Trabajo de investigación: Implementación de Árboles en Python utilizando listas

Alumnos:

- García Galfione, Fiorella

  fiorella_gargal@hotmail.com

- Escobar, Joaquin

  joa.escobar04@gmail.com

Docente: Bruselario, Sebastián

Materia: Programación I

Fecha de entrega: 09/06/2025

ÍNDICE

1. INTRODUCCIÓN

En programación, las estructuras de datos permiten organizar la información de manera eficiente para facilitar su manipulación. Una de las estructuras más relevantes en informática es el árbol, especialmente el árbol binario, por su utilidad en algoritmos de búsqueda, bases de datos, inteligencia artificial y sistemas de archivos.
El presente trabajo tiene como objetivo explorar la implementación de árboles binarios en Python utilizando listas anidadas. La elección de este tema surge del interés por comprender en profundidad una de las estructuras de datos más importantes en el campo de la programación. Los árboles permiten organizar la información de forma jerárquica y eficiente, siendo fundamentales en áreas como bases de datos, inteligencia artificial, algoritmos de búsqueda y compiladores.

Aunque tradicionalmente se utilizan clases y objetos para representar árboles en lenguajes orientados a objetos, este trabajo se enfoca en una alternativa didáctica: el uso de listas anidadas. Esta aproximación permite a los estudiantes adquirir una comprensión clara de la lógica estructural de los árboles, sin la necesidad de incorporar conceptos complejos de programación orientada a objetos desde el inicio.

El objetivo principal de este trabajo es demostrar cómo construir, recorrer e imprimir un árbol binario utilizando listas en Python. A través del desarrollo práctico, se busca afianzar conceptos fundamentales sobre estructuras de datos, fomentar el pensamiento recursivo y evidenciar cómo Python permite múltiples paradigmas de programación. Se espera que, al finalizar el trabajo, el lector cuente con una herramienta más para resolver problemas estructurados jerárquicamente, empleando técnicas simples pero potentes.

2. MARCO TEÓRICO

¿Qué es un Árbol?

Un **árbol** es una estructura de datos **jerárquica** compuesta por nodos conectados entre sí. Cada árbol tiene un **nodo raíz** (root), y cada nodo puede tener **cero o más hijos**. Un **árbol binario** es una variante particular en la que cada nodo tiene **como máximo dos hijos**, denominados izquierdo y derecho. Según Cormen et al. (2009), un árbol se compone de un nodo raíz y subnodos organizados de forma que cada elemento (salvo la raíz) tiene un único antecesor.

2.1 Propiedades de los Árboles Binarios

- **Raíz**: Nodo principal desde el cual se accede al resto del árbol
- **Nodos internos**: nodos con al menos un hijo.
- **Hojas**: Nodos sin hijos, situados en los extremos.
- **Altura**: Número de niveles desde la raíz hasta la hoja más profunda.
- **Subárboles**: Cualquier nodo con sus descendientes puede considerarse un subárbol independiente.

2.2 Aplicaciones de los Árboles

Los árboles binarios son fundamentales en la computación por sus aplicaciones en:

- Búsquedas eficientes (como en árboles binarios de búsqueda).
- Representación de expresiones matemáticas (árboles de expresión).
- Sistemas de archivos jerárquicos.
- Algoritmos de parsers en compiladores.
- Inteligencia artificial (árboles de decisión).

2.3 Representación con Listas en Python

Python, al ser un lenguaje multiparadigma, permite implementar árboles binarios tanto con clases (enfoque orientado a objetos) como con **listas anidadas**, lo que facilita el aprendizaje inicial.

La estructura básica de un nodo utilizando listas se define como:

[valor, subarbol_izquierdo, subarbol_derecho]

Por ejemplo, el nodo `'A'` con hijos `'B'` e `'C'` se representaría así:

```
['A', ['B', [], []], ['C', [], []]]
```

Cada subárbol también es una lista del mismo formato. Esta representación evita el uso de clases, lo que permite concentrarse en la lógica estructural del árbol sin distracciones de la POO.

2.4 Operaciones Básicas

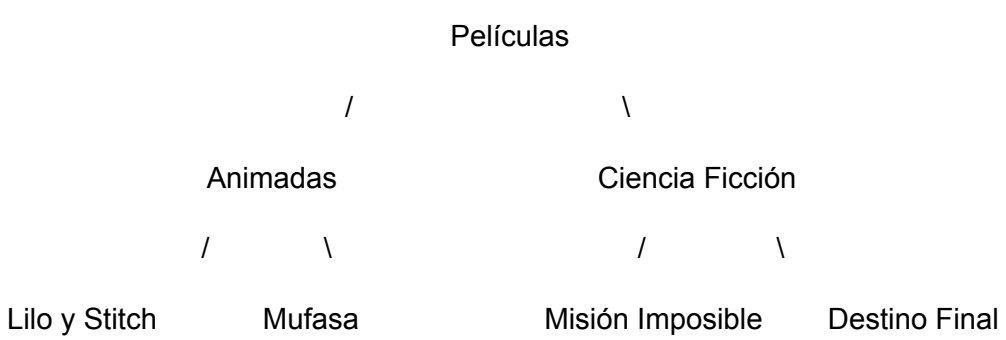
Entre las operaciones básicas que se pueden realizar sobre árboles binarios, se encuentran:

- **Inserción de nodos**: tanto a la izquierda como a la derecha.
- **Recorridos**:
    - **Preorden**: raíz → izquierda → derecha
    - **Inorden**: izquierda → raíz → derecha
    - **Postorden**: izquierda → derecha → raíz

Estas operaciones pueden implementarse de forma recursiva utilizando listas.

3. CASO PRÁCTICO

Se plantea la creación de un árbol con películas.



```
# TP INTEGRADOR - Árbol binario de películas - implementado con listas

# Función para crear un nuevo árbol (nodo raíz)


def crear_arbol(valor):

    # Cada nodo es una lista con tres elementos: [valor, subárbol izquierdo, subárbol derecho]

    return [valor, [], []]


# Función para insertar un nodo a la izquierda

def insertar_izquierda(nodo, valor):

    # Si ya existe un subárbol izquierdo, lo anida como hijo del nuevo nodo

    if nodo[1]:

        nodo[1] = [valor, nodo[1], []]

    else:

        # Si no hay subárbol izquierdo, simplemente lo inserta

        nodo[1] = [valor, [], []]


# Función para insertar un nodo a la derecha

def insertar_derecha(nodo, valor):

    # Si ya existe un subárbol derecho, lo anida como hijo del nuevo nodo

    if nodo[2]:

        nodo[2] = [valor, [], nodo[2]]

    else:

        # Si no hay subárbol derecho, simplemente lo inserta
```

```
        nodo[2] = [valor, [], []]


# Función para recorrer el árbol en preorden: raíz → izquierda → derecha
def recorrido_preorden(nodo):
    if nodo:
        print(nodo[0], end=' ')          # Imprime el valor del nodo actual
        recorrido_preorden(nodo[1])      # Recorre subárbol izquierdo
        recorrido_preorden(nodo[2])      # Recorre subárbol derecho


# Función para recorrer el árbol en inorden: izquierda → raíz → derecha
def recorrido_inorden(nodo):
    if nodo:
        recorrido_inorden(nodo[1])       # Recorre subárbol izquierdo
        print(nodo[0], end=' ')          # Imprime el valor del nodo actual
        recorrido_inorden(nodo[2])       # Recorre subárbol derecho


# Función para recorrer el árbol en postorden: izquierda → derecha → raíz
def recorrido_postorden(nodo):
    if nodo:
        recorrido_postorden(nodo[1])     # Recorre subárbol izquierdo
        recorrido_postorden(nodo[2])     # Recorre subárbol derecho
        print(nodo[0], end=' ')          # Imprime el valor del nodo actual
```

```
# Función para imprimir el árbol rotado 90° hacia la izquierda
def imprimir_arbol(nodo, nivel=0):
    if nodo:
        imprimir_arbol(nodo[2], nivel + 1)                          # Muestra
primero el subárbol derecho (más arriba)
        print('    ' * nivel + str(nodo[0]))                        # Imprime
el nodo actual con sangría según su nivel
        imprimir_arbol(nodo[1], nivel + 1)                          # Luego
muestra el subárbol izquierdo


# Ejemplo práctico: creación de un árbol de películas
if __name__ == "__main__":
    # Nivel 1 - Nodo raíz
    arbol = crear_arbol("Películas")

    # Nivel 2 - Se agregan las dos categorías principales
    insertar_izquierda(arbol, "Animadas")
    insertar_derecha(arbol, "Ciencia Ficción")

    # Nivel 3 - Películas dentro de la categoría "Animadas"
    insertar_izquierda(arbol[1], "Lilo y Stich")
    insertar_derecha(arbol[1], "Mufasa")

    # Nivel 3 - Películas dentro de la categoría "Ciencia Ficción"
    insertar_izquierda(arbol[2], "Misión Imposible")
    insertar_derecha(arbol[2], "Destino Final")
```

```python
# Visualización del árbol rotado 90°
print(" Árbol rotado 90°:")
imprimir_arbol(arbol)

# Recorridos del árbol
print(" Recorrido Preorden:")
recorrido_preorden(arbol)

print(" Recorrido Inorden:")
recorrido_inorden(arbol)

print(" Recorrido Postorden:")
recorrido_postorden(arbol)
```

```
> & C:/Users/fiore/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/fiore/OneDrive/Documentos/Programación I/TP INTEGRADOR/TP_integrador.py"
 Árbol rotado 90°:
      Destino Final
   Ciencia Ficción
      Misión Imposible
Películas
      Mufasa
   Animadas
      Lilo y Stich
 Recorrido Preorden:
Películas Animadas Lilo y Stich Mufasa Ciencia Ficción Misión Imposible Destino Final  Recorrido Inorden:
Lilo y Stich Animadas Mufasa Películas Misión Imposible Ciencia Ficción Destino Final  Recorrido Postorden:
Lilo y Stich Mufasa Animadas Misión Imposible Destino Final Ciencia Ficción Películas
PS C:\Users\fiore\OneDrive\Documentos\Programación I>
```

4. METODOLOGÍA UTILIZADA

El desarrollo del trabajo se llevó a cabo siguiendo una metodología estructurada en distintas etapas, que permitieron abordar el contenido desde la teoría hasta la implementación práctica. A continuación, se detallan los pasos seguidos:

4.1. Investigación previa

En una primera instancia, se realizó una búsqueda de información en fuentes bibliográficas y digitales para comprender el concepto de árboles binarios y su relevancia en programación. Algunas de las fuentes consultadas son libros como *"Introduction to Algorithms"* de Cormen et al. (2009), documentación oficial de Python (python.org), y artículos académicos.

4.2. Diseño y desarrollo del código

Con base en la información recopilada, se diseñó una estructura de árbol binario utilizando listas anidadas. Se definieron funciones básicas para:

- Crear nodos raíz.
- Insertar nodos a la izquierda o a la derecha.
- Realizar recorridos (preorden, inorden y postorden).
- Imprimir visualmente el árbol rotado.

4.3. Prueba y validación

El código fue desarrollado y probado en el entorno **Visual Studio Code**, utilizando el intérprete de Python 3.12. Durante esta etapa, se validó el correcto funcionamiento de cada operación mediante pruebas individuales y un caso práctico que simula un árbol de clasificación de películas. Se incorporaron funciones auxiliares para facilitar la visualización y comprensión del árbol resultante.

4.4. Herramientas utilizadas

- **Lenguaje de programación**: Python 3.12.
- **Entorno de desarrollo (IDE)**: Visual Studio Code.

- **Control de versiones**: Git

- **Fuentes de documentación**: Sitios web oficiales, bibliografía académica, tutoriales en línea.

5. RESULTADOS OBTENIDOS

A partir del desarrollo del caso práctico propuesto, se logró implementar correctamente un árbol binario en Python utilizando listas anidadas. La estructura definida permitió representar nodos y subárboles de manera clara y funcional.

5.1. Caso práctico implementado

Se construyó un árbol que clasifica distintos tipos de películas. La raíz del árbol es el nodo "Películas", con dos ramas principales: "Animadas" y "Ciencia Ficción". A su vez, cada categoría incluye dos películas representadas como nodos hoja. El árbol fue construido paso a paso mediante funciones de inserción y validado visualmente utilizando la función imprimir_arbol, que rota el árbol 90° para facilitar su lectura.

5.2. Funcionalidades probadas

Se probaron exitosamente las siguientes operaciones:

- **Creación de árbol**: mediante la función crear_arbol( ).

- **Inserción de nodos**: tanto a izquierda como derecha.

- **Recorridos**: preorden, inorden y postorden, con salida por consola.

- **Visualización estructural**: impresión jerárquica del árbol.

Las pruebas confirmaron que las funciones recorrían correctamente todos los nodos en el orden esperado, y que la estructura del árbol se mantenía consistente luego de cada inserción.

5.3. Dificultades y errores corregidos

Durante la etapa de desarrollo, se presentaron algunas dificultades:

- **Sobrescritura de nodos**: en las primeras versiones de insertar_izquierda y insertar_derecha, al insertar sobre un nodo que ya tenía un hijo, se perdía la rama existente. Esto se corrigió ajustando las funciones para que el nuevo nodo se anide respetando el subárbol anterior.

- **Visualización invertida**: en la impresión rotada, se invirtieron accidentalmente las ramas izquierda y derecha. La lógica fue corregida para reflejar fielmente la jerarquía esperada.

5.5. Repositorio del proyecto

Se empleó Git de forma local para registrar versiones del código y facilitar la colaboración entre los integrantes del grupo. En el repositorio creado se encuentra el archivo.py, un archivo readme.md con información de los estudiantes y el enlace al video explicativo del trabajo.

6. CONCLUSIONES

Este trabajo permitió al grupo comprender en profundidad el funcionamiento de los árboles binarios y su implementación práctica en Python utilizando listas anidadas. A través del proceso de investigación, diseño y prueba del código, se adquirieron conocimientos valiosos sobre estructuras de datos jerárquicas, pensamiento recursivo y organización eficiente de la información. El uso de listas como alternativa a la programación orientada a objetos resultó ser una herramienta didáctica efectiva, especialmente en etapas de formación inicial. Este enfoque facilitó la construcción lógica del árbol sin requerir conocimientos avanzados sobre clases, herencia o punteros, lo cual contribuyó a una mejor comprensión del concepto central. En cuanto a la utilidad del tema, los árboles binarios son ampliamente aplicables en múltiples áreas de la programación, desde algoritmos de búsqueda y clasificación hasta inteligencia artificial y diseño de compiladores. Por ello, dominar su implementación, incluso en formas simples como las abordadas en este trabajo, representa una base sólida para el desarrollo de proyectos más complejos en el futuro.Durante el proceso surgieron algunas dificultades técnicas, principalmente relacionadas con la inserción de nodos sin sobrescribir ramas existentes y con la visualización estructurada del árbol. Estos problemas fueron resueltos mediante pruebas reiteradas y ajustes progresivos en el código, lo cual reforzó habilidades de depuración y trabajo colaborativo.

Finalmente, se reconocen posibles mejoras futuras al proyecto, como incorporar validaciones automáticas para evitar duplicación de nodos, implementar recorridos iterativos, o rediseñar la estructura usando clases para dotarla de mayor escalabilidad y reutilización.

En resumen, el trabajo cumplió con los objetivos propuestos y permitió a los integrantes afianzar conceptos teóricos clave mediante una aplicación práctica concreta, demostrando el valor de los árboles binarios como una herramienta esencial en la programación.

7. BIBLIOGRAFÍA

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3.ª ed.). MIT Press.

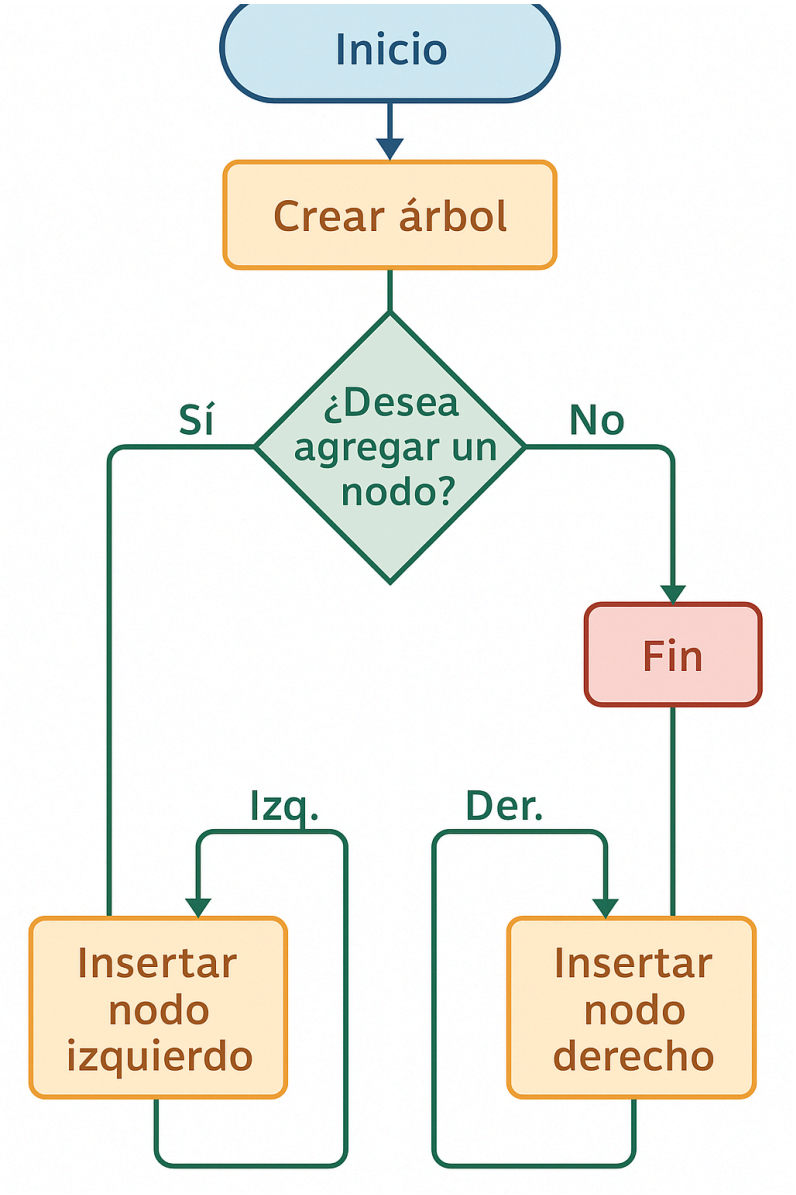
- Python Software Foundation. (2024). *Python 3 Documentation*. https://docs.python.org/3/

- Javier Fernandes (2021). Árboles binarios. https://sites.google.com/site/programacioniiuno/temario/unidad-5---grafos/rboles-binarios

8. ANEXOS

- Enlace al video explicativo: https://youtu.be/ePj73viRtJ0?si=6zipmrsYVqvRWEVX
- Enlace del repositorio subido a Git:
    - https://github.com/1sharko1/TP-Programacion-
    - https://github.com/FioreGG/UTN-TUPaD-P1/Programacion/TP_Integrador
- Diagrama de flujo de un árbol binario

    El diagrama de flujo representa de forma visual y simplificada el proceso de creación e inserción de nodos en un árbol binario utilizando listas en Python. Este flujo ilustra la lógica que se aplicó en el código para construir el árbol dinámicamente, permitiendo agregar nodos recursivamente mientras el usuario (o el programa) lo desee.


Inicio
Crear árbol
¿Desea agregar un nodo?
Sí
No
Fin
Izq.
Der.
Insertar nodo izquierdo
Insertar nodo derecho