Carátula

Título del proyecto:
Implementación y recorrido de árboles binarios en Python

Alumnos:
Juan Pérez (juanperez@email.com)
Camila Gómez (camilagomez@email.com)

Materia:
Programación I

Profesor:
Lic. Andrés Muñoz

Fecha de Entrega:
10 de mayo de 2025

---

1. Introducción

Las estructuras de datos permiten organizar y almacenar información de manera eficiente. Dentro de ellas, los árboles son fundamentales para resolver múltiples problemas: búsquedas rápidas, organización jerárquica y toma de decisiones.

En esta investigación se estudian los árboles binarios, su implementación en Python, distintos recorridos y casos prácticos de aplicación.

---

2. Marco Teórico

¿Qué es un Árbol?

Un árbol es una estructura jerárquica que consiste en nodos conectados:

- Cada nodo tiene un valor y puede tener 0, 1 o 2 hijos en el caso de árboles binarios.
- El primer nodo se llama **raíz**.
- Los nodos sin hijos se llaman **hojas**.

Tipos de recorridos de un árbol binario:

- **In-Orden** (Izquierda - Raíz - Derecha)
- **Pre-Orden** (Raíz - Izquierda - Derecha)
- **Post-Orden** (Izquierda - Derecha - Raíz)

Aplicaciones de árboles:

- Bases de datos.
- Compiladores.

- Juegos de inteligencia artificial.
- Motores de búsqueda.

---

3. Caso Práctico

Se implementa un árbol binario simple en Python, insertando nodos manualmente y mostrando los distintos tipos de recorrido.

Código principal (arbol_binario.py):

```
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izquierda = None
        self.derecha = None


class ArbolBinario:
    def __init__(self):
        self.raiz = None


    def insertar(self, valor):
        if self.raiz is None:
            self.raiz = Nodo(valor)
        else:
            self._insertar_recursivo(self.raiz, valor)


    def _insertar_recursivo(self, nodo_actual, valor):
        if valor < nodo_actual.valor:
            if nodo_actual.izquierda is None:
                nodo_actual.izquierda = Nodo(valor)
            else:
                self._insertar_recursivo(nodo_actual.izquierda, valor)
        else:
            if nodo_actual.derecha is None:
```

```
            nodo_actual.derecha = Nodo(valor)
        else:
            self._insertar_recursivo(nodo_actual.derecha, valor)

    def recorrido_inorden(self, nodo):
        if nodo:
            self.recorrido_inorden(nodo.izquierda)
            print(nodo.valor, end=" ")
            self.recorrido_inorden(nodo.derecha)

    def recorrido_preorden(self, nodo):
        if nodo:
            print(nodo.valor, end=" ")
            self.recorrido_preorden(nodo.izquierda)
            self.recorrido_preorden(nodo.derecha)

    def recorrido_postorden(self, nodo):
        if nodo:
            self.recorrido_postorden(nodo.izquierda)
            self.recorrido_postorden(nodo.derecha)
            print(nodo.valor, end=" ")

if __name__ == "__main__":
    arbol = ArbolBinario()
    valores = [50, 30, 70, 20, 40, 60, 80]

    for v in valores:
        arbol.insertar(v)

    print("Recorrido Inorden:")
    arbol.recorrido_inorden(arbol.raiz)
```

```
print("\nRecorrido Preorden:")
arbol.recorrido_preorden(arbol.raiz)
print("\nRecorrido Postorden:")
arbol.recorrido_postorden(arbol.raiz)
```

Resultado esperado (salida en consola):

Recorrido Inorden:

20 30 40 50 60 70 80

Recorrido Preorden:

50 30 20 40 70 60 80

Recorrido Postorden:

20 40 30 60 80 70 50

---

4. Metodología Utilizada

- Estudio de las características de los árboles binarios.
- Implementación manual de nodos e inserciones en Python.
- Programación de funciones recursivas para recorridos Inorden, Preorden y Postorden.
- Testeo con una lista predefinida de valores.

---

5. Resultados Obtenidos

- El árbol binario fue construido correctamente con nodos insertados según su valor.
- Los recorridos Inorden, Preorden y Postorden se realizaron de manera correcta.
- Se pudo visualizar el orden en que se accede a los nodos en cada tipo de recorrido.

---

6. Conclusiones

El uso de árboles binarios en Python facilita la organización de datos de manera eficiente. Los algoritmos de recorrido permiten explorar la estructura según las necesidades de cada problema.

Reflexión: Aprender estructuras de datos como árboles no solo mejora las habilidades de programación, sino que también enseña a pensar de manera estructurada para resolver problemas complejos.

7. Bibliografía

- "Introduction to Algorithms", Cormen et al.
- Documentación de Python: https://docs.python.org/3/tutorial/classes.html
- Visualgo.net (herramienta para visualizar estructuras): https://visualgo.net/en/bst

8. Anexos

- Capturas de pantalla del código funcionando en consola.
- Repositorio en GitHub: https://github.com/grupo-arboles/estructura-arbol-python
- Video explicativo mostrando la estructura del árbol, los recorridos y reflexión final.