TRABAJO INTEGRADOR
ÁRBOLES CON LISTAS

Alumnos: Emanuel Aaron Brahim Pollini (emanuelaaronb@gmail.com)
Nadia Celeste Almada (almada.nadia.c@gmail.com)

Materia: programación 1 Comisión 6
Profesor/a: Cinthia Rigoni y Oscar Londero
Fecha de Entrega: 09/06/2025
Repositorio: https://github.com/nadiaalmadac/Trabajo_Integrador_P1
Video: https://youtu.be/kgdVzj6vf-U

Índice

1. Introducción

En este Trabajo Práctico Final se abordará el tema de los árboles con listas en Python, una estructura de datos fundamental tanto en la programación como en la informática en general. Se comenzará presentando el marco teórico básico, incluyendo tres tipos principales de árboles: el **árbol binario**, el **árbol balanceado (AVL)** y el **árbol B**, cada uno con características particulares que los hacen útiles en diferentes contextos. Luego, se analizarán sus formas de recorrido, como el **inorden**, **preorden** y **postorden**, para comprender mejor su lógica de funcionamiento.

La elección de este tema se debe a su relevancia en la optimización del almacenamiento y acceso a los datos, así como a su frecuente aplicación en algoritmos de búsqueda, ordenamiento y estructuras jerárquicas. Además, se optó por implementar los árboles utilizando **listas**, ya que es el método trabajado durante la cursada, lo que permite enfocarse en el funcionamiento lógico de los árboles sin recurrir a estructuras más complejas como objetos o diccionarios.

Este trabajo tiene como objetivo comprender y explicar los conceptos fundamentales de los árboles, aplicar ese conocimiento en un desarrollo práctico con Python y lograr que el usuario pueda interactuar con el árbol: **insertando**, **recorriendo** y **eliminando** nodos de manera funcional. Para ello, se realizará una investigación teórica consultando fuentes confiables y se desarrollarán funciones que permitan visualizar claramente el comportamiento del árbol implementado

---

2. Marco Teórico

Un **árbol** es una estructura de datos **no lineal y jerárquica** formada por nodos conectados entre sí mediante aristas. Cada árbol posee un nodo raíz (root), del cual se ramifican nodos hijos, permitiendo representar relaciones jerárquicas. Esta estructura es ampliamente utilizada en informática para modelar sistemas de archivos, bases de datos y algoritmos de búsqueda y ordenamiento.

fuente: https://www.geeksforgeeks.org/tree-data-structure/

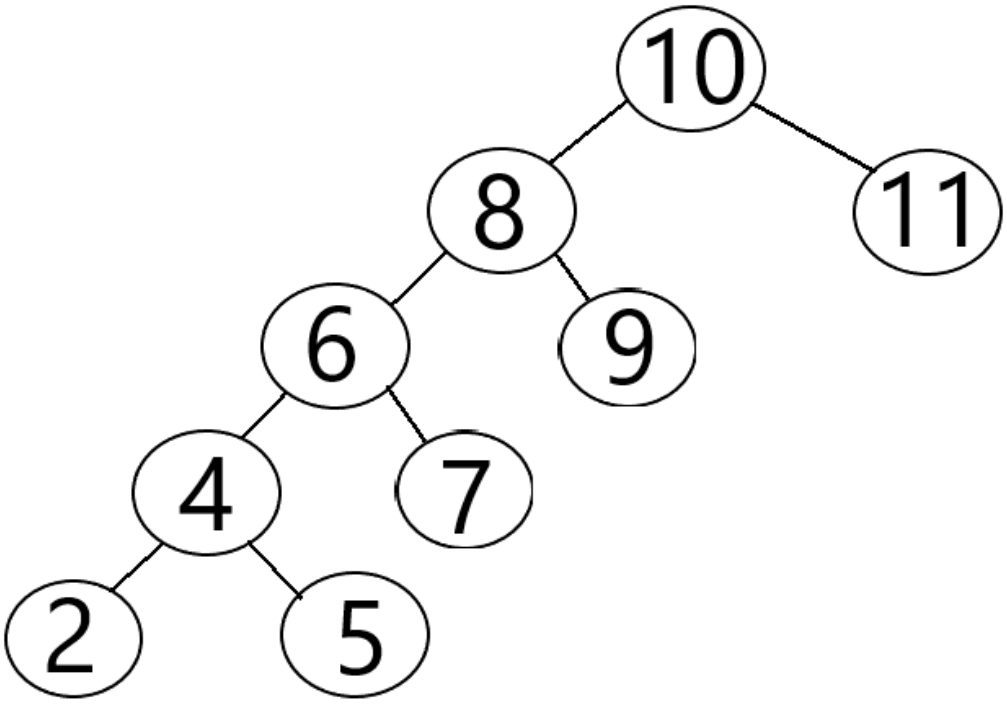
Tipos de árboles

Existen diversos tipos de árboles, entre los cuales se destacan:

- **Árbol Binario (Binary Tree):**
  Es una estructura en la cual cada nodo puede tener como máximo dos hijos:

un hijo izquierdo y un hijo derecho. Este tipo de árbol facilita operaciones como la búsqueda, la inserción y el recorrido.

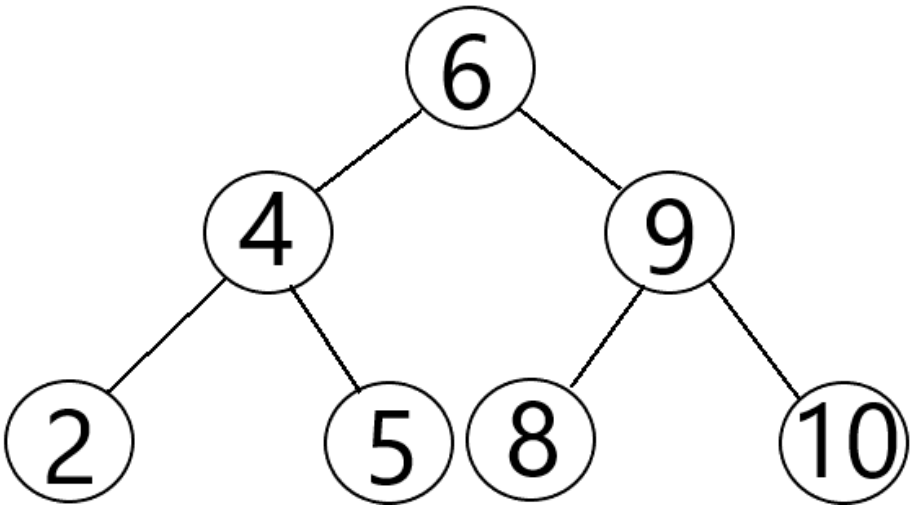
fuente: https://www.geeksforgeeks.org/introduction-to-binary-tree/



- **Árbol AVL (Adelson-Velsky y Landis Tree):**

Es un árbol binario de búsqueda balanceado automáticamente. Garantiza que la diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo no sea mayor que uno, asegurando eficiencia en operaciones de búsqueda, inserción y eliminación. Para mantener este equilibrio, los árboles AVL utilizan un conjunto de rotaciones automáticas cada vez que se inserta o elimina un nodo. Existen cuatro tipos de rotaciones: rotación simple a la derecha, rotación simple a la izquierda, rotación doble a la derecha (izquierda-derecha) y rotación doble a la izquierda (derecha-izquierda). Estas operaciones reestructuran el árbol de manera que la diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo nunca superé 1.

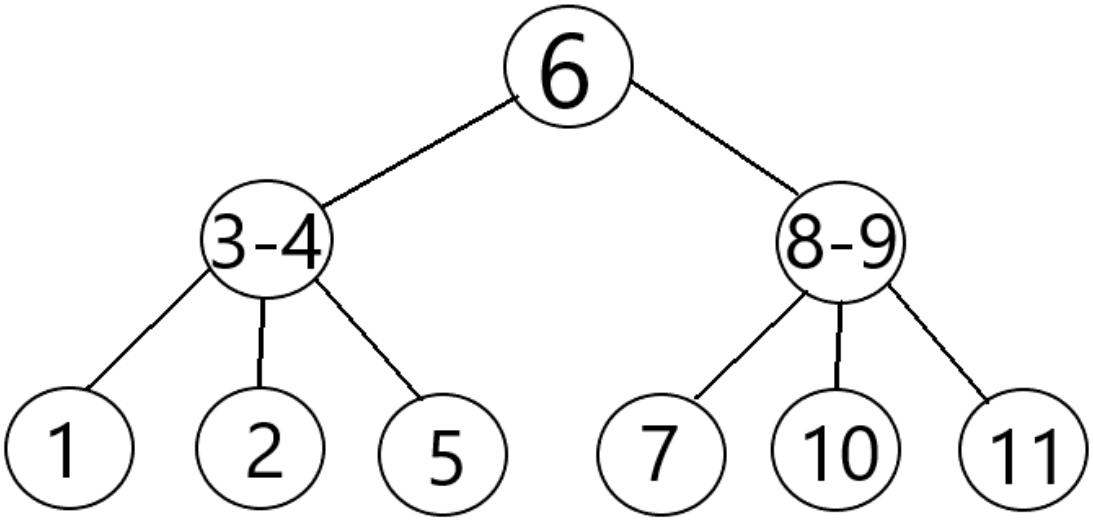
fuente: https://www.geeksforgeeks.org/insertion-in-an-avl-tree/



- **Árbol B (B-Tree):**

Se trata de una estructura de datos autobalanceada diseñada para manejar grandes cantidades de datos. A diferencia de los árboles binarios, los árboles B permiten múltiples hijos por nodo, lo que los hace ideales para sistemas de gestión de bases de datos y sistemas de archivos.

fuente: https://www.geeksforgeeks.org/introduction-of-b-tree-2/



Recorridos de árboles

Los **recorridos** permiten visitar todos los nodos de un árbol siguiendo un orden específico. Los más comunes son:

- **Inorden (In-order):** se recorre primero el subárbol izquierdo, luego el nodo raíz y finalmente el subárbol derecho. (Izquierda → Raíz → Derecha)

ejemplo basado en el grafico del Arbol AVL: 2,4,5,6,8,9,10

- **Preorden (Pre-order):** se visita primero la raíz, luego el subárbol izquierdo y después el derecho. (Raíz → Izquierda → Derecha)

ejemplo basado en el grafico del Arbol AVL: 6,4,2,5,9,8,10

- **Postorden (Post-order):** se recorre primero el subárbol izquierdo, luego el derecho y por último la raíz. (Izquierda → Derecha → Raíz)

ejemplo basado en el grafico del Arbol AVL: 2,5,4,8,10,9,6

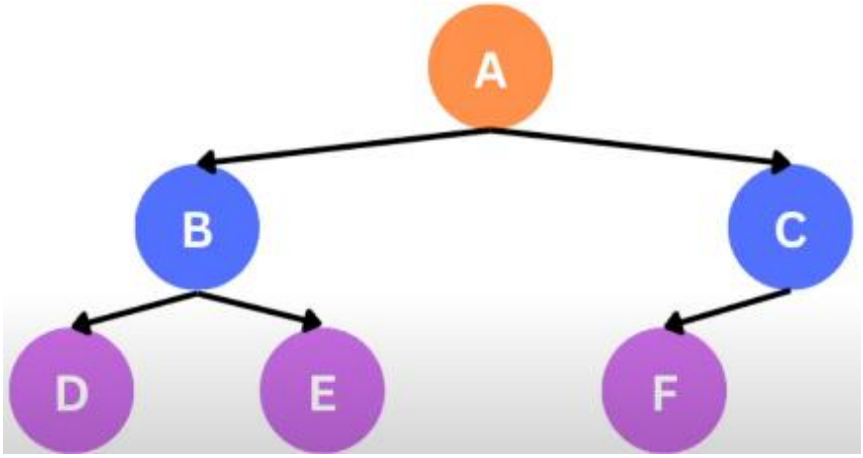
fuente: https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

Representación con listas en Python

En Python, una forma sencilla de representar árboles es mediante **listas anidadas**, sin necesidad de utilizar clases o diccionarios. Cada nodo se representa como una lista con tres elementos: el valor del nodo, el subárbol izquierdo y el subárbol derecho.

```
# Nodo con valor 1, hijo izquierdo 2 y derecho 3
arbol = [1,
    [2, [], []],
    [3, [], []]
   ]
```

Por ejemplo:




```
arbol = ["A", ["B", ["D", [], []], ["E", [], []]], ["C", ["F", [], []], []]]
```

Este método fue trabajado durante la cursada y permite enfocarse en la lógica del árbol sin estructuras complejas.

3. Caso Práctico

Desarrollo del Problema y Código Fuente Comentado

En este caso se eligió crear un programa con un menú simple para interactuar con el usuario. Este menú ofrece las siguientes opciones:

1. Agregar valores al árbol binario
2. Imprimir el árbol en recorrido Inorden
3. Imprimir el árbol en recorrido Preorden
4. Imprimir el árbol en recorrido Postorden
5. Salir del programa

Cuando se selecciona la opción para agregar valores al árbol, el programa solicita primero el valor para la raíz. Luego, se van agregando los hijos izquierdos y derechos en orden, comenzando por el subárbol izquierdo, luego el derecho, y así sucesivamente, siguiendo un recorrido en anchura. El programa también informa dónde se va a insertar el siguiente valor. Una vez finalizada la carga, el árbol se muestra en forma gráfica.

Código utilizado:

```
import os  # Librería recomendada para la función decorativa limpiar pantalla

# Funcion para limpiar la pantalla según el sistema operativo
def limpiar_pantalla():
    if os.name == 'nt':
        os.system('cls')  # Comando para Windows
    else:
        os.system('clear')  # Comando para Linux/Mac

# Función para imprimir el árbol en forma gráfica con indentación
def imprimir_arbol_grafico(arbol, nivel=0):
    if arbol == []:
        return
    imprimir_arbol_grafico(arbol[2], nivel + 1)
    print("    " * nivel + str(arbol[0]))
    imprimir_arbol_grafico(arbol[1], nivel + 1)
```

```
# Función para construir el árbol binario con datos ingresados por el usuario
def construir_arbol(arbol):
    pendientes = []

    while True:
        limpiar_pantalla()
        if arbol == []:
            print("El árbol está vacío. Ingrese el valor para la raíz.")
        else:
            for nodo in pendientes:
                if nodo[1] == []:
                    print(f"El próximo valor se agregará como hijo izquierdo de '{nodo[0]}'")
                    break
                elif nodo[2] == []:
                    print(f"El próximo valor se agregará como hijo derecho de '{nodo[0]}'")
                    break

        valor = input("Ingresá un valor para el árbol (o escribí 'salir' para terminar): ")
        if valor.lower() == 'salir':
            break

        if arbol == []:
            arbol += [valor, [], []]
            pendientes.append(arbol)
            print(f"Raíz creada con el valor '{valor}'")
            continue

        for nodo in pendientes:
            if nodo[1] == []:
                nodo[1] = [valor, [], []]
                pendientes.append(nodo[1])
                print(f"'{valor}' agregado como hijo izquierdo de '{nodo[0]}'")
                break
            elif nodo[2] == []:
                nodo[2] = [valor, [], []]
```

```
                pendientes.append(nodo[2])
                print(f"'{valor}' agregado como hijo derecho de '{nodo[0]}'")
                break

    limpiar_pantalla()
    print("Asi quedo el Arbol Binario:")
    imprimir_arbol_grafico(arbol)
    input("\nPresiona Enter para continuar...")

# Recorridos del árbol

def inorden(arbol):
    if arbol == []:
        return
    inorden(arbol[1])
    print(arbol[0], end=", ")
    inorden(arbol[2])

def preorden(arbol):
    if arbol == []:
        return
    print(arbol[0], end=", ")
    preorden(arbol[1])
    preorden(arbol[2])

def postorden(arbol):
    if arbol == []:
        return
    postorden(arbol[1])
    postorden(arbol[2])
    print(arbol[0], end=", ")

# Menú principal de interacción
def menu():
    arbol = []
    while True:
```

```
        limpiar_pantalla()
        print("\n--- MENÚ PRINCIPAL ---")
        print("1. Agregar valor al árbol")
        print("2. Imprimir árbol (Inorden)")
        print("3. Imprimir árbol (Preorden)")
        print("4. Imprimir árbol (Postorden)")
        print("5. Salir")

        opcion = input("Elegí una opción: ")

        if opcion == '1':
            construir_arbol(arbol)
        elif opcion == '2':
            inorden(arbol)
            input("\nPresiona Enter para continuar...")
        elif opcion == '3':
            preorden(arbol)
            input("\nPresiona Enter para continuar...")
        elif opcion == '4':
            postorden(arbol)
            input("\nPresiona Enter para continuar...")
        elif opcion == '5':
            break
        else:
            print("Opción inválida.")
            input("\nPresiona Enter para continuar...")

# Ejecutar el menú
menu()
```

Decisiones de diseño

Se decidió trabajar con **listas** en lugar de objetos o diccionarios, porque es el formato trabajado durante la cursada. Se utilizaron estructuras de control while, for y condicionales if para mantener el programa sencillo y accesible, con una lógica clara y lineal.

Validación del funcionamiento

El programa fue ejecutado múltiples veces, permitiendo:

- Agregar nodos correctamente en orden de amplitud.
- Visualizar el árbol en forma estructurada.
- Imprimir los valores del árbol según los tres recorridos (inorden, preorden, postorden).

Capturas de pantalla

Las siguientes capturas se incluyen en la carpeta imagenes/ del proyecto y también están disponibles en el repositorio:

- *ejecución del programa*
- *Selección de todas las opciones*
- *Ingreso exitoso de valores al arbol*

Las imágenes pueden visualizarse en el siguiente enlace:
https://github.com/nadiaalmadac/Trabajo_Integrador_P1

---

4. Metodología Utilizada

Primero se revisó el material provisto sobre árboles en la cursada de Programación 1. Luego, se ampliaron los conocimientos consultando información adicional en la página GeeksforGeeks, lo cual ayudó a comprender mejor tanto la teoría como su aplicación práctica.

Para el desarrollo del código, se comenzó diseñando la estructura general del programa, centrada en una función principal llamada menu, basada en un bucle while True. Esta función presenta opciones al usuario y, según la elección realizada, llama a la función correspondiente.

Una vez definido este diseño inicial, se desarrollaron las funciones encargadas de las distintas tareas: construir_arbol, imprimir_arbol_grafico, inorden, preorden y postorden. Además, se incorporó una función extraída de un foro para limpiar la pantalla de forma decorativa, utilizando la librería os.

El trabajo se realizó de manera colaborativa, manteniendo una comunicación constante mediante llamadas por Discord. De esta forma, se avanzó de manera

uniforme, con participación activa y aportes de ambos integrantes durante todo el proceso.

Para completar el proyecto se utilizaron las siguientes herramientas y recursos:

- Visual Studio Code con Python 3 para la programación.
- Discord y Google Colab para la organización y trabajo grupal.
- La librería os para detectar el sistema operativo y limpiar la pantalla.

---

5. Resultados Obtenidos

Como resultado del desarrollo de este trabajo práctico, se logró una mejor comprensión de la teoría básica sobre árboles y su aplicación práctica, logrando implementarla mediante un programa sencillo en Python que utiliza estructuras y conceptos trabajados a lo largo de la cursada.

Casos de prueba realizados:
Durante el proceso de codificación, se realizaron múltiples pruebas ejecutando el programa mientras se implementaban las funciones. Estas pruebas permitieron detectar errores y ajustar el comportamiento del árbol binario. Uno de los errores más significativos fue en la lógica de inserción: inicialmente se había planteado un bucle while True que provocaba que todos los valores se almacenaran únicamente en el subárbol izquierdo de la raíz, dejando el subárbol derecho vacío. Este inconveniente se solucionó rediseñando la lógica de inserción, utilizando una lista de nodos pendientes, como se muestra en la versión final del código.

Una vez finalizado el desarrollo, se realizaron pruebas con valores específicos, incluyendo los mismos del ejemplo ilustrado en la sección de árboles AVL del presente trabajo, lo que permitió validar el correcto funcionamiento del programa, tanto en la construcción como en las distintas formas de recorrido (inorden, preorden y postorden).

Repositorio: https://github.com/nadiaalmadac/Trabajo_Integrador_P1

---

6. Conclusiones

Al realizar este trabajo práctico, el grupo pudo afianzar y profundizar los conocimientos teóricos y prácticos sobre árboles binarios, un tema fundamental en estructuras de datos y algoritmos. La implementación en Python permitió comprender cómo manejar estructuras recursivas y listas anidadas para representar árboles, además de practicar técnicas de recorrido (inorden, preorden, postorden).

Este tema tiene gran utilidad en la programación, ya que los árboles son estructuras clave en bases de datos, sistemas de archivos, compiladores, y muchas otras aplicaciones donde la organización y búsqueda eficiente de información es fundamental.

Durante el desarrollo, se presentaron dificultades al diseñar la función de construcción del árbol, especialmente para gestionar la correcta asignación de nodos hijos. Estas se resolvieron mediante pruebas iterativas y ajustes en la lógica de control, así como la ayuda de recursos en línea y el trabajo colaborativo.

Como posibles mejoras a futuro, se podría reemplazar la estructura de datos basada en listas por diccionarios, lo cual facilitaría la manipulación y acceso a los nodos del árbol. Además, se podría brindar al usuario mayor libertad al ingresar nuevos valores, permitiéndole especificar la ubicación exacta donde desea agregar un nodo. También sería conveniente implementar funcionalidades para eliminar valores incorrectos o ya obsoletos, entre otras mejoras que aumentarían la flexibilidad y robustez del programa.

---

7. Bibliografía

- Programadora (2024). Videos de teoría sobre árboles binarios y estructuras de datos. YouTube. https://www.youtube.com/watch?v=jN78OB9d8Uk (consultado el 7 de junio de 2025) https://www.youtube.com/watch?v=-D4SxeHQGIg (consultado el 7 de junio de 2025)
- GeeksforGeeks. (s.f.). Tree Data Structure. https://www.geeksforgeeks.org/tree-data-structure/ (consultado el 7 de junio de 2025)
- GeeksforGeeks. (s.f.). Introduction to Binary Tree. https://www.geeksforgeeks.org/introduction-to-binary-tree/ (consultado el 7 de junio de 2025)
- GeeksforGeeks. (s.f.). Insertion in an-AVL Tree. https://www.geeksforgeeks.org/insertion-in-an-avl-tree/ (consultado el 7 de junio de 2025)
- GeeksforGeeks. (s.f.). Introduction of B-Tree. https://www.geeksforgeeks.org/introduction-of-b-tree-2/ (consultado el 7 de junio de 2025)
- GeeksforGeeks. (s.f.). Tree Traversals — Inorder, Preorder and Postorder. https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/ (consultado el 7 de junio de 2025)
- *Repositorio del Trabajo Práctico Final* – GitHub. https://github.com/nadiaalmadac/Trabajo_Integrador_P1

---

8. Anexos

Se adjunta el siguiente repositorio de GitHub que contiene:

- Capturas del código en ejecución

- Archivos fuente del script y del informe.

- Link del Video Explicativo https://youtu.be/kgdVzj6vf-U

Repositorio: https://github.com/nadiaalmadac/Trabajo_Integrador_P1