UNIVERSIDAD TECNOLÓGICA DE PANAMÁCENTRO REGIONAL DE CHIRIQUÍ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

CARRERA:
Gestión y Desarrollo de Software

ACTIVIDAD No. 7

LABORATORIO No. 3

“Laboratorio 3”

ASIGNATURA: Estructura de Datos II

DOCENTE:
Profa. Nunehar Mondul

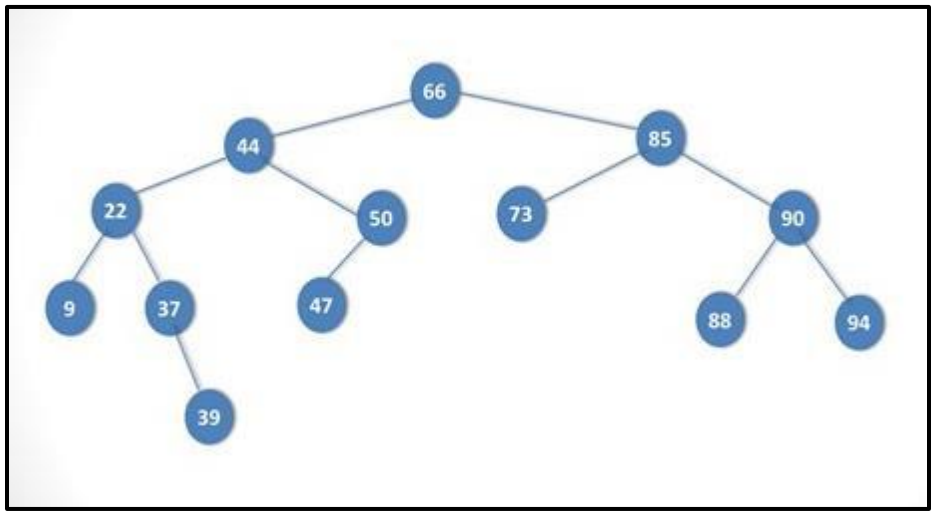
ESTUDIANTE/s:
Jorge Jiménez (4-826-874)
Jorge Santos (12-717-1795)
Manuel Jaen (4-803-949)

I SEMESTRE 2025

FECHA:
04/29/2025

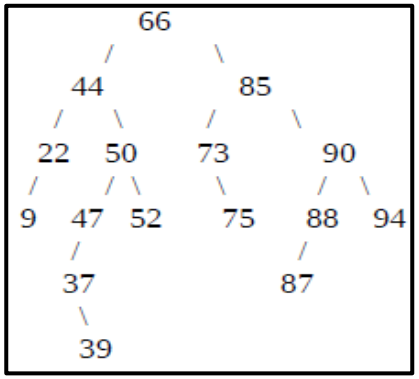
Desarrollo

1. "Para el siguiente árbol escribir recorrido pre-orden, in orden y post orden." (MJ)



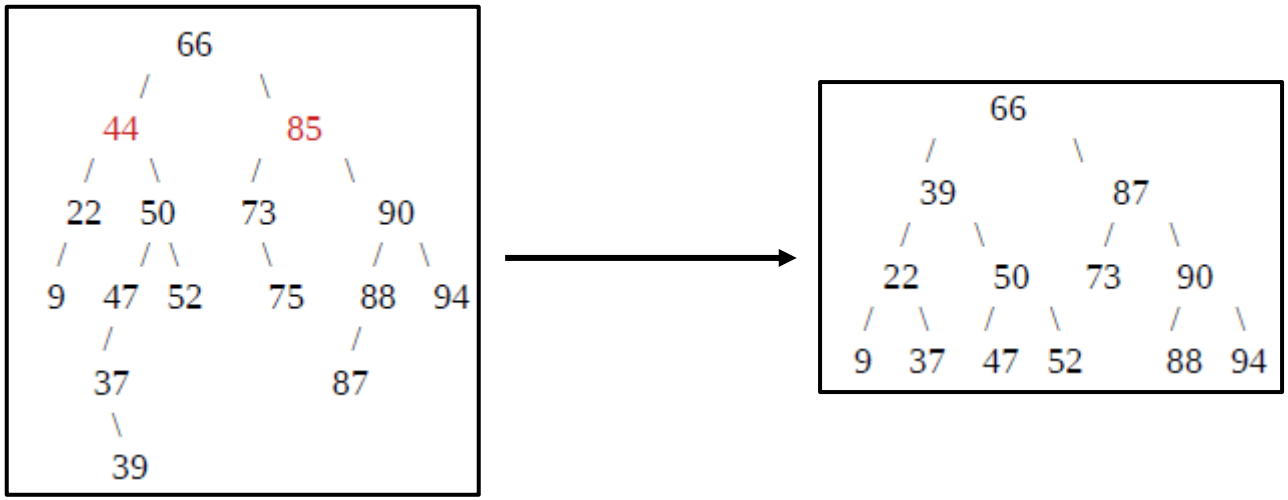
Pre-order(R→I→D). {66, 44, 22, 9, 37, 39, 50, 47, 85,73, 90, 88, 94}
In-order(I→R→D). {9, 22,37, 39, 44, 50, 47, 66, 73, 85, 88, 90, 94}
Post-order(I→D→R). {9, 39, 37, 22, 47, 50, 44, 73, 88, 94, 90, 85, 66}

2. "Añadir nodo 52, nodo 87 y nodo 75." (MJ)



Pre-order(R→I→D). {66,44,22,9,50,47,37,39,52,85,73,75,90,88,87,94}
In-order(I→R→D). {9,22,44,37,39,47,50,52,66,73,75,85,87,88,90,94}
Post-order(I→D→R). {9,22,37,39,47,52,50,44,73,75,87,88,94,90,85,66}

3. "Eliminar nodo 85 y 44." **(MJ)**



4. "Buscar que es un árbol binario balanceado. Porque es mejor balancearlos. Como se balancean." **(JS)**

Un árbol binario balanceado es un tipo de árbol binario en el que la altura de los subárboles izquierdo y derecho de cada nodo difiere en, como máximo, uno. Ejemplos de árboles balanceados incluyen los árboles AVL y los árboles rojo-negro.

¿Por qué es mejor balancearlos?

1. *Eficiencia en operaciones:* Las operaciones como búsqueda, inserción y eliminación tienen un rendimiento óptimo (O(log n)) en árboles balanceados, mientras que, en árboles desbalanceados, estas operaciones pueden degradarse a O(n) en el peor caso.
2. *Reducción del tiempo de ejecución:* Mantener una estructura balanceada mejora el tiempo promedio de ejecución de las operaciones.

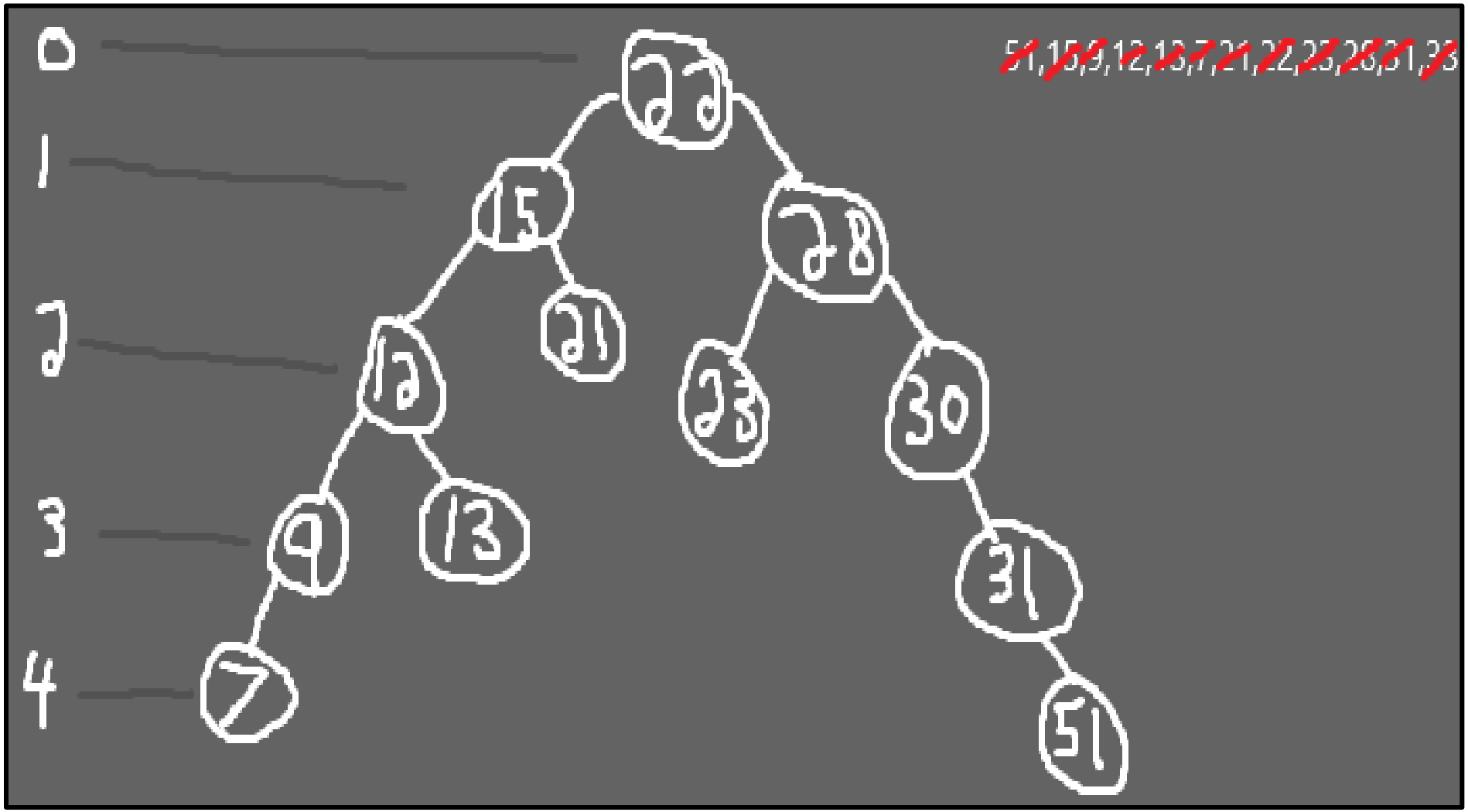
¿Cómo se balancean?

1. *Rotaciones:* Los árboles se pueden balancear mediante rotaciones (simples y dobles) para ajustar la estructura después de inserciones o eliminaciones.
2. *Reestructuración:* Al insertar o eliminar nodos, se pueden hacer ajustes en la estructura del árbol para mantener el balance.
3. *Árboles auto-balanceables:* Implementar árboles como AVL o rojo-negro que tienen reglas específicas y operaciones automáticas para mantener el balance.

5. "¿Qué ocurre cuando queremos eliminar la raíz? Por cual nodo puede ser reemplazada." **(JS)**

Al eliminar la raíz de un árbol, puede ser reemplazada por el nodo más pequeño del subárbol derecho (sucesor inorden) o por el nodo más grande del subárbol izquierdo (predecesor inorden). Esto asegura que la estructura del árbol se mantenga válida.

6. "Construir árbol binario de máximo nivel 4. Con los siguientes elementos 51,15,9,12,13,7,21,22,23,28,31,33." (JJ)



7. "Para el siguiente código añadirle eliminar nodo. Luego eliminar nodo 60. Correr in orden luego de la eliminación." (JJ)

```
73      void eliminar(int valor) { // Método público para eliminar un nodo
74          raiz = eliminarRec(raiz, valor);
75      }
76
77      Nodo eliminarRec(Nodo raiz, int valor) { // Método recursivo para eliminar
78          if (raiz == null) return raiz;
79
80          if (valor < raiz.valor) {
81              raiz.izquierdo = eliminarRec(raiz.izquierdo, valor);
82          } else if (valor > raiz.valor) {
83              raiz.derecho = eliminarRec(raiz.derecho, valor);
84          } else {
85              // Nodo con un hijo o sin hijos
86              if (raiz.izquierdo == null)
87                  return raiz.derecho;
88              else if (raiz.derecho == null)
89                  return raiz.izquierdo;
90
91              // Nodo con dos hijos: obtener el menor del subárbol derecho (sucesor)
92              raiz.valor = minValor(raiz.derecho);
93              raiz.derecho = eliminarRec(raiz.derecho, raiz.valor);
94          }
95
96          return raiz;
97      }
```

Lo que se añadió al Código

```
99      int minValor(Nodo nodo) { // Obtener el valor mínimo de un subárbol
100         int min = nodo.valor;
101         while (nodo.izquierdo != null) {
102             nodo = nodo.izquierdo;
103             min = nodo.valor;
104         }
105         return min;
106     }
```

Resultado:

```
128             // Eliminar nodo 60 y mostrar nuevo recorrido inorden
129             arbol.eliminar(60);
130             System.out.println("\n\nRecorrido en orden luego de eliminar 60:");
131             arbol.enOrden();
132             System.out.println(" ");
133         }
134     }
```

ArbolBinario > main >

Output - arbol (run) ×

```
run:
Recorrido en orden del arbol binario:
20 30 40 50 60 70 80
Recorrido pre orden del arbol binario:
50 30 20 40 70 60 80
Recorrido post orden del arbol binario:
20 40 30 60 80 70 50

Recorrido en orden luego de eliminar 60:
20 30 40 50 70 80
BUILD SUCCESSFUL (total time: 0 seconds)
```