📝 **Trabajo de Investigación: Implementación de Árboles en Python Utilizando Listas**

---

1. Introducción

Las estructuras de datos son fundamentales en la programación, ya que permiten almacenar y manipular información de forma eficiente. Entre ellas, los **árboles** son estructuras jerárquicas ampliamente utilizadas en algoritmos de búsqueda, bases de datos, inteligencia artificial y más.

Python, un lenguaje de alto nivel y flexible, permite representar estructuras complejas de diversas maneras. Este trabajo explora una forma alternativa de implementar árboles **sin clases ni objetos**, utilizando únicamente **listas anidadas**, aprovechando la capacidad de las listas de contener otras listas.

---

2. Objetivos

- Comprender cómo se puede representar un árbol binario utilizando listas en Python.
- Implementar operaciones básicas sobre árboles: inserción y recorridos.
- Evaluar las ventajas y desventajas de este enfoque frente a implementaciones orientadas a objetos.
- Promover el uso de representaciones simples como herramienta educativa.

---

3. Marco Teórico

Un **árbol** es una estructura de datos no lineal compuesta por nodos. El nodo superior se llama **raíz**, y cada nodo puede tener cero o más nodos hijos.

Un caso particular es el **árbol binario**, donde cada nodo tiene como máximo dos hijos: izquierdo y derecho.

Propiedades clave de los árboles binarios:

- **Raíz**: nodo inicial.
- **Hojas**: nodos sin hijos.
- **Altura**: número de niveles desde la raíz hasta la hoja más profunda.
- **Subárboles**: cualquier nodo junto a sus descendientes forma un subárbol.

Las implementaciones típicas en lenguajes como Java o C++ utilizan punteros y clases. En Python, una alternativa es usar listas del tipo [valor, subárbol_izquierdo, subárbol_derecho], donde cada subárbol también es una lista similar.

---

4. Metodología

- Se utilizó el lenguaje **Python 3.x**.

- Cada nodo del árbol se representa como una lista de tres elementos:
    1. Valor del nodo.
    2. Subárbol izquierdo.
    3. Subárbol derecho.
- Se desarrollaron funciones para crear árboles, insertar nodos y recorrerlos.
- Se incluyó una función de impresión rotada del árbol para facilitar la visualización.

---

5. Desarrollo / Implementación

A continuación, se presenta el código completo para implementar un árbol binario utilizando únicamente listas:

```
# Cada nodo se representa como: [valor, hijo_izquierdo, hijo_derecho]


def crear_arbol(valor):
    return [valor, [], []]


def insertar_izquierda(nodo, nuevo_valor):
    subarbol_izq = nodo[1]
    if subarbol_izq:
        nodo[1] = [nuevo_valor, subarbol_izq, []]
    else:
        nodo[1] = [nuevo_valor, [], []]


def insertar_derecha(nodo, nuevo_valor):
    subarbol_der = nodo[2]
    if subarbol_der:
        nodo[2] = [nuevo_valor, [], subarbol_der]
    else:
        nodo[2] = [nuevo_valor, [], []]


def preorden(arbol):
    if arbol:
```

```
        print(arbol[0], end=' ')
        preorden(arbol[1])
        preorden(arbol[2])

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

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

def imprimir_arbol(arbol, nivel=0):
    if arbol:
        imprimir_arbol(arbol[2], nivel + 1)
        print('   ' * nivel + str(arbol[0]))
        imprimir_arbol(arbol[1], nivel + 1)

# ----------------------
# Ejemplo de uso
# ----------------------

# Crear nodo raíz
arbol = crear_arbol('A')

# Insertar hijos
insertar_izquierda(arbol, 'B')
```

```
insertar_derecha(arbol, 'C')

insertar_izquierda(arbol[1], 'D')
insertar_derecha(arbol[1], 'E')

insertar_izquierda(arbol[2], 'F')
insertar_derecha(arbol[2], 'G')

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

# Recorridos
print("\nRecorrido Preorden:")
preorden(arbol)

print("\nRecorrido Inorden:")
inorden(arbol)

print("\nRecorrido Postorden:")
postorden(arbol)
```

Salida esperada:

```
mathematica
CopiarEditar
Árbol visualizado (rotado 90°):
    G
  C
    F
A
    E
  B
```

D

Recorrido Preorden:

A B D E C F G

Recorrido Inorden:

D B E A F C G

Recorrido Postorden:

D E B F G C A

---

6. Resultados

La implementación permite la construcción y recorrido de un árbol binario de manera clara y funcional. Se observó que, si bien la implementación con listas es menos escalable que una basada en clases, su simplicidad la convierte en una herramienta excelente para la enseñanza inicial de estructuras de datos.

Ventajas:

- Sencillez conceptual.
- Bajo nivel de complejidad sintáctica.
- Ideal para estudiantes principiantes.

Desventajas:

- Menor flexibilidad y extensibilidad.
- Ausencia de encapsulamiento.
- Dificultad para implementar árboles balanceados o genéricos.

---

7. Conclusión

La implementación de árboles mediante listas en Python ofrece una alternativa simple pero poderosa para comprender la estructura y operación de árboles binarios. Aunque no es adecuada para aplicaciones a gran escala, sí lo es como herramienta didáctica.

Este enfoque permite centrarse en la lógica estructural del árbol sin distracciones de programación orientada a objetos. Se recomienda su uso en cursos introductorios de estructuras de datos o en entornos donde se requiera una solución rápida y sencilla.

---

8. Bibliografía

- Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). *Introduction to Algorithms*.
- Python Software Foundation. https://docs.python.org
- Miller, B. & Ranum, D. *Problem Solving with Algorithms and Data Structures using Python*.