Trabajo de Investigación: Análisis e Implementación de Árboles Binarios con Módulos en Python

1. Introducción

Las estructuras de datos son fundamentales para el desarrollo de algoritmos eficientes. Entre ellas, los arboles binarios destacan por su utilidad en busquedas, inteligencia artificial, parsers y mas. Este trabajo presenta una implementacion modular de arboles binarios en Python utilizando listas anidadas, dividiendo responsabilidades por funcionalidad en distintos modulos, a fin de facilitar su reutilizacion y mantenimiento.

2. Objetivos

- Representar un arbol binario mediante listas anidadas.

- Implementar funciones para recorrido, analisis de nodos y rutas entre nodos.

- Promover una arquitectura modular para la ensenanza y mantenimiento del codigo.

- Comparar el enfoque estructurado por modulos con implementaciones tradicionales.

3. Marco Teorico

Un arbol binario es una estructura de datos jerarquica en la que cada nodo tiene como maximo dos hijos. Los conceptos clave incluyen:

- Raiz: el nodo principal del arbol.

- Hoja: nodo sin hijos.

- Rama: nodo con al menos un hijo, que no es raiz.

- Nivel/Profundidad: distancia desde la raiz.

- Grado: numero de hijos de un nodo.

En lugar de utilizar clases y objetos, en esta implementacion se emplean listas del tipo:

```
["A", ["B", [], []], ["C", [], []]]
```

4. Metodologia

- Se desarrollaron modulos independientes en Python:

  * tree.py contiene la estructura del arbol.

* print_utils.py incluye funciones de recorrido y visualizacion.

* node_utils.py analiza nodos y encuentra rutas.

* main.py maneja la interaccion del usuario por menu.

- El arbol se construyo manualmente mediante listas anidadas, sin clases.

- Los recorridos implementados fueron: preorden, inorden, y postorden.

- Se anadieron funciones avanzadas como busqueda de caminos entre nodos y analisis del tipo de nodo.

5. Desarrollo / Implementacion

Estructura del arbol (archivo tree.py):

```
tree = [
  "A",
    ["B", ["D", ["H", [], []], ["I", [], []]], ["E", ["J", [], []], []]],
    ["C", ["F", [], ["K", [], []]], ["G", ["L", [], []], ["M", [], []]]]
]
```

Recorridos (print_utils.py):

```
def preorder(tree): ...
def inorder(tree): ...
def postorder(tree): ...
```

Visualizacion estructurada:

```
def tree_structure(tree, nivel=0): ...
```

Analisis de nodos (node_utils.py): funciones como find_path_between, degree, is_leaf, etc.

Interfaz (main.py): Menu interactivo por consola para explorar el arbol.

6. Resultados

La implementacion modular permite trabajar con arboles de forma clara y extensible.

Ventajas:

- Modularidad y legibilidad.

- Interactividad via consola.

- Funciones completas de analisis y visualizacion.

- Util en el ambito educativo.

Desventajas:

- Requiere conocimientos previos sobre modulos y estructuras anidadas.

- No incluye inserciones dinamicas en tiempo real.

- Dificil de escalar a arboles balanceados o genericos.

7. Conclusion

El uso de listas anidadas para representar arboles binarios en Python es una herramienta potente para el aprendizaje inicial de estructuras de datos. Cuando se combina con una arquitectura modular como en este proyecto, se logra una solucion didactica, clara y facil de mantener. Aunque no es adecuada para proyectos a gran escala, si lo es para educacion, pruebas o prototipos.

8. Bibliografia

- Cormen, T. et al. Introduction to Algorithms (2009).

- Python Software Foundation – https://docs.python.org

- Miller, B. & Ranum, D. Problem Solving with Algorithms and Data Structures using Python