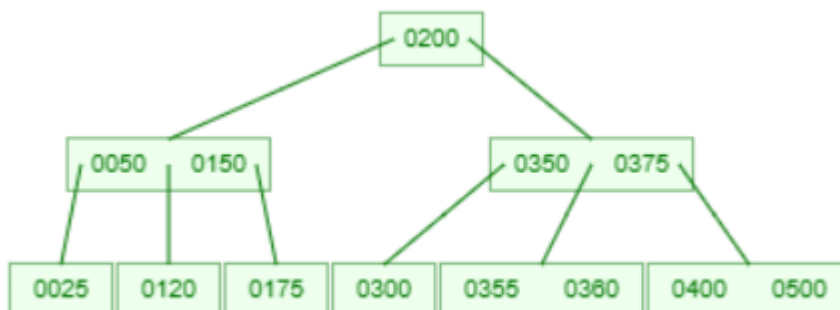


- 11° (Árbol resultante)

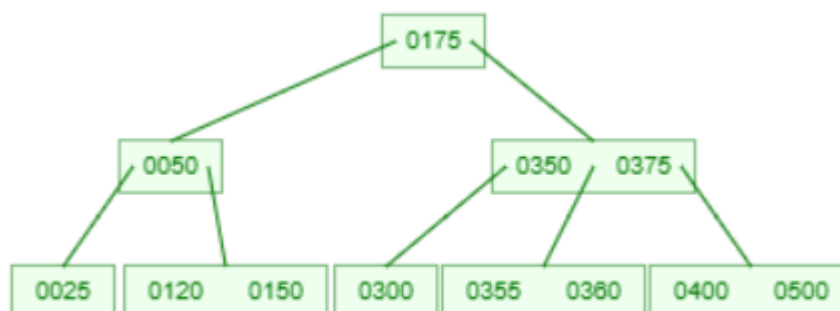


ELIMINACIÓN

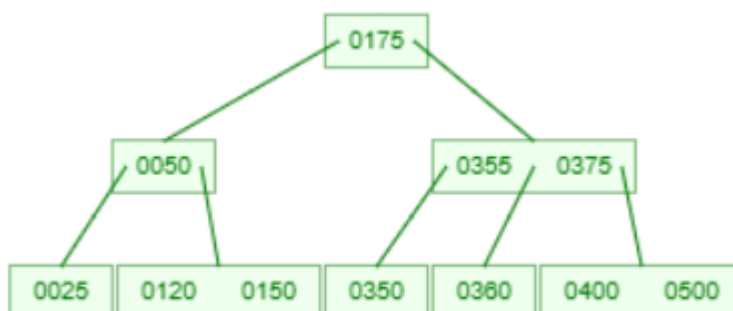
- 1°



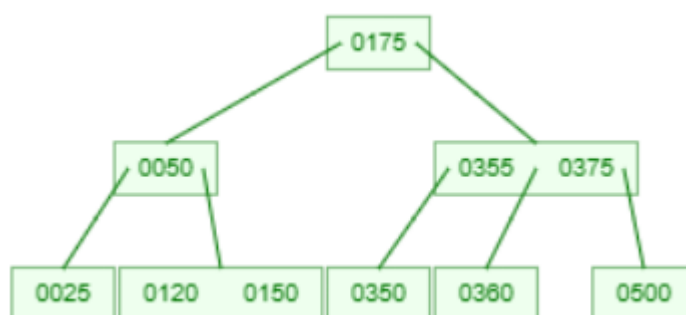
- 2°



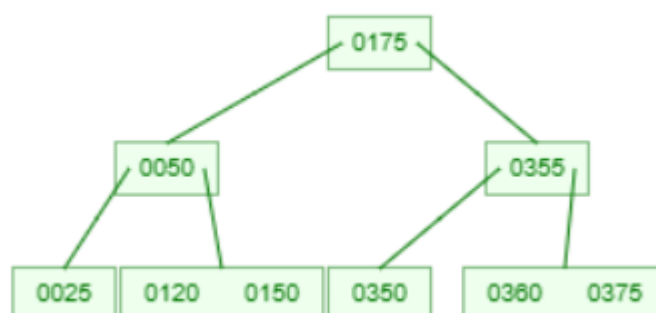
• 3°



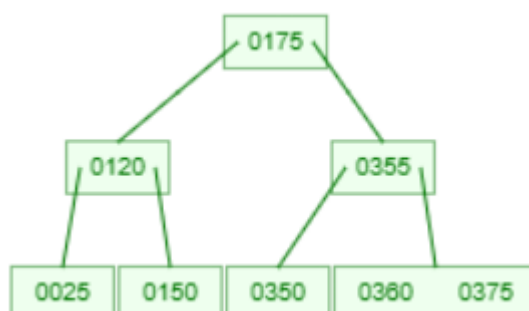
• 4°



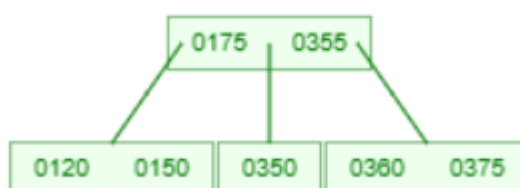
• 5°



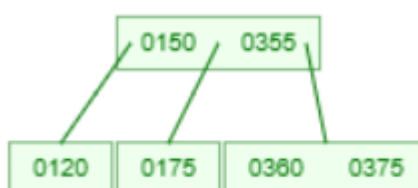
• 6°



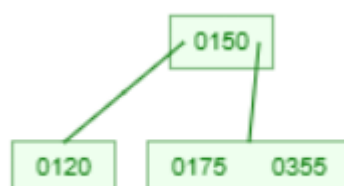
• 7°



• 8°



• 9°



	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 15</p>

• 10°

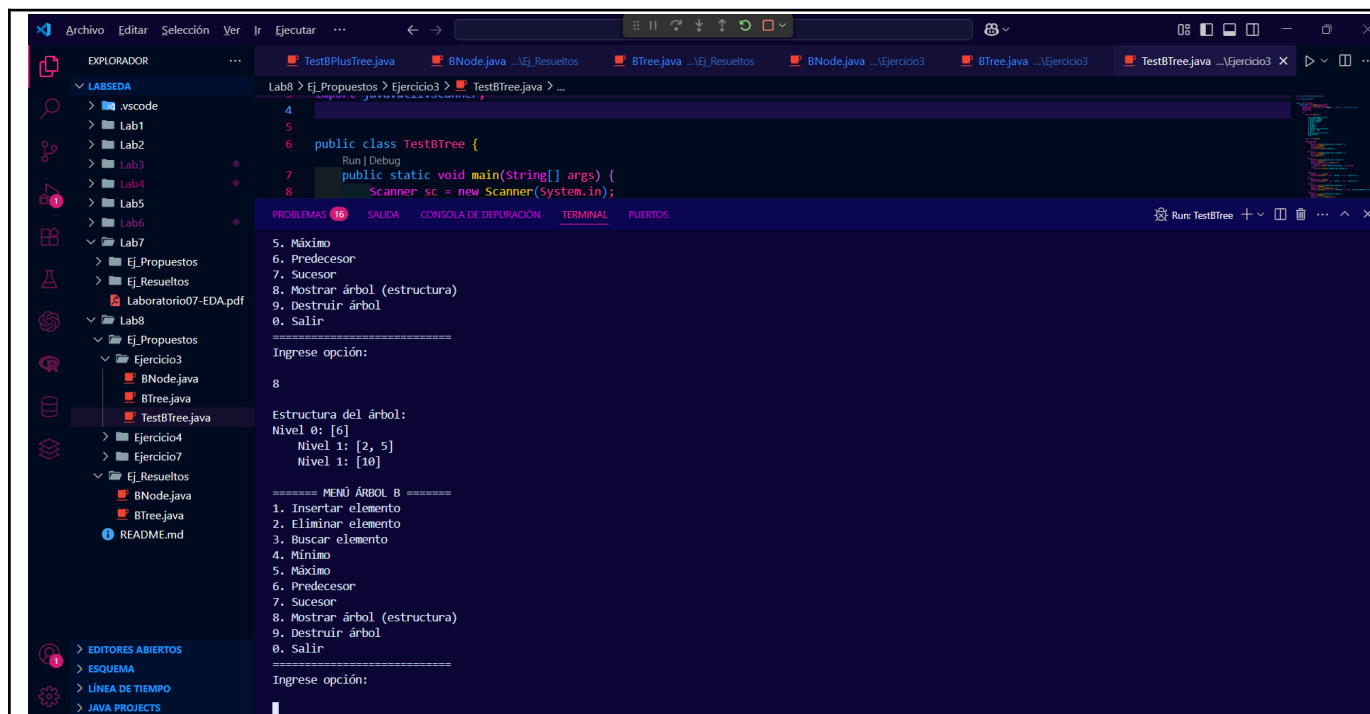
0120 0175 0190

iii. EJERCICIO 3: Implementar un árbol B, con clases y métodos genéricos:

- Implementar todas las operaciones: `destroy()`, `isEmpty()`, `insert(x)`, `remove(x)`, `search(x)`, `Min()`, `Max()`, `Predecesor()`, `Sucesor()`, `toString()`, `writeTree()`, `FuzeNode()`, `dividedNode()` y los necesarios.
- Implementar una clase `Test` para probar los métodos y mostrar los resultados. utilizando clases y métodos genéricos, utilizando un menú de opciones para todas las operaciones del árbol B.
- Para comprobar utilice los ejercicios 1 y 2 para mostrar el árbol resultante luego de la inserción y la eliminación.

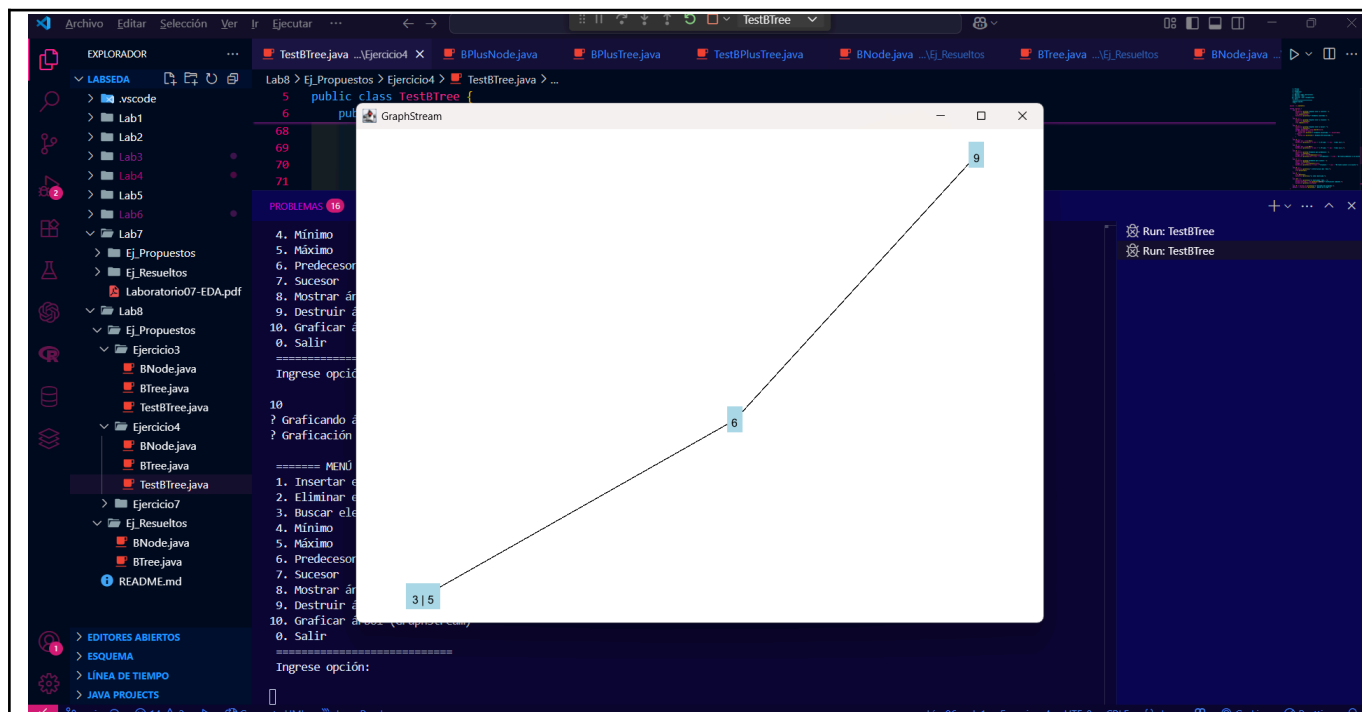
En el Ejercicio 3, se desarrolló un árbol B genérico a través de la clase `BTree<T>`, acompañado por la clase de nodos `BNode<T>`. El objetivo fue implementar una estructura de árbol balanceado en la que cada nodo puede contener múltiples claves y referencias a hijos, permitiendo un acceso eficiente y ordenado a los elementos. Las operaciones clave incluidas fueron: `insert()` para añadir elementos de manera ordenada, dividiendo nodos cuando se llenan; `remove()` para eliminar claves manteniendo la estructura del árbol mediante redistribuciones o fusiones; y `search()` para localizar una clave específica. También se incluyeron métodos como `destroy()` para reiniciar el árbol, `isEmpty()` para verificar si está vacío, y métodos de recorrido como `Min()`, `Max()`, `Predecesor()` y `Sucesor()` que permiten obtener los extremos o claves vecinas. Además, se implementaron `FuzeNode()` y `dividedNode()` para controlar la fusión y división de nodos. Todo el árbol fue desarrollado con clases y métodos genéricos, asegurando flexibilidad para trabajar con cualquier tipo de dato comparable.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 16</p>



iv. EJERCICIO 4: Implementar un método para graficar el árbol B resultante, mostrando todos sus nodos, claves, sus aristas izquierda y derecha utilizando clases y métodos genéricos, utilizar la librería Graph Stream o similar.

En el Ejercicio 4, se añadió un método de graficación del árbol B utilizando la librería GraphStream, lo cual permitió visualizar gráficamente la estructura del árbol. Se desarrolló un método graficar() que recorre los nodos del árbol y crea un grafo jerárquico, asignando a cada nodo una posición coordenada (x, y) para representar los niveles y ramas del árbol. Las claves de cada nodo se muestran agrupadas en cajas, y las conexiones entre nodos se dibujan sin flechas, simulando un diagrama de árbol. Esta visualización fue clave para comprobar visualmente el comportamiento del árbol tras las operaciones de inserción y eliminación.



v. EJERCICIO 5: : Mediante la dinámica de un árbol B+, sea de grado: 5 realizar lo siguiente:

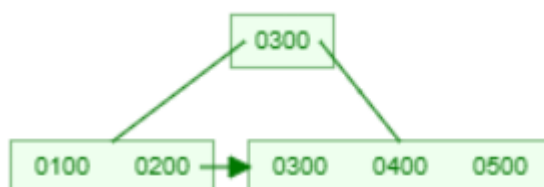
- Inserción de los siguientes nodos: 100 – 200 – 300 – 400 – 500 - 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el árbol resultante de eliminar los nodos: 100 – 200 – 300 – 400 – 500 – 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el paso a paso de la inserción y eliminación de cada nodo.

INSERCIÓN

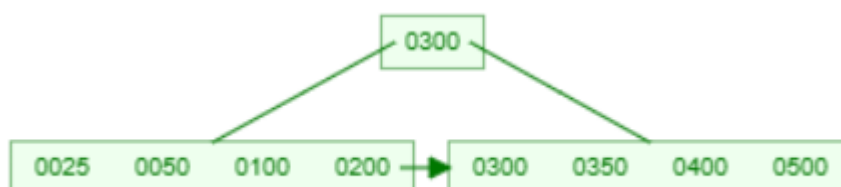
- 1º

0100	0200	0300	0400
------	------	------	------

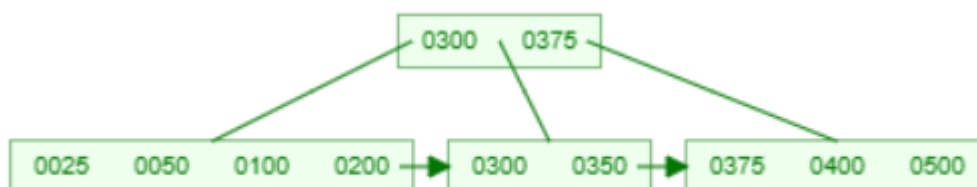
- 2°



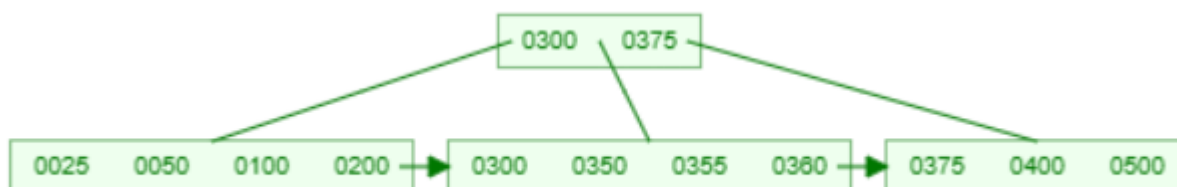
- 3°



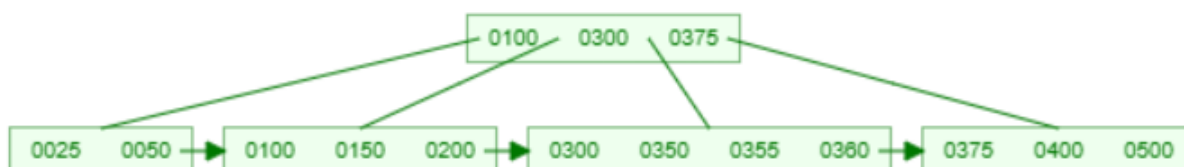
- 4°



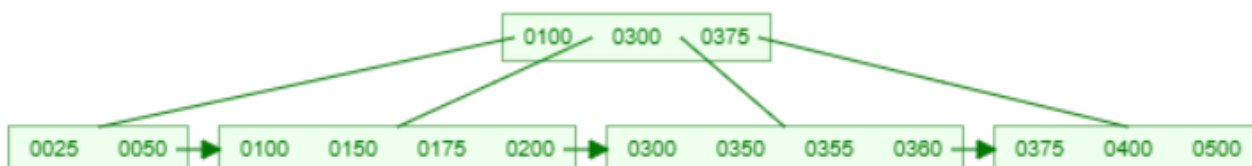
- 5°



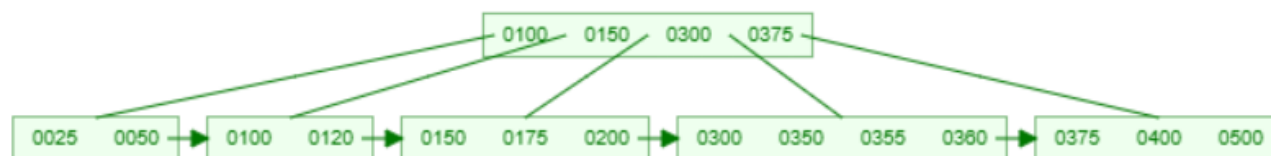
• 6°



• 7°



• 8°



• 9° (Árbol resultante)

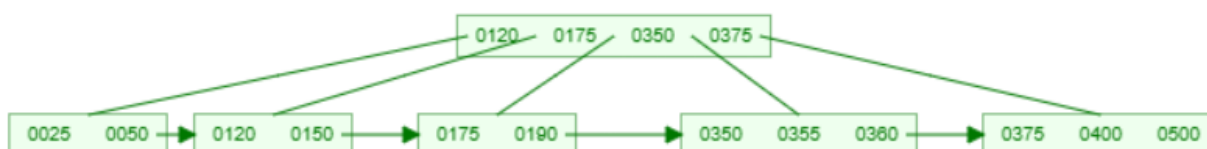


ELIMINACIÓN

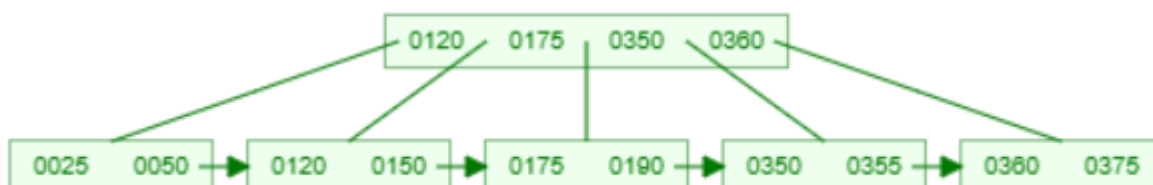
- 1°



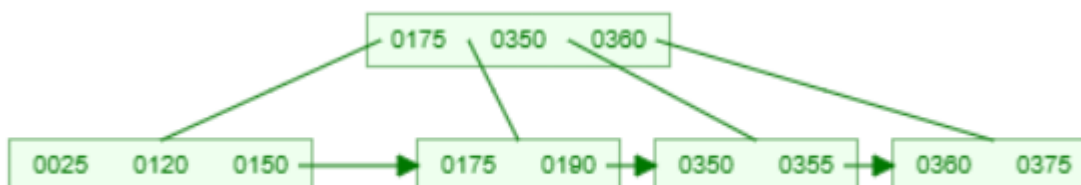
- 2°



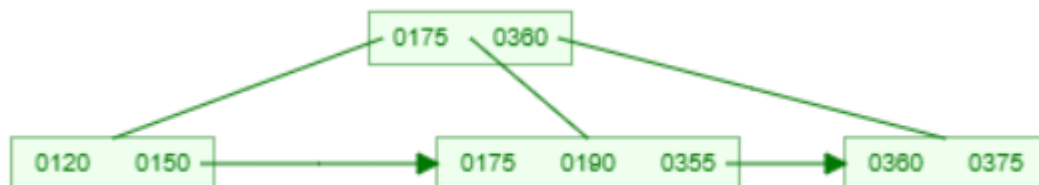
- 3°



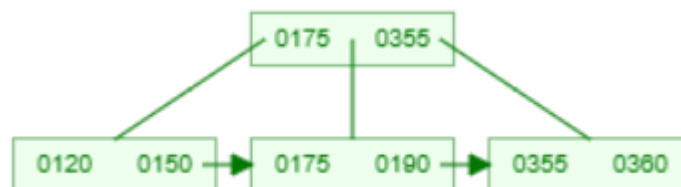
- 4°



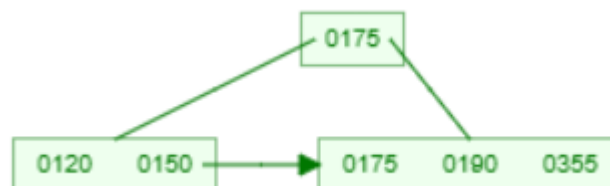
• 5°



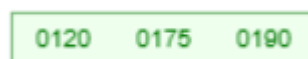
• 6°



• 7°



• 8°

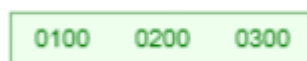


vi. **EJERCICIO 6:** Mediante la dinámica de un árbol B+, sea de grado: 4 realizar lo siguiente:

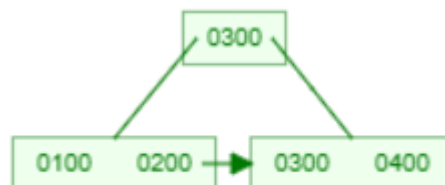
- Inserción de los siguientes nodos: 100 – 200 – 300 – 400 – 500 - 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el árbol resultante de eliminar los nodos: 100 – 200 – 300 – 400 – 500 – 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el paso a paso de la inserción y eliminación de cada nodo. o Mostrar el árbol resultante.

INSERCIÓN

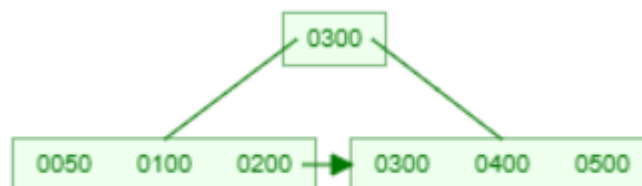
- 1°



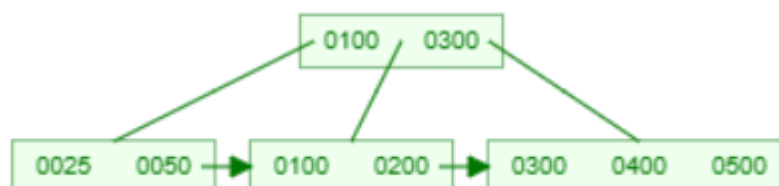
- 2°



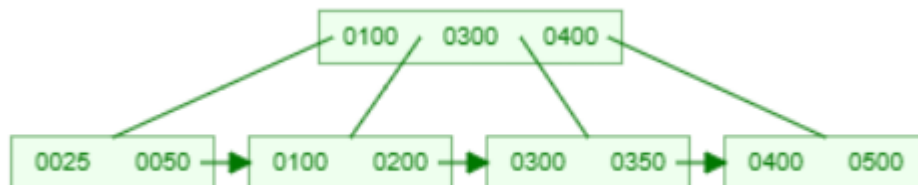
- 3°



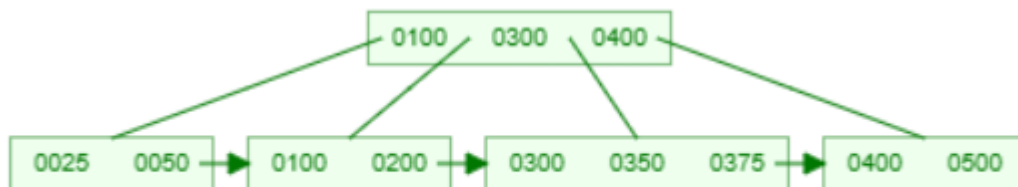
- 4°



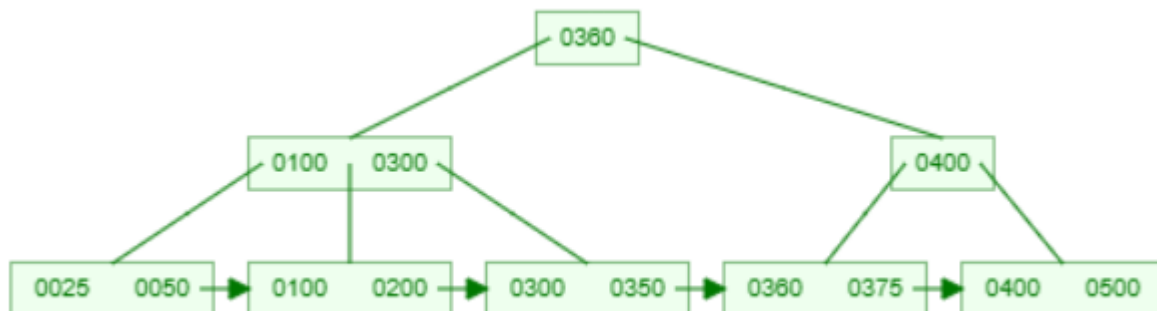
• 5°



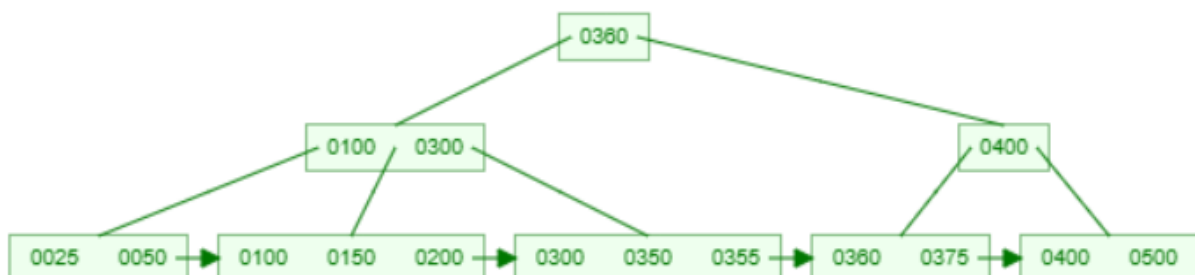
• 6°



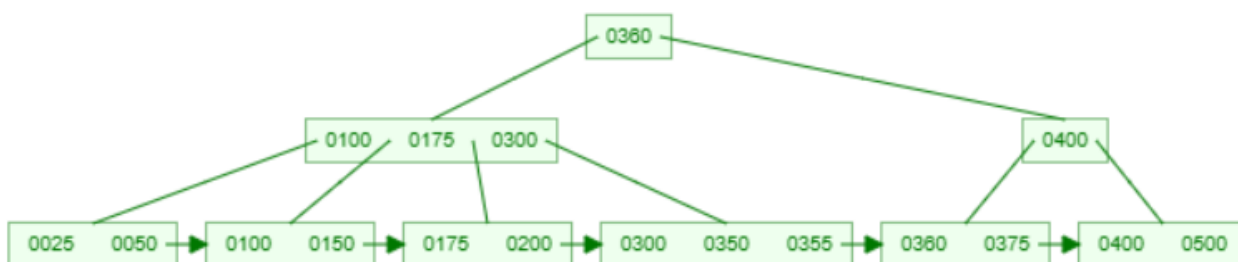
• 7°



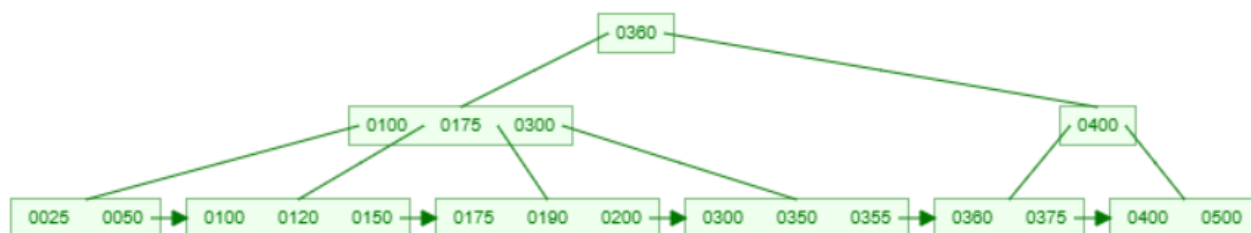
• 8°



- 9°

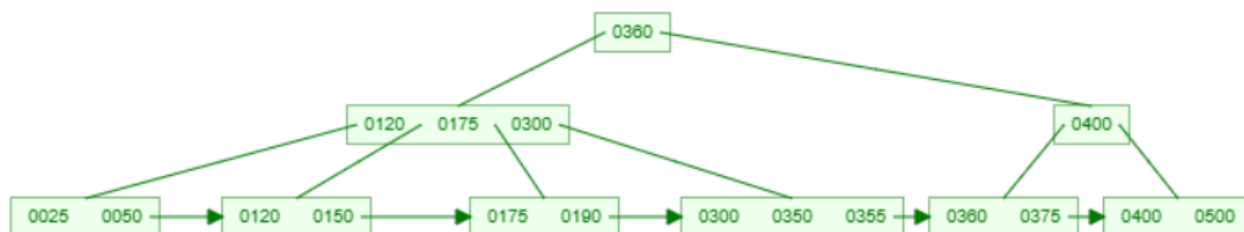


- 10° (Árbol resultante)

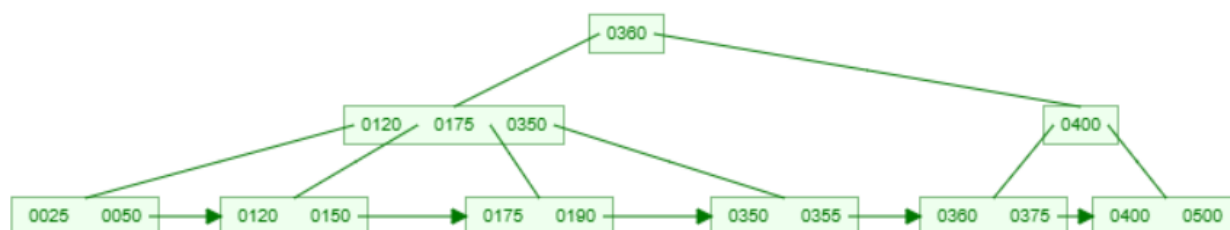


ELIMINACIÓN

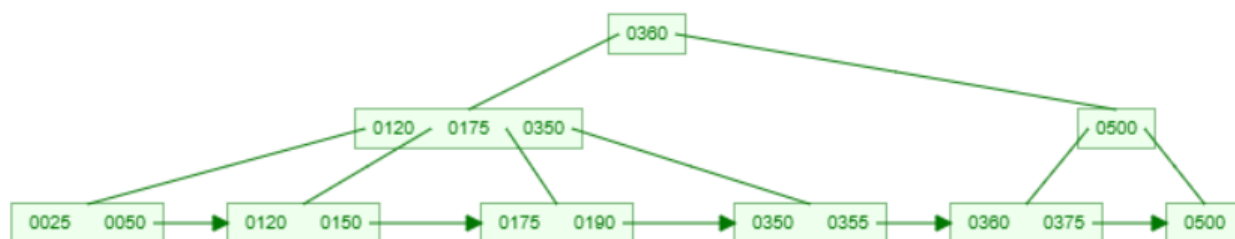
- 1°



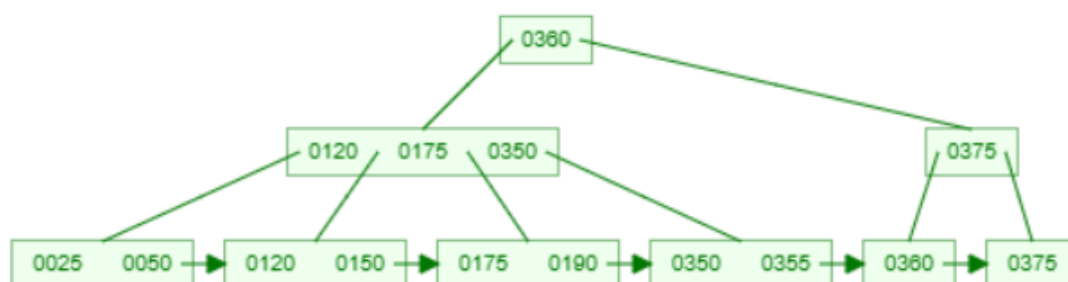
• 2°



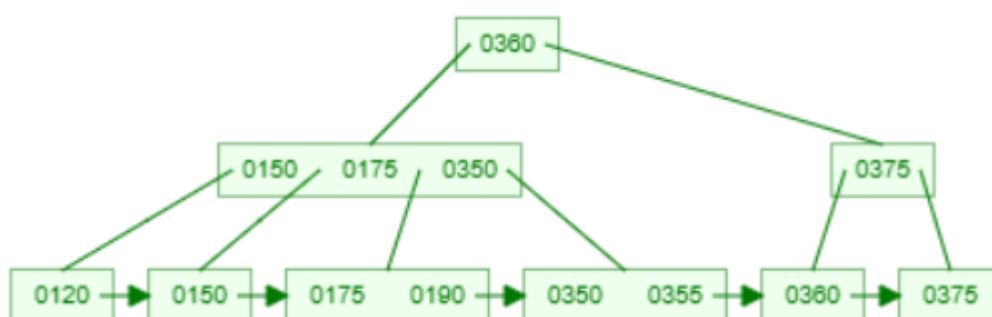
• 3°



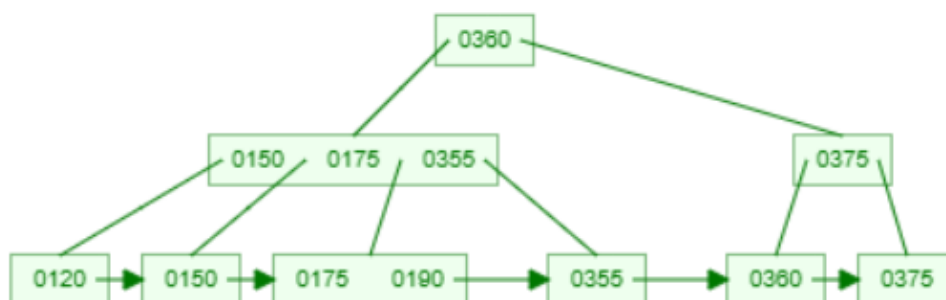
• 4°



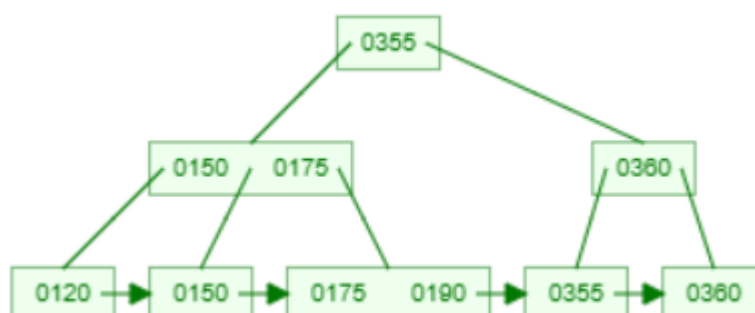
• 5°



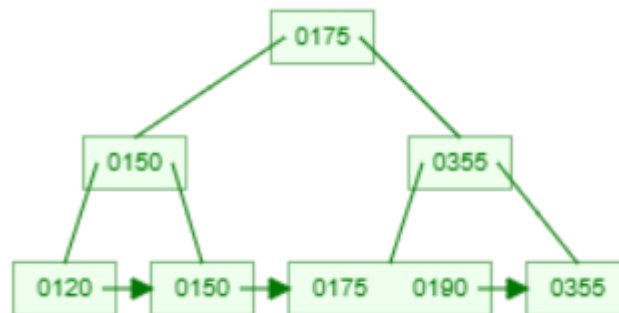
• 6°



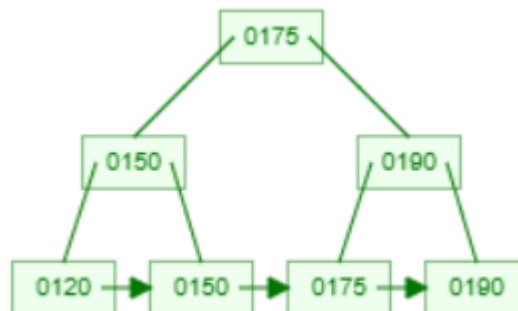
• 7°



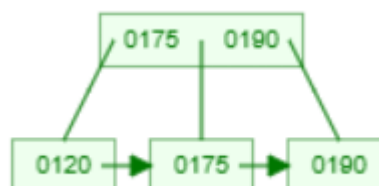
• 8°



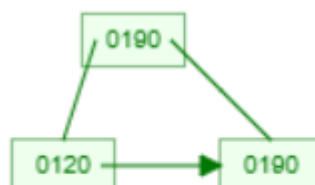
• 9°



• 10°



• 11°



	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 28</p>

• 12°

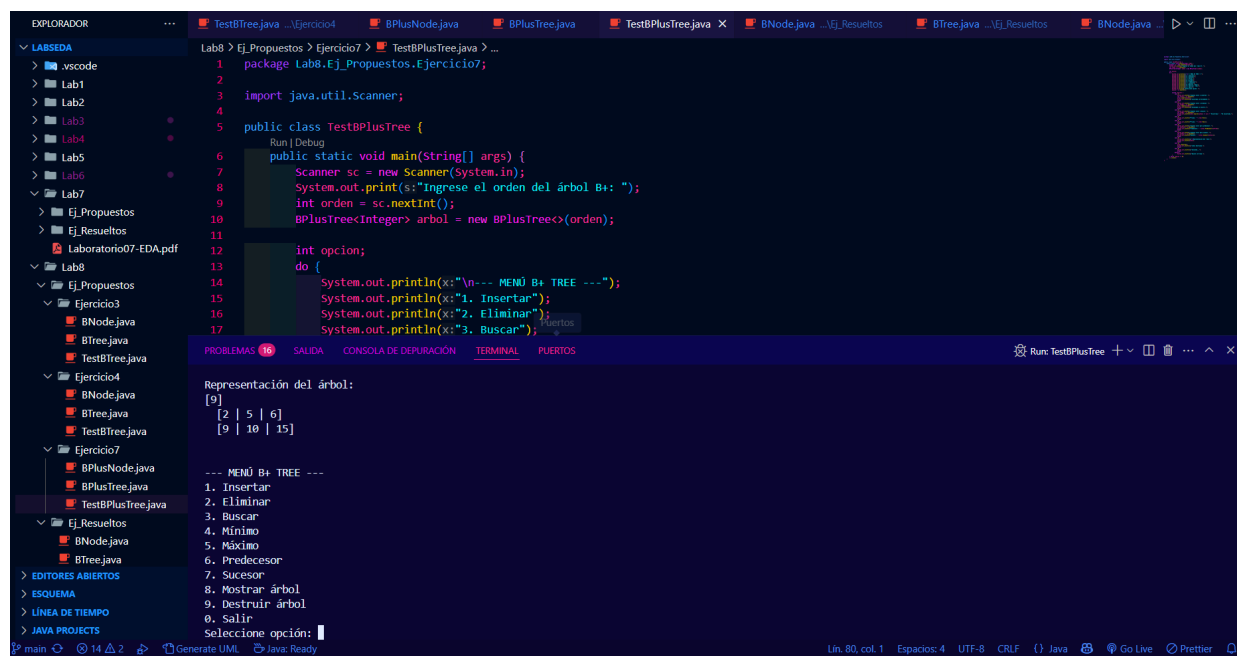
0190

vii. EJERCICIO 7: Implementar un árbol B+, con clases y métodos genéricos:

- Implementar todas las operaciones: `destroy()`, `isEmpty()`, `insert(x)`, `remove(x)`, `search(x)`, `Min()`, `Max()`, `Predecesor()`, `Sucesor()`, `toString()`, `writeTree()`, `FuzeNode()`, `dividedNode()` y los necesarios.
- Implementar una clase `Test` para probar los métodos y mostrar los resultados. utilizando clases y métodos genéricos, utilizando un menú de opciones para todas las operaciones del árbol B+.
- Para comprobar utilice los ejercicios 5 y 6 para mostrar el árbol resultante luego de la inserción y la eliminación

En el Ejercicio 7, se implementó un árbol B+, una variante especializada del árbol B utilizada ampliamente en bases de datos y sistemas de archivos. En el B+, todas las claves se almacenan en nodos hoja y no en los nodos internos, lo que mejora la eficiencia en recorridos secuenciales. Se desarrolló la clase `BPlusTree<T>` junto con `BPlusNode<T>`, ambas usando generics para mantener la flexibilidad. Se implementaron todas las operaciones solicitadas: `insert()` para añadir claves y dividir hojas si es necesario; `remove()` con manejo de redistribución o fusión en caso de subdesbordamiento; `search()` para localizar claves; `destroy()` e `isEmpty()` para manejar el estado del árbol; y operaciones adicionales como `Min()`, `Max()`, `Predecesor()` y `Sucesor()`, que aprovechan la estructura de hojas enlazadas del árbol. También se incluyeron `FuzeNode()` y `dividedNode()` para manejar la unión o partición de nodos. El método `toString()` y `writeTree()` permiten imprimir el árbol en consola. Finalmente, se creó una clase `TestBPlusTree` con un menú de opciones para ejecutar y validar cada una de las operaciones de forma interactiva, comprobando la funcionalidad con inserciones y eliminaciones como en los ejercicios anteriores.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 29</p>



```

package Lab8.Ej_Propuestos.Ejercicio7;

import java.util.Scanner;

public class TestBPlusTree {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Ingrese el orden del árbol B+: ");
        int orden = sc.nextInt();
        BPlusTree<Integer> arbol = new BPlusTree<>(orden);

        int opcion;
        do {
            System.out.println("\n--- MENÚ B+ TREE ---");
            System.out.println("1. Insertar");
            System.out.println("2. Eliminar");
            System.out.println("3. Buscar");
        } while (opcion != 0);
    }
}

```

Representación del árbol:

```

[9]
[2 | 5 | 6]
[9 | 10 | 15]

```

--- MENÚ B+ TREE ---

1. Insertar
2. Eliminar
3. Buscar
4. Mínimo
5. Máximo
6. Predecesor
7. Sucesor
8. Mostrar árbol
9. Destruir árbol
0. Salir

Seleccione opción:

II. SOLUCIÓN DEL CUESTIONARIO

a. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc

- **Escasa documentación sobre árboles B/B+ genéricos en Java:** Muchos ejemplos en línea están hechos con clases específicas o implementaciones simplificadas. Fue difícil encontrar referencias completas que incluyeran todas las operaciones (inserción, eliminación, fusión, búsqueda, etc.) usando clases y métodos genéricos.
- **Complejidad en la implementación de eliminación (remove):** La operación `remove()` en árboles B y B+ no es trivial. Requiere manejar casos como redistribución, fusión de nodos, y reestructuración del árbol, lo cual puede volverse complejo especialmente al conservar balance y orden.
- **Gestión de punteros y estructura en nodos hoja (B+):** En el árbol B+, es necesario mantener enlaces `next` entre los nodos hoja. Coordinar estos enlaces correctamente durante la inserción y eliminación de claves fue un reto adicional.
- **Visualización con GraphStream sin guía directa:** La librería GraphStream es poderosa, pero no está diseñada específicamente para

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 30</p>

árboles. Se tuvo que diseñar una estrategia personalizada para ubicar cada nodo en el plano sin que se superpongan, y mantener el orden jerárquico del árbol.

- **Asignación correcta de coordenadas (x, y):** Para graficar adecuadamente el árbol, fue necesario implementar un control de posición horizontal (posX) y nivel vertical (nivel) para cada nodo, algo que no está disponible automáticamente y debe calcularse manualmente.
- **Adaptación de los métodos toString() y writeTree():** Estos métodos requieren recorrer el árbol y representar correctamente los nodos internos y hojas. En el árbol B+ se tuvo que ajustar la impresión para que solo las hojas contengan claves reales.
- **Complejidad del lenguaje Java al manejar referencias entre objetos:** Java no permite paso por referencia directo, lo cual dificultó algunas operaciones internas como dividir nodos y actualizar raíces o hijos, especialmente en métodos recursivos.

b. ¿Explique cómo es el algoritmo que implemento para obtener el B con la librería Graph Stream? Recuerda que puede agregar operaciones sobre la clase BST.

Para graficar el Árbol B con GraphStream, se desarrolló un algoritmo recursivo dentro de la clase BTree que recorre el árbol en orden jerárquico, desde la raíz hasta las hojas. El algoritmo asigna a cada nodo un ID único basado en su posición y un par de coordenadas (x, y) que se utilizan para ubicar el nodo en el plano. La coordenada y depende del nivel del nodo en el árbol, mientras que x es un contador global que se incrementa a medida que se procesan nodos hoja, lo cual asegura una distribución horizontal adecuada. Cada nodo se representa como una caja que contiene todas sus claves, unidas por un separador (|). Además, se generan aristas entre cada nodo y sus hijos siguiendo el índice de los punteros hijos. Este diseño garantiza que la forma jerárquica del árbol B se conserve gráficamente, representando con claridad la estructura interna del árbol, incluyendo claves múltiples por nodo y sus respectivas ramificaciones.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 31</p>

c. ¿Explique cómo es el algoritmo que implemento para obtener el B+ con la librería Graph Stream? Recuerda que puede agregar operaciones sobre la clase BST.

El algoritmo para graficar el Árbol B+ es similar al del Árbol B, pero tiene en cuenta las características particulares del B+, como que todas las claves están en nodos hoja y estos están enlazados secuencialmente mediante punteros next. En el recorrido recursivo, se sigue la misma lógica de asignar coordenadas (x, y) a cada nodo y representar sus claves agrupadas visualmente. Sin embargo, se añadió un paso opcional para visualizar las conexiones entre nodos hoja consecutivos mediante aristas adicionales (por ejemplo, con color o estilo diferente), lo cual permite visualizar cómo el árbol permite recorridos eficientes en orden. Esta visualización refuerza la diferencia entre los árboles B y B+, mostrando no solo su estructura jerárquica, sino también su capacidad para búsquedas y recorridos ordenados de manera eficiente. El uso de GraphStream en este contexto resultó útil, pero fue necesario un control más preciso de las coordenadas y el nombre de los nodos para evitar solapamientos o errores de referencia durante la creación del grafo.

III. CONCLUSIONES

- El uso de estructuras genéricas en Java mejora la reutilización del código, permitiendo construir árboles B y B+ que funcionen con distintos tipos de datos sin necesidad de redefinir las clases para cada tipo.
- Implementar todas las operaciones fundamentales (insertar, eliminar, buscar, mínimo, máximo, predecesor, sucesor) requiere un entendimiento profundo de la estructura interna de los árboles, especialmente la redistribución y fusión de nodos tras eliminaciones.
- La visualización con GraphStream aporta un valor adicional al aprendizaje y la depuración, permitiendo observar la evolución estructural del árbol a medida que se realizan operaciones, aunque exige manejar cuidadosamente las posiciones de los nodos para evitar superposiciones.
- El árbol B es ideal para operaciones jerárquicas y búsquedas eficientes, mientras que el árbol B+ es más adecuado para recorridos secuenciales y sistemas de bases de datos, gracias a sus hojas enlazadas y mejor manejo de rangos.
- Diseñar un menú de prueba completo (clase Test) es clave para validar la implementación, ya que permite verificar todas las funcionalidades de forma sistemática y facilita detectar errores lógicos o estructurales.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 32</p>

- La experiencia adquirida al implementar estas estructuras desde cero fortalece habilidades clave en estructuras de datos avanzadas, programación orientada a objetos, recursividad y uso de librerías gráficas externas.

REFERENCIAS Y BIBLIOGRAFÍA

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.
- Weiss, M. A. (2012). *Data structures and algorithm analysis in Java* (3rd ed.). Pearson Education.
- Knuth, D. E. (1998). *The art of computer programming, Volume 3: Sorting and searching* (2nd ed.). Addison-Wesley.
- Baase, S., & Van Gelder, A. (2000). *Computer algorithms: Introduction to design and analysis* (3rd ed.). Addison-Wesley.
- GraphStream Project. (n.d.). *GraphStream - A dynamic graph library*. Recuperado de <http://graphstream-project.org>
- GeeksforGeeks. (n.d.). *B-Tree Set 1 (Introduction)*. Recuperado de <https://www.geeksforgeeks.org/b-tree-set-1-introduction-2/>
- Oracle. (n.d.). *Java Platform, Standard Edition Documentation*. Recuperado de <https://docs.oracle.com/javase/>