TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

Trabajo integrador Programación I

Implementación de árboles con listas anidadas

Alumnos:

- Luis Cisneros - luis.cisneros@tupad.utn.edu.ar
- Guillermo Campoy - guillermo.campoy@tupad.utn.edu.ar

Profesor: **Ariel Enferrel**
Fecha de Entrega: **16/06/2025**

Repositorio
https://github.com/guillecampoy/UTN-TUaD-Integrador-Programacion-I

Video youtube
https://www.youtube.com/watch?v=XH0ylZCb_Rw

Índice

Introducción

El presente trabajo se centra en la implementación de árboles binarios utilizando listas anidadas, una temática seleccionada por el equipo debido a su relevancia en el contexto de la materia y a su potencial para consolidar los conocimientos adquiridos durante la cursada. Entre las diferentes opciones propuestas, esta fue considerada particularmente enriquecedora, ya que permite integrar conceptos clave de estructuras de datos de manera práctica, al mismo tiempo que introduce desafíos que estimulan el razonamiento lógico y la planificación estructurada del código.

La elección de este tema responde a la intención de profundizar en un tipo de estructura que resulta fundamental dentro del campo de la programación. Los árboles binarios, por su naturaleza jerárquica y su capacidad para representar relaciones padre-hijo, son ampliamente utilizados en una variedad de aplicaciones informáticas. Desde sistemas de organización de información hasta motores de búsqueda, pasando por algoritmos de compresión y sistemas de toma de decisiones, su versatilidad los convierte en una herramienta esencial en el desarrollo de software. Particularmente en contextos como el análisis de riesgos, los motores de scoring crediticio o los sistemas de prevención de fraudes, el uso de árboles permite modelar reglas y evaluar condiciones de forma eficiente y extensible.

El enfoque adoptado en este trabajo busca no solo construir la estructura de árbol utilizando listas anidadas, sino también aplicar y demostrar diversos algoritmos de recorrido —preorden, inorden y postorden— detallando su funcionamiento. Además, se incorporan conceptos fundamentales abordados en clase como la recursividad y la validación de datos ingresados por teclado mediante bucles controlados. Como complemento, se integró una propuesta opcional que permite visualizar gráficamente el árbol generado, haciendo uso de librerías estándar de Python, con el objetivo de reforzar la comprensión del comportamiento interno de la estructura implementada.

El trabajo persigue como principal objetivo evidenciar la comprensión y correcta aplicación de los contenidos desarrollados durante la materia, promoviendo un aprendizaje activo mediante la construcción de soluciones concretas que podrían ser la base para proyectos más complejos en el futuro.

Marco Teórico

¿Qué es un árbol en programación?

Un árbol es una estructura de datos fundamental en informática que organiza información de manera jerárquica. A diferencia de las estructuras lineales como arrays o listas enlazadas, los árboles permiten representar relaciones más complejas donde un elemento puede tener múltiples "descendientes".

Características principales:

No linealidad: No existe una secuencia simple de primero a último
Jerarquía: Los nodos tienen relaciones padre-hijo claramente definidas
Direccionalidad: Las conexiones van siempre de padres a hijos (no hay ciclos)
Analogías del mundo real:
Árbol genealógico
Sistema de archivos de un ordenador
Organigrama empresarial

Elementos básicos de un árbol

Nodo raíz

Es el nodo superior del árbol, el único que no tiene padre. Todo árbol tiene exactamente un nodo raíz. Desde este nodo se puede acceder a todos los demás.

Nodos hoja (o terminales)

Son nodos que no tienen hijos. Representan los "finales" de las ramas del árbol. En muchos contextos, contienen los datos reales (por ejemplo, en un árbol de expresión, las hojas serían los operandos).

Nodos internos

Son nodos que tienen al menos un hijo. Estos nodos generalmente contienen información que organiza o estructura los datos (en un árbol de expresión, serían los operadores).

Relaciones familiares

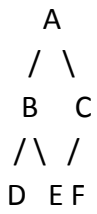
Padre: Nodo que tiene uno o más nodos hijos
Hijo: Nodo que tiene un nodo padre
Hermanos: Nodos que comparten el mismo padre
Ancestros: Todos los nodos en el camino desde la raíz hasta el padre del nodo actual
Descendientes: Todos los nodos que se pueden alcanzar desde el nodo actual siguiendo las relaciones hijo

Representaciones de un árbol

1. Representación gráfica

La más intuitiva, donde se dibujan nodos como círculos y las relaciones como flechas o líneas. Es excelente para visualizar pero poco práctica para implementar en código.

Ejemplo:



2. Conjuntos anidados

Se representa como conjuntos que contienen otros conjuntos. Muy útil en bases de datos para consultas jerárquicas.

Ejemplo: {A, {B, {D}, {E}}, {C, {F}}}

3. Paréntesis anidados

Similar a la notación polaca o la forma en que se escriben expresiones matemáticas.

Ejemplo: (A (B (D)(E))(C (F)))

4. Indentación

Muy usado en configuraciones o para mostrar estructuras de archivos.

Ejemplo:

```
A
  B
    D
    E
  C
    F
```

Propiedades de los árboles

Camino

Secuencia de nodos que conectan dos nodos en el árbol. La longitud del camino es el número de aristas (conexiones) entre ellos.

Ejemplo: En A→B→E, el camino de A a E tiene longitud 2.

Profundidad

Número de aristas desde la raíz hasta el nodo. La raíz tiene profundidad 0.

Nivel

Profundidad + 1. La raíz está en el nivel 1.

Altura

Máxima profundidad de cualquier nodo en el árbol. Es decir, la longitud del camino más largo desde la raíz hasta una hoja.

Grado

Número de hijos directos que tiene un nodo. En un árbol binario, el grado máximo es 2.

Orden

Máximo número de hijos que puede tener cualquier nodo en el árbol. Un árbol binario es de orden 2.

Peso

Número total de nodos en el árbol.

Árbol binario

Es un árbol donde cada nodo tiene como máximo dos hijos, llamados convencionalmente hijo izquierdo e hijo derecho.

Características especiales:

Cada nodo tiene 0, 1 o 2 hijos
El subárbol izquierdo y derecho son a su vez árboles binarios
Muy eficientes para búsquedas cuando están balanceados (O(log n))

Recorridos de árboles

Los recorridos son formas sistemáticas de visitar todos los nodos del árbol. Para árboles binarios hay tres recorridos principales:

1. Preorden (Raíz-Izquierdo-Derecho)

Visita la raíz
Recorre el subárbol izquierdo en preorden
Recorre el subárbol derecho en preorden
Ejemplo: A, B, D, E, C, F

2. Inorden (Izquierdo-Raíz-Derecho)

Recorre el subárbol izquierdo en inorden
Visita la raíz
Recorre el subárbol derecho en inorden
Ejemplo: D, B, E, A, F, C
Útil para obtener la expresión original en árboles de expresión.

3. Postorden (Izquierdo-Derecho-Raíz)

Recorre el subárbol izquierdo en postorden
Recorre el subárbol derecho en postorden
Visita la raíz
Ejemplo: D, E, B, F, C, A
Útil para liberar memoria o evaluar expresiones postfijas.

Caso Práctico

El caso práctico desarrollado consiste en la implementación de un árbol binario utilizando listas anidadas, con el objetivo de aplicar de manera concreta los conceptos fundamentales vistos en clase, tales como estructuras dinámicas, validaciones lógicas y recursividad. Para ello, se diseñaron e implementaron una serie de funciones que permiten interactuar con el árbol de forma ordenada, controlada y fácilmente comprensible.

Entre las funcionalidades principales que se ofrecen, se encuentra la posibilidad de definir de manera explícita la raíz del árbol, asegurando que sea siempre el punto inicial a partir del cual se construye la estructura jerárquica. A partir de esa raíz, el sistema permite agregar nodos hijos, brindando al usuario la opción de especificar si desea insertar un valor en la rama izquierda o derecha. Dicha elección es validada cuidadosamente, incluyendo controles que impiden la sobrescritura de nodos ya existentes o la inserción en ubicaciones no permitidas, mostrando mensajes informativos ante cualquier acción inválida.

Se desarrolló un pequeño menú en consola que da acceso a las funcionalidades implementadas, lo que permite una navegación sencilla para el usuario, reforzando así buenas prácticas en la interfaz por línea de comandos. Este menú también contempla validaciones dinámicas que orientan al usuario, evitando errores comunes durante la manipulación del árbol.

En cuanto a la visualización del árbol, se ofrecen tres métodos distintos que permiten evaluar la estructura de forma progresivamente más amigable:

1. **Impresión directa en consola**, sin aditamentos ni librerías externas, que muestra la estructura de manera básica.

1. **Representación en consola utilizando caracteres ASCII**, lo cual facilita una lectura más clara de las relaciones entre nodos, ideal para entornos sin entorno gráfico.

2. **Exportación a imagen**, generando un archivo visual del árbol binario mediante el uso de librerías estándar de Python. Para ello, se desarrollaron funciones auxiliares que traducen la estructura interna del árbol a un formato compatible con las herramientas de graficación elegidas.

Asimismo, se contempló una **demostración práctica en vivo**, que permite verificar en tiempo real el funcionamiento de todas las características implementadas, consolidando así el entendimiento tanto del diseño lógico como de la ejecución funcional del sistema.

Respecto a la organización del proyecto, se adoptó una estructura de repositorio clara y modular. Se incluyó un archivo **README.md** como documento principal de referencia, explicando el uso del programa y su estructura. El directorio **Documentación** agrupa los materiales complementarios, como los criterios de evaluación, el enunciado original del trabajo práctico, material de clases de referencia, y este mismo documento. En la carpeta **src** se encuentra el archivo principal en Python (.py), correspondiente a la implementación del árbol binario. Por último, se incluye una carpeta denominada **gráficos**, destinada a almacenar las salidas visuales generadas por la ejecución del programa.

En conclusión, la elección de este diseño busca facilitar tanto la comprensión del código como la reutilización y evaluación del trabajo por parte de los docentes, cumpliendo con los criterios propuestos y potenciando la claridad del entregable final.

Metodología Utilizada

Durante la etapa inicial, el equipo llevó a cabo una breve instancia de debate para definir el enfoque del trabajo práctico. Tras considerar las distintas opciones propuestas, se optó por desarrollar una implementación de árboles binarios, tal como se detalló en secciones anteriores. Esta decisión se basó en la riqueza conceptual del tema, su aplicabilidad práctica y su potencial para integrar contenidos teóricos vistos a lo largo de la cursada.

Como parte de la **investigación previa**, se retomaron y analizaron detenidamente los materiales disponibles en la plataforma de aprendizaje de la materia, especialmente aquellos relacionados con estructuras dinámicas y recursividad. Se complementa este estudio con la lectura de artículos técnicos disponibles en línea, tutoriales especializados sobre árboles en Python y documentación oficial de las librerías que se evaluaron para tareas de visualización. Las fuentes consultadas se encuentran debidamente listadas en la sección de bibliografía al final del documento, y fueron fundamentales para orientar tanto el diseño del código como la exploración de alternativas para su implementación.

El **proceso de diseño y desarrollo** fue llevado adelante en varias etapas. En primer lugar, se creó un nuevo repositorio en GitHub destinado exclusivamente a este proyecto, estableciendo una estructura organizada de carpetas y subdirectorios que facilitara el versionado y la colaboración. A partir de allí, se desarrolló una **versión funcional mínima** del sistema, centrada en la creación del árbol binario y la implementación básica de una función de recorrido, lo que permitió validar el enfoque inicial. Luego de comprobar el funcionamiento correcto del prototipo, se procedió a **esquematizar en pseudocódigo** el resto de las funcionalidades a incorporar, incluyendo los recorridos inorden, preorden y postorden, así como la lógica de validación de ingreso de datos y el menú interactivo.

Posteriormente, se avanzó con la implementación completa del menú, el manejo de entradas por teclado, la validación de opciones y la incorporación de los métodos de búsqueda planificados. Como etapa final, se añadieron funcionalidades para la **visualización del árbol**, desarrollando distintas versiones: impresión directa en consola, representación en caracteres ASCII, y exportación del árbol como imagen, utilizando librerías externas.

Para llevar adelante este desarrollo, se emplearon diversas **herramientas y recursos vistos a lo largo de la materia**. El entorno de desarrollo utilizado fue **Visual Studio Code (VSCode)**. El control de versiones se realizó mediante **Git**, con alojamiento en **GitHub**, donde se trató de ir subiendo commits y trabajo con ramas, permitiendo trabajar de forma ordenada y mantener registros de

cada iteración. En cuanto a librerías, se utilizó **anytree** para la representación lógica del árbol y **pydot** para la generación de gráficos exportables. Todas las dependencias utilizadas se encuentran declaradas para su fácil instalación y reproducibilidad del proyecto.

Este proceso permitió no solo cumplir con los objetivos planteados inicialmente, sino también fortalecer habilidades de trabajo colaborativo, planificación modular del código y aplicación de herramientas estándar del entorno profesional.

Trabajo en equipo

Integrante	**Tareas**
Luis Cisneros	Marco teórico, desarrollo operaciones árbol en python, menú y demostración funcionalidad.
Guillermo Campoy	Documentación entregable, organización de repositorio, prototipo inicial, funcionalidades de graficado y presentación.

Resultados Obtenidos

A lo largo del desarrollo del caso práctico, se logró cumplir satisfactoriamente con los objetivos propuestos desde el inicio. Se llevó a cabo una implementación funcional de árboles binarios utilizando listas anidadas, abarcando tanto la construcción de la estructura como la incorporación de operaciones clave como inserción, recorridos y visualización. Este ejercicio permitió no solo aplicar los contenidos teóricos de la materia, sino también fortalecer habilidades prácticas de diseño, validación e iteración sobre código funcional.

Respecto a las **operaciones implementadas sobre los árboles**, los tiempos de codificación y prueba se mantuvieron dentro de lo esperado, lo cual fue posible gracias a una planificación adecuada y una división clara de tareas dentro del equipo. La implementación inicial incluyó funciones para la construcción de nodos, inserción guiada (con validaciones por menú), y recorridos clásicos (inorden, preorden y postorden), lo que constituyó una base sólida para avanzar hacia las funciones de visualización.

En cuanto a los **casos de prueba**, no se presentaron inconvenientes significativos. Las funcionalidades fueron validadas con entradas variadas y se observaron salidas acordes a lo esperado, lo que reforzó la solidez de las decisiones tomadas en la etapa de diseño. Se destaca también el uso de pruebas manuales durante el desarrollo, que si bien no fueron formalizadas con frameworks automatizados, resultaron eficaces para el alcance de este trabajo.

No obstante, el equipo enfrentó ciertos **desafíos relacionados con la reutilización de lógica compartida**, en particular al momento de consolidar cambios provenientes de distintas ramas del repositorio. En algunos casos, se produjeron conflictos de merge o sobreescrituras no intencionadas, lo que llevó a revisar la integración de cada componente. Este tipo de dificultades, si bien comunes en entornos colaborativos, sirvieron como una instancia de aprendizaje valiosa, especialmente en lo referido al uso de control de versiones y a la importancia de mantener una comunicación fluida entre los integrantes del equipo.

A modo cierre, el desarrollo de este caso práctico permitió alcanzar los objetivos planteados, explorar en profundidad un tema central de la materia, y además, incorporar aprendizajes significativos sobre trabajo colaborativo, gestión de código y resolución de conflictos técnicos, lo que enriquece no solo la comprensión de la temática sino también la experiencia general como futuros desarrolladores.

Conclusiones

La implementación de árboles mediante listas en Python es una estrategia valiosa, especialmente en contextos educativos y prototipos iniciales. Esta aproximación simplifica la comprensión de la estructura y las operaciones básicas de los árboles binarios, permitiéndonos centrarnos en la lógica fundamental sin la complejidad añadida de la programación orientada a objetos. Aunque este método no es óptimo para aplicaciones a gran escala debido a limitaciones en eficiencia y escalabilidad, resulta ideal para cursos introductorios de estructuras de datos o situaciones que demandan soluciones rápidas y sencillas. En resumen, el uso de listas para representar árboles en Python es una herramienta didáctica poderosa que facilita el aprendizaje y la experimentación con conceptos clave en programación.

Como mejoras a futuro sobre el material relevado podemos mencionar, desde una aproximación con objetos a operaciones avanzadas. Sobre las operaciones, pueden ser mecanismos de inserción parametrizada, balanceo, conteo de nodos, búsqueda por niveles, altura o profundidad.

Bibliografía

- UTN – Universidad Tecnológica Nacional. (2025). *Plataforma Moodle – Unidad de datos complejos*. Recuperado el 15 de junio de 2025, de https://tup.sied.utn.edu.ar/course/view.php?id=12&section=66
- UTN – Universidad Tecnológica Nacional. (2025). *Videos introductorios sobre estructuras complejas*. Recuperado el 15 de junio de 2025, de https://tup.sied.utn.edu.ar/course/view.php?id=12&section=67
- Diego Coder. (2021). *Grafos y árboles en Python*. Medium. Recuperado el 12 de junio de 2025, de https://medium.com/@diego.coder/grafos-y-%C3%A1rboles-en-python-2d165dfc8bbd
- Runestone Academy. (s.f.). *Implementación de Árbol Binario de Búsqueda en Python*. En *Python Data Structures* (capítulo Trees). Recuperado el 13 de junio de 2025, de https://runestone.academy/ns/books/published/pythoned/Trees/ImplementacionArbolBusqueda.html
- Open Source Contributors. (s.f.). *Anytree Documentation (v2.8.0)*. Read the Docs. Recuperado el 13 de junio de 2025, de https://anytree.readthedocs.io/en/latest/