

Trabajo Práctico Integrador N°2

Programación I

Tema elegido: Estructuras de datos avanzados – Árboles en Python utilizando listas

Alumno

Agustín Hurtado

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Organización Empresarial

8 de junio de 2025

Introducción

Las estructuras de datos en programación son formas específicas de organizar y almacenar información en una computadora para que pueda ser utilizada de forma eficiente. Permite al desarrollador manejar, acceder, modificar y manipular datos de forma ordenada, facilitando la solución. Son fundamentales para manejar grandes cantidades de datos, como en bases de datos o servicios de indexación en internet. Hay muchos tipos de estructuras de datos, los más simples son los vectores (arrays), luego están las matrices (array de arrays), listas enlazadas (linked lists), etc. Saber que estructura utilizar en cada caso es una facultad clave que debe desarrollar el programador. Hoy pondremos el foco en los árboles (trees).

Se ha elegido abordar este tema por la gran importancia que tiene esta estructura de datos en el mundo de la programación. Los árboles son utilizados en una amplia variedad de aplicaciones en el mundo real, ya sea como la organización de archivos en sistemas operativos, la gestión de bases de datos (por ejemplo, en árboles de búsqueda binaria), en algoritmos de optimización, análisis léxico, inteligencia artificial (AI, por sus siglas en inglés) y más. La propuesta de este trabajo es estudiar y aplicar una forma alternativa de implementar árboles binarios en Python utilizando listas anidadas, en lugar de objetos o clases.

El objetivo principal es integrar los conceptos teóricos con una implementación práctica en Python, haciendo uso de funciones para así construir árboles, insertar nodos, recorrerlos, imprimirlos, y reflexionar sobre sus ventajas y limitaciones.

Objetivos

- Comprender la estructura y funcionamiento de los árboles binarios.
- Implementar funciones para insertar y recorrer árboles binarios representados mediante listas.
- Analizar las ventajas y desventajas del uso de listas, frente a enfoques orientados a objetos.
- Desarrollar un ejemplo práctico funcional y documentado en Python.
- Reflexionar sobre la utilidad didáctica de este enfoque como herramienta para el aprendizaje de estructuras de datos.

Marco Teórico

Un árbol es una estructura de datos no lineal compuesta por nodos organizados jerárquicamente. Cada nodo contiene un dato y referencias a sus nodos hijos, formando así , una estructura ramificada que comienza en el nodo principal denominado raíz (root). Los nodos sin hijos se llaman hojas (leaves), y los que tienen al menos un hijo son nodos internos. Cada nodo puede no tener hijos, o tener uno o más hijos. Uno de los tipos más comunes y usados de árboles, es el árbol binario, donde cada nodo tiene como máximo dos hijos: el izquierdo y el derecho. La terminología de los nodos es la siguiente:

Nodos: Se le llama Nodo a cada elemento que contiene un Árbol.

Nodo Raíz: Se refiere al primer nodo de un Árbol, Solo un nodo del Árbol puede ser la Raíz.

Nodo Padre: Se utiliza este término para llamar a todos aquellos nodos que tiene al menos un hijo.

Nodo Hijo: Los hijos son todos aquellos nodos que tiene un padre.

Nodo Hermano: Los nodos hermanos son aquellos nodos que comparte a un mismo padre en común dentro de la estructura.

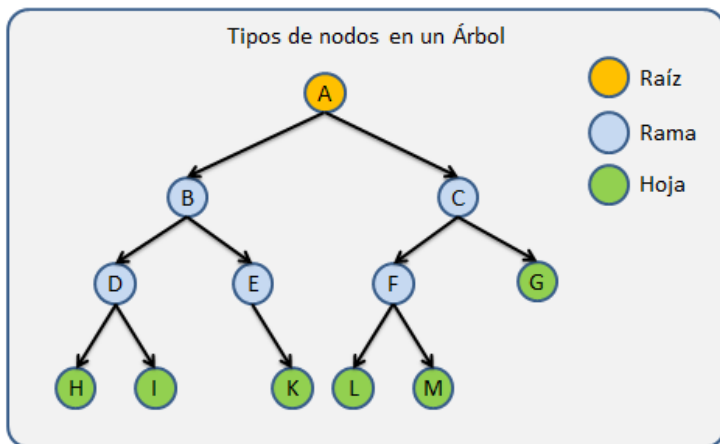
Nodo Hoja: Son todos aquellos nodos que no tienen hijos, los cuales siempre se encuentran en los extremos de la estructura.

Nodo Rama: Estos son todos aquellos nodos que no son la raíz y que además tiene al menos un hijo.

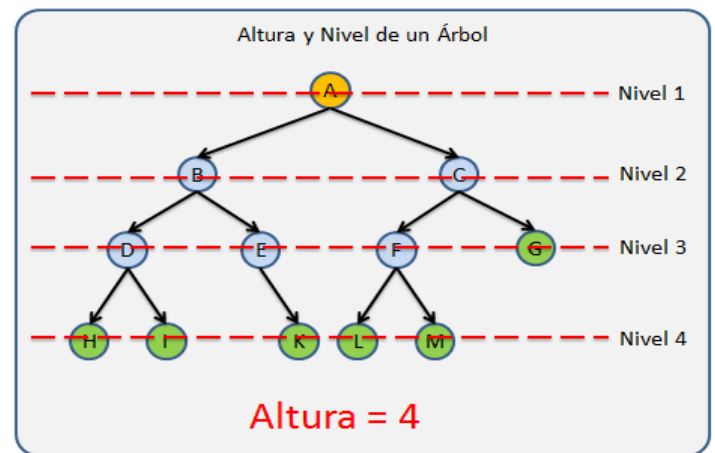
Los árboles tienen **niveles** (a menos de que sea un árbol vacío), cada nivel se refiere a cada generación dentro del árbol. Si a un nodo hoja, le agregamos un hijo, el nodo hoja pasará a ser nodo rama, y además el árbol habrá crecido una generación, por lo que ahora tendrá un nivel más. Además

se le llama **altura** al número máximo de niveles de un árbol. Por otro lado, tenemos al **peso**, el cual se conoce como el número de nodos que tiene un árbol. Este factor es importante porque nos da una idea del tamaño del árbol y el tamaño en memoria que nos puede ocupar en tiempo de ejecución. El **orden** de un árbol es el número máximo de hijos que puede tener un nodo. Por último, se conoce como **subárbol** a todo árbol generado a partir de una sección determinada del árbol. Se puede decir entonces que un árbol es un nodo raíz con N subárboles.

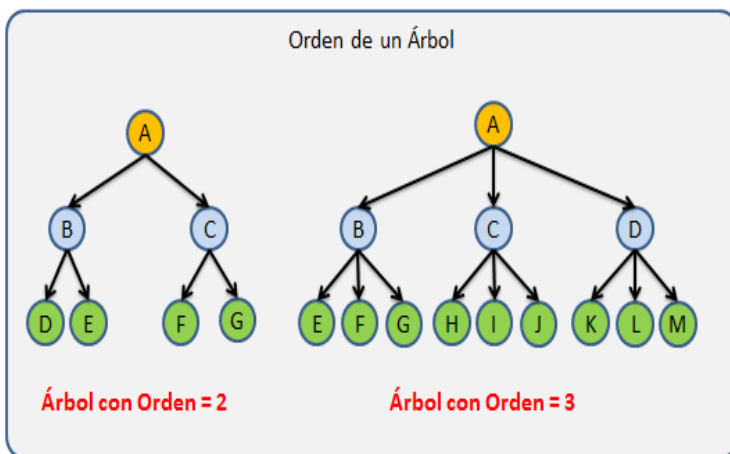
Img 1



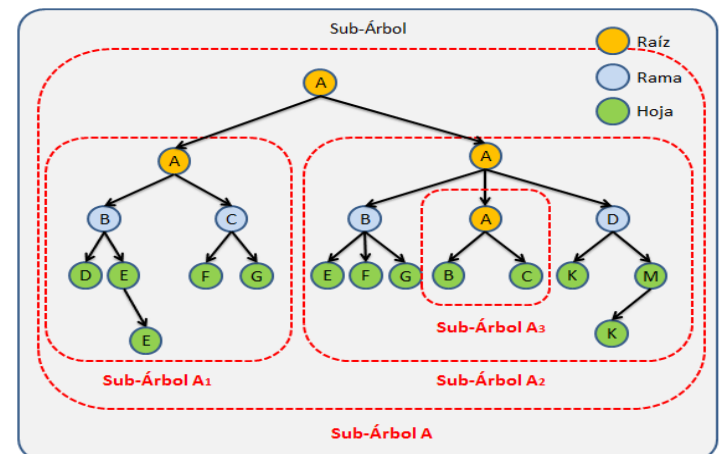
Img 2



Img 3



Img 4



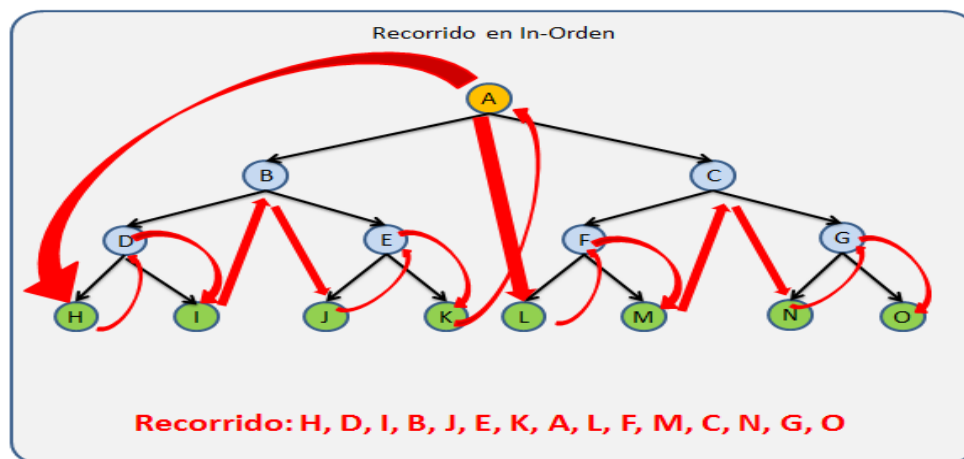
Hablando desde una perspectiva técnica del uso de árboles en programación, sabemos que en lenguajes como C++ o Java, es común representar árboles usando estructuras de clases y punteros. Sin embargo, en Python es posible utilizar listas anidadas, aprovechando que en este lenguaje las listas pueden contener otras listas. Esta técnica representa un nodo como una lista de tres elementos: [valor, subarbol_izquierdo, subarbol_derecho].

Esta forma de representación es particularmente útil cuando queremos explicar el concepto de estructuras de datos como árboles con fines educativos, ya que permite focalizarse en la lógica de la estructura sin la complejidad adicional de la programación orientada a objetos. No obstante, presenta limitaciones cuando se desea escalar o manipular árboles más complejos.

Los árboles binarios pueden ser **recorridos** de diferentes maneras, cada una con sus propias aplicaciones y características. Los tres métodos principales son:

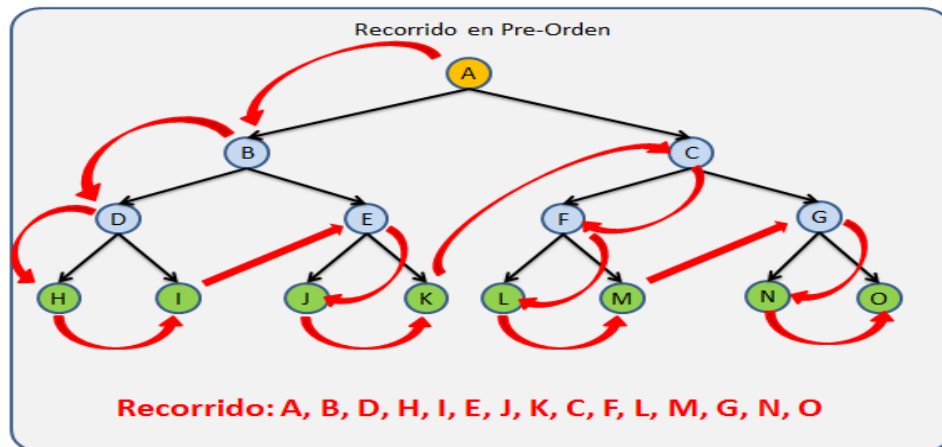
In-Orden (Izquierda - Raíz - Derecha):

- Primero se recorre el subárbol izquierdo
- Luego se visita la raíz
- Finalmente se recorre el subárbol derecho
- Es útil para obtener los valores en orden ascendente en árboles de búsqueda binaria

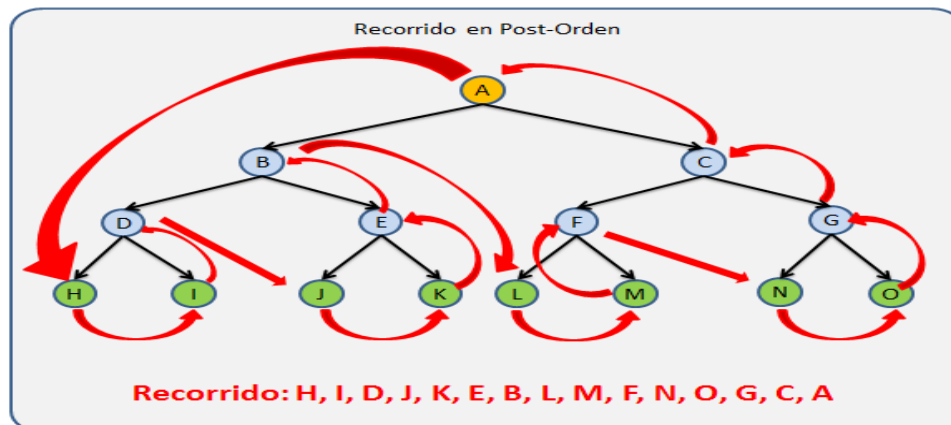


Pre-Orden (Raíz - Izquierda - Derecha):

- Primero se visita la raíz
- Luego se recorre el subárbol izquierdo
- Finalmente se recorre el subárbol derecho
- Útil para crear una copia del árbol o para expresiones prefijas

**Post-Orden (Izquierda - Derecha - Raíz):**

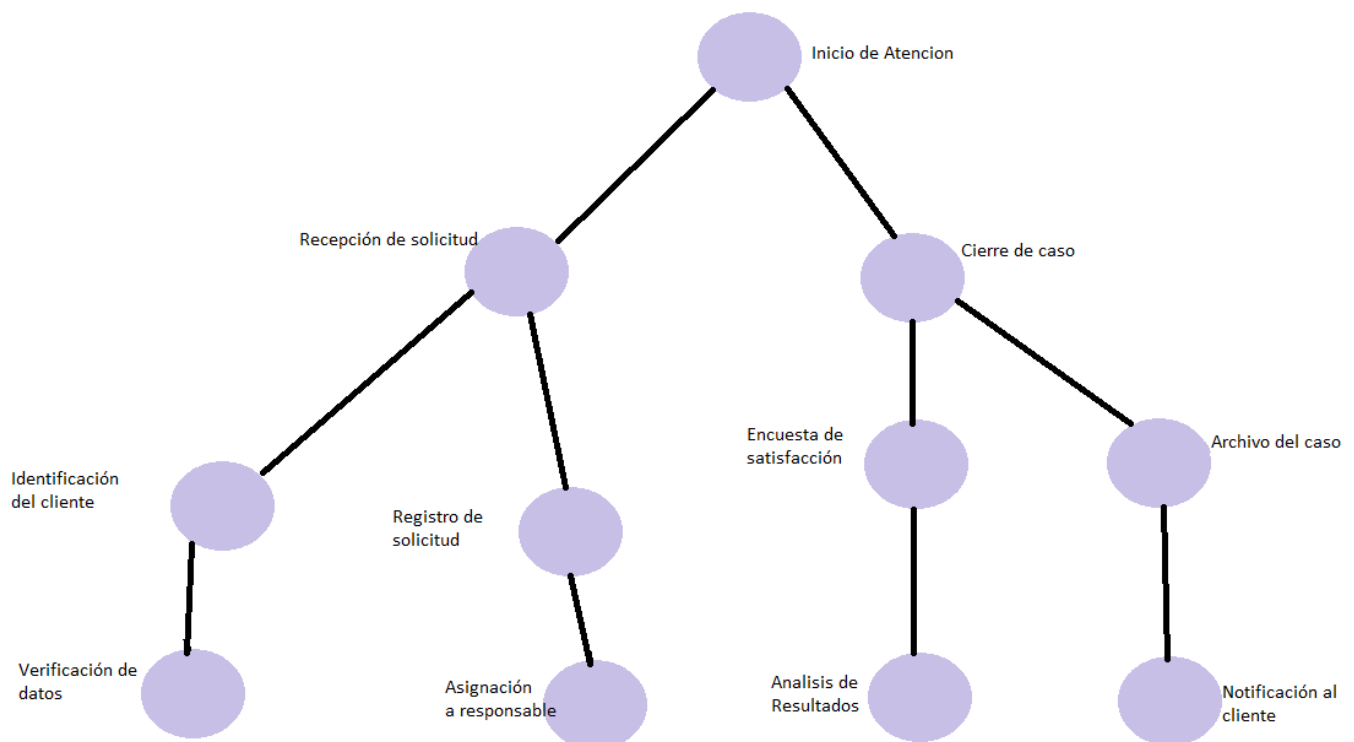
- Primero se recorre el subárbol izquierdo
- Luego se recorre el subárbol derecho
- Finalmente se visita la raíz
- Útil para eliminar el árbol o para expresiones postfijas



Caso práctico

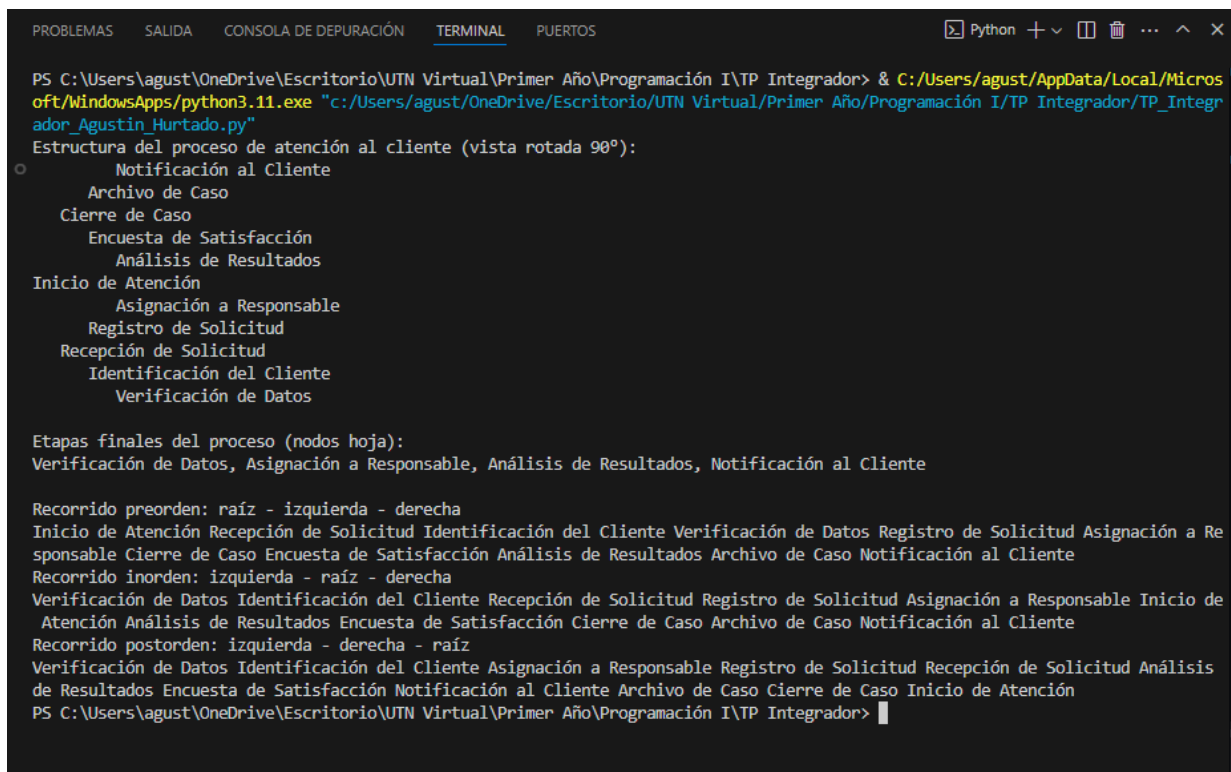
La situación en la que nos situaremos será en una simulación y análisis del flujo de atención al cliente en una empresa de servicios. La situación o problema es el siguiente: Una empresa de servicios desea mejorar la eficiencia y la satisfacción del cliente en su proceso de atención. Para ello, quiere modelar y visualizar las etapas del proceso, identificar claramente las tareas finales (sin subdivisiones) y analizar el orden en que se ejecutan las actividades.

El árbol binario representa las etapas del proceso, desde la "Inicio de Atención" hasta el "Cierre de Caso", pasando por subprocessos como "Recepción de Solicitud", "Registro de Solicitud", "Encuesta de Satisfacción", etc. La estructura permite visualizar la jerarquía y ramificación del proceso mediante la impresión rotada, facilitando la comprensión del flujo.



Se eligió una representación sencilla usando listas con tres elementos: valor, subárbol izquierdo y subárbol derecho. Esto facilita la manipulación directa y la comprensión didáctica del árbol sin requerir clases o estructuras complejas, ideal para ejemplificar jerarquías y procesos con dos ramas principales por nodo. No se usaron listas planas o diccionarios porque no permiten representar fácilmente la relación jerárquica padre-hijo con dos subárboles diferenciados. Se implementaron los tres recorridos clásicos (pre-orden, in-orden, post-orden) para mostrar distintas formas de visitar los nodos. Por último, la impresión rotada 90° facilita la visualización jerárquica en consola, mostrando claramente la estructura del proceso con indentación que refleja niveles, lo que es más legible que una impresión lineal.

Funcionamiento



```
PS C:\Users\agust\OneDrive\Escritorio\UTN Virtual\Primer Año\Programación I\TP Integrador> & C:/Users/agust/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/agust/OneDrive/Escritorio/UTN Virtual/Primer Año/Programación I/TP Integrador/TP_Integrador_Agustin_Hurtado.py"
Estructura del proceso de atención al cliente (vista rotada 90°):
├── Notificación al Cliente
│   ├── Archivo de Caso
│   ├── Cierre de Caso
│   ├── Encuesta de Satisfacción
│   └── Análisis de Resultados
└── Inicio de Atención
    ├── Asignación a Responsable
    ├── Registro de Solicitud
    ├── Recepción de Solicitud
    ├── Identificación del Cliente
    └── Verificación de Datos

Etapas finales del proceso (nodos hoja):
Verificación de Datos, Asignación a Responsable, Análisis de Resultados, Notificación al Cliente

Recorrido preorden: raíz - izquierda - derecha
Inicio de Atención Recepción de Solicitud Identificación del Cliente Verificación de Datos Registro de Solicitud Asignación a Responsable Cierre de Caso Encuesta de Satisfacción Análisis de Resultados Archivo de Caso Notificación al Cliente
Recorrido inorden: izquierda - raíz - derecha
Verificación de Datos Identificación del Cliente Recepción de Solicitud Registro de Solicitud Asignación a Responsable Inicio de Atención Análisis de Resultados Encuesta de Satisfacción Cierre de Caso Archivo de Caso Notificación al Cliente
Recorrido postorden: izquierda - derecha - raíz
Verificación de Datos Identificación del Cliente Asignación a Responsable Registro de Solicitud Recepción de Solicitud Análisis de Resultados Encuesta de Satisfacción Notificación al Cliente Archivo de Caso Cierre de Caso Inicio de Atención
PS C:\Users\agust\OneDrive\Escritorio\UTN Virtual\Primer Año\Programación I\TP Integrador>
```


Metodología

El trabajo fue realizado en torno a la investigación y aplicación práctica de estructuras de datos avanzadas. Numerosas fuentes fueron utilizadas y quedaran detalladas en la bibliografía. El foco está puesto en los árboles binarios representados mediante listas anidadas en Python. Ha sido utilizada la versión de Python 3 en Visual Studio Code.

La metodología consistió en los siguientes pasos:

1. Selección del enfoque: se eligió representar árboles sin clases ni objetos, utilizando listas de la forma [valor, subárbol izquierdo, subárbol derecho] por su simplicidad y valor didáctico.
2. Investigación teórica: se consultaron fuentes académicas y documentación oficial de Python para comprender la estructura y operaciones fundamentales de los árboles.
3. Diseño del caso práctico: se propuso como ejemplo una estructura jerárquica de atención al cliente representada mediante un árbol binario, simulando niveles de dirección, áreas y empleados.
4. Desarrollo e implementación: se programaron funciones para crear nodos, insertar hijos, realizar recorridos (pre-orden, in-orden, post-orden), e imprimir el árbol con formato visual horizontal y rotado 90°.
5. Visualización y pruebas: se generaron distintas salidas en consola para validar el funcionamiento del árbol, identificando correctamente la raíz, los subárboles y los nodos hoja.
6. Documentación y reflexión: se organizaron los resultados, observaciones y aprendizajes en este documento y se preparó un archivo README junto al código fuente documentado.

Análisis de Resultados

Se pudo demostrar la funcionalidad de los árboles binarios representados con listas. Se logró:

- Crear estructuras jerárquicas claras, incluyendo distintos niveles de profundidad.
- Identificar fácilmente la raíz, los subárboles y los nodos hoja mediante funciones recursivas.
- Visualizar el árbol rotado 90°, lo que facilita la comprensión de las relaciones jerárquicas.
- Mantener bajo nivel de complejidad sintáctica, favoreciendo la claridad del código para estudiantes que se inician en estructuras de datos.

A pesar de que esta implementación con listas es limitada en cuanto a la escalabilidad y flexibilidad, resulta muy útil a la hora de ilustrar conceptos básicos de árboles binarios en un entorno educativo.

Se implementaron los tres tipos principales de recorridos de árboles binarios (in-orden, pre-orden y post-orden), demostrando cómo cada uno produce un orden diferente al visitar los nodos. Esto resulta particularmente útil para diferentes aplicaciones: el recorrido in-orden es ideal para obtener una secuencia ordenada, el pre-orden para copiar la estructura del árbol, y el post-orden para operaciones que requieren procesar los hijos antes que el nodo padre, como en el cálculo de expresiones matemáticas.

Conclusión

Durante la realización de este trabajo, se aprendió a representar estructuras jerárquicas complejas mediante árboles binarios en Python, utilizando listas anidadas y funciones para manipular y recorrer el árbol. También reforzamos conceptos fundamentales de estructuras de datos y su aplicación práctica en contextos empresariales.

El uso de árboles binarios es una base importante en programación para organizar datos jerárquicos y realizar recorridos eficientes. Este conocimiento es aplicable en múltiples áreas, como la gestión de procesos, sistemas contables, o cualquier proyecto que requiera modelar relaciones padre-hijo o flujos de trabajo. Además, la capacidad de simular y analizar procesos mediante estructuras de datos contribuye a la mejora continua.

Para futuras mejoras, se podría migrar la implementación a una estructura orientada a objetos para mejorar la modularidad y escalabilidad. También sería útil ampliar el modelo para árboles n-arios, permitiendo más de dos subprocesos por etapa, reflejando procesos más complejos. Otra extensión valiosa es incorporar funcionalidades de búsqueda, edición y eliminación de nodos, o integrar la visualización gráfica mediante librerías especializadas para facilitar aún más el análisis. Finalmente, se podría conectar este modelo con bases de datos o sistemas reales para simular datos en tiempo real y evaluar el impacto de cambios en el proceso.

Por último, lo que rescataría de este tipo de trabajos prácticos, es que impulsan a buscar información, investigar, practicar temas que uno no conoce, asemejándose a situaciones que viven los programadores día a día.

Bibliografía

- Python Software Foundation. <https://docs.python.org>
- [Video de Charly Cimino](#)
- [Video de Vida MRR - Programación web](#)
- [Blog de Oscar Blancarte](#)
- [Árboles](#)

Anexo

Link video de la presentación: <https://drive.google.com/file/d/1qSoA0FmY-LIsnuXI1NGTpYZ3uo-vJDzE/view?usp=sharing>