TRABAJO INTEGRADOR DE PROGRAMACIÓN: ÁRBOLES CON LISTAS

INTEGRANTES:

Cornejo, Diego Gaston

Dantur, Daniel Fernando Jesus

Comisión: M2025-12

Materia: Programación

Docente: Ariel Enferrel

Docente tutor: Luciano Chiroli

Fecha de entrega: 09/06/2025

Introducción

En el presente trabajo integrador correspondiente a la materia Programación, se desarrolla una aplicación en lenguaje Python que implementa la estructura de datos conocida como árbol, utilizando listas anidadas como mecanismo de representación.

El objetivo principal de este proyecto es comprender en profundidad el funcionamiento de los árboles como estructuras jerárquicas, explorar sus métodos de recorrido, búsqueda, inserción y visualización, y aplicar estos conocimientos en la resolución de problemas prácticos.

La estructura de árbol es ampliamente utilizada en diversas áreas de la informática, como bases de datos, compiladores, estructuras de archivos, inteligencia artificial y algoritmos de búsqueda. Por tal motivo, su dominio resulta esencial para cualquier programador en formación.

El desarrollo se realizó íntegramente en Python, haciendo énfasis en una implementación sencilla, didáctica y comprensible para su posterior análisis y reutilización con fines educativos. A lo largo del informe se detallan los conceptos teóricos, las decisiones de diseño, el funcionamiento del programa y las conclusiones obtenidas tras su desarrollo.

Marco Teórico

Los árboles son estructuras de datos jerárquicas fundamentales en ciencias de la computación. Se utilizan ampliamente en algoritmos de búsqueda, estructuras de directorios, análisis sintáctico, almacenamiento jerárquico de información y sistemas de decisión. Su naturaleza recursiva permite representar relaciones de dependencia o subordinación entre elementos de manera eficiente.

1. Definición de árbol

Un árbol es una estructura de datos no lineal compuesta por nodos conectados entre sí de manera jerárquica. Está formado por un nodo raíz, del cual se desprenden ramas hacia otros nodos, conocidos como hijos. Cada uno de estos nodos, a su vez, puede tener sus propios descendientes. A diferencia de otras estructuras como listas o arreglos, los árboles no presentan una organización secuencial, sino que se organizan en niveles.

Un caso particular de árbol es el árbol binario, en el que cada nodo tiene como máximo dos hijos, tradicionalmente denominados hijo izquierdo e hijo derecho. Esta restricción facilita la implementación de algoritmos de recorrido, búsqueda y modificación de estructuras jerárquicas.

2. Componentes y propiedades

Los elementos principales de un árbol son:

- **Raíz**: es el nodo inicial del cual parten todos los demás.
- **internos:Nodos** nodos que tienen al menos un hijo.
- **Hojas**: nodos que no tienen hijos.
- **Altura del árbol**: cantidad máxima de niveles desde la raíz hasta una hoja.
- **Nivel de un nodo**: distancia del nodo respecto a la raíz.

- **Subárboles**: cualquier nodo con sus descendientes forma un subárbol.

Estas propiedades permiten definir métricas, analizar rendimiento algorítmico y organizar la lógica de recorrido de manera sistemática**.**

3. Representación de árboles en programación

En la mayoría de los lenguajes de programación, los árboles se representan mediante estructuras de nodos enlazados utilizando clases y punteros (en lenguajes como Java o C++). Sin embargo, algunos lenguajes como Python ofrecen alternativas más flexibles. Una de ellas es el uso de listas anidadas, donde cada nodo es una lista que contiene su valor y referencias a sus subárboles.

Este enfoque, si bien más limitado en términos de escalabilidad y organización formal, resulta sumamente útil en contextos educativos. Permite abstraer la implementación orientada a objetos y centrarse en la lógica estructural del árbol, facilitando la comprensión de los principios fundamentales de la estructura.

4. Operaciones fundamentales sobre árboles

Los árboles soportan diversas operaciones que permiten su manipulación:

- Inserción: consiste en añadir un nuevo nodo como descendiente de un nodo existente.

- Búsqueda: proceso de localización de un nodo que contiene un valor específico.

- Recorridos: métodos sistemáticos para visitar todos los nodos del árbol. Los principales tipos son:

    - *Preorden*: se visita primero el nodo actual, luego el subárbol izquierdo y por último el derecho.

    - *Inorden*: se visita el subárbol izquierdo, luego el nodo actual, y finalmente el derecho.

- *Postorden*: se visitan primero ambos subárboles y luego el nodo actual.

- Visualización: proceso de representar la estructura del árbol de forma gráfica o textual para facilitar su análisis.

Estas operaciones son esenciales para la manipulación eficiente de información estructurada y para el desarrollo de algoritmos de clasificación, evaluación de expresiones, y toma de decisiones automatizada.

5. Enfoques de implementación

Existen diversos enfoques para la implementación de árboles. La elección depende del objetivo del sistema y del nivel de complejidad requerido.

- Implementación clásica con clases: se define una clase Nodo con atributos para el valor y los enlaces a los nodos hijos. Este enfoque ofrece mayor control y escalabilidad.

- Implementación con listas: cada nodo se representa como una lista que contiene su valor y referencias a sus descendientes. Esta forma es menos formal pero resulta eficaz en entornos de aprendizaje.

- Implementación con diccionarios: útil en árboles de n-arios o estructuras menos rígidas, donde el número de hijos no es fijo.

Cada enfoque tiene sus ventajas y limitaciones. El uso de listas anidadas, como el empleado en el trabajo práctico, destaca por su simplicidad conceptual, lo que lo convierte en una excelente herramienta didáctica para introducir los conceptos de jerarquía, recursividad y estructuración de datos.

6. Aplicaciones de los árboles

Los árboles tienen múltiples aplicaciones prácticas en programación y ciencias de la computación, entre ellas:

- Sistemas de archivos: organización jerárquica de carpetas y archivos.

- Estructuras de búsqueda: como los árboles binarios de búsqueda (BST), árboles AVL, y árboles B +.

- Compiladores: análisis de sintaxis mediante árboles de derivación.

- Bases de datos jerárquicas: organización lógica de registros relacionados.

- Sistemas expertos e inteligencia artificial: árboles de decisión y árboles de juegos.

Estas aplicaciones reflejan la importancia de dominar esta estructura, no solo desde un punto de vista teórico, sino también práctico y funcional.

Caso Práctico:

Árbol Binario con Listas

Paso 1: Definición de la estructura base

Se definió un modelo lógico para representar un árbol binario con listas anidadas. Cada nodo incluye un valor y dos posibles hijos (izquierdo y derecho). Esta convención sirvió de base para las operaciones de inserción, búsqueda y recorrido.

Construcción del nodo raíz

Se estableció un mecanismo para inicializar el árbol binario desde una raíz, punto central a partir del cual se organiza la jerarquía completa.

Incorporación de nodos secundarios

Se desarrollaron funciones para agregar nuevos nodos como hijos de un nodo existente, manteniendo la lógica binaria. Se incluyó una búsqueda previa del nodo padre.

Búsqueda de elementos en la estructura

Se programaron recorridos sistemáticos que permiten ubicar un valor dentro del árbol, garantizando que las operaciones se realicen sobre nodos válidos.

Implementación de recorridos jerárquicos

Se aplicaron tres tipos clásicos de recorrido:

- Preorden
- Inorden
- Postorden

  Estos métodos permiten visualizar la disposición de los datos según distintas estrategias.

Visualización estructural del árbol

Se desarrolló una visualización teórica de la estructura para comprender la profundidad, ramificación y equilibrio del árbol.

Organización del flujo de trabajo

Se estableció un flujo operativo: creación del árbol, inserción de nodos, búsqueda, recorridos y visualización. Esto permitió validar progresivamente los contenidos teóricos.

Validación teórica del funcionamiento

Se analizaron escenarios como árboles balanceados, desequilibrados, múltiples inserciones y búsquedas sin resultado, evaluando el comportamiento de la estructura.

Aplicación práctica - Gestor de tareas con árbol general

Con el objetivo de aplicar los conceptos teóricos adquiridos, se desarrolló un sistema de gestión de tareas jerárquicas mediante un árbol general, representado en Python como nodos con múltiples hijos.

Estructura general:

- Se utilizó una estructura tipo árbol donde cada nodo representa una tarea.
- La raíz es un nodo llamado "Tareas".
- Cada tarea puede tener subtareas, formando así una jerarquía anidada.

Funcionalidades implementadas:

Inicialización del árbol

Se crea el nodo raíz "Tareas" y se establece una referencia activa a la tarea seleccionada actual.

Agregado de tareas

Se permite al usuario crear nuevas tareas y subtareas. Cada nodo incluye:

- Título
- Descripción (opcional)
- Estado (completado o no)

Visualización jerárquica

El programa muestra las tareas en consola con su jerarquía, subtareas e información adicional, permitiendo visualizar claramente la estructura.

Navegación dentro del árbol

El usuario puede moverse dentro de la jerarquía para seleccionar tareas, volver atrás, ir a la raíz o explorar subtareas.

Gestión de tareas

Se habilitan acciones sobre las tareas:

- Marcar como completada
- Agregar descripción
- Eliminar tarea
- Crear subtareas

Control de flujo e interacción

Todo el sistema funciona en un ciclo interactivo que limpia pantalla, muestra tareas y espera la acción del usuario, ofreciendo una experiencia fluida y ordenada.

Resultados obtenidos:

- Se logró una implementación funcional de un árbol general para organizar tareas de manera anidada.
- La aplicación permite navegar, modificar y visualizar tareas en múltiples niveles.
- Se fortalecieron los conocimientos teóricos mediante una experiencia práctica concreta.

Conclusiones:

- El árbol binario y el árbol general representan dos modelos jerárquicos útiles para múltiples propósitos.

- La combinación de teoría (árbol binario) y práctica (árbol general aplicado) permitió entender las ventajas y desafíos de cada enfoque.

- El proyecto integrador demostró la aplicabilidad real de las estructuras de datos y la programación modular en Python.

Metodología Utilizada

La metodología adoptada para el desarrollo del presente trabajo se basó en una secuencia sistemática de análisis, diseño e implementación teórica de una estructura de árbol binario, haciendo uso del lenguaje Python como soporte técnico. Se prioriza una estrategia de programación modular, que permitiera organizar las funciones en unidades conceptualmente independientes, facilitando la comprensión y el mantenimiento del sistema.

La elección del enfoque mediante listas anidadas respondió a criterios pedagógicos, con el propósito de facilitar la internalización de conceptos fundamentales como la recursividad, la jerarquía estructural y la relación entre nodos. Este modelo permitió una construcción directa de la estructura, sin la necesidad de recurrir a clases ni abstracciones propias de la programación orientada a objetos.

Durante el proceso, se promovió la validación conceptual de cada operación implementada, mediante la reflexión sobre su impacto en la estructura del árbol, su coherencia lógica y su utilidad dentro del conjunto del sistema. Cada módulo funcional fue evaluado de manera individual y luego integrado al flujo general, garantizando que las dependencias entre componentes fuesen claras, predecibles y correctas desde el punto de vista teórico.

La metodología incluyó también la elaboración de representaciones jerárquicas de los árboles resultantes, a modo de recurso visual que apoyara el análisis estructural y favorece el desarrollo de la intuición en torno a la forma y profundidad de las ramas generadas. Este recurso fue clave para vincular la teoría con la comprensión visual de las estructuras.

Finalmente, el trabajo fue desarrollado bajo un enfoque iterativo, lo que permitió la revisión constante de decisiones, el reajuste de criterios de inserción y la mejora progresiva de la representación interna del árbol, en función de los objetivos de claridad y didáctica que guiaron todo el proyecto.

Resultados Obtenidos

Como resultado del desarrollo teórico y funcional del trabajo, se logró una implementación íntegra de una estructura de árbol binario utilizando listas como representación interna. Esta implementación permitió abordar de manera efectiva las principales operaciones vinculadas a este tipo de estructura: creación de nodos, inserción de elementos, búsqueda dentro del árbol, recorrido en distintas secuencias y visualización de la jerarquía.

Desde el punto de vista educativo, la propuesta resultó exitosa al facilitar la comprensión del comportamiento recursivo del árbol y la forma en que los nodos se relacionan entre sí. La ausencia de programación orientada a objetos no representó un obstáculo, sino que por el contrario, permitió centrar la atención en los principios lógicos y estructurales fundamentales.

El sistema desarrollado demostró ser funcional para árboles de tamaño medio, manteniendo su consistencia y estabilidad en diversos escenarios de inserción. Las distintas estrategias de recorrido permitieron verificar el orden lógico de los datos y su distribución en la jerarquía.

Se comprobó que la representación mediante listas, si bien más limitada desde una perspectiva profesional o industrial, resulta altamente eficaz en contextos de formación básica, donde lo fundamental es adquirir una comprensión clara de los principios que rigen las estructuras de datos no lineales.

Conclusiones Generales

La experiencia obtenida a partir del presente trabajo integrador permitió comprobar que es posible implementar estructuras complejas como los árboles binarios utilizando representaciones simples y accesibles como las listas en Python. Esta estrategia, aunque no la más potente ni la más escalable, se presenta como una herramienta valiosa en los procesos de enseñanza y aprendizaje de programación estructurada.

El trabajo contribuyó al desarrollo de habilidades clave en la manipulación de estructuras jerárquicas, en el diseño modular de sistemas, y en el razonamiento lógico para la resolución de problemas mediante herramientas computacionales.

A continuación, se resumen las principales ventajas, desventajas, posibles mejoras identificadas y las complicaciones que pudieran surgir durante el desarrollo:

Ventajas del enfoque utilizado

- Claridad estructural: el uso de listas anidadas permite visualizar con facilidad la jerarquía entre nodos.

- Simplicidad conceptual: se evita la complejidad inherente a la programación orientada a objetos, facilitando el aprendizaje.

- Accesibilidad: no requiere conocimientos avanzados de estructuras ni uso de bibliotecas externas.

- Aplicación inmediata: permite poner en práctica rápidamente los conceptos teóricos trabajados en clase.

Desventajas del enfoque

- Limitada escalabilidad: la representación con listas se vuelve poco eficiente a medida que crece la profundidad del árbol.

- Falta de encapsulamiento: no hay protección de datos ni separación de responsabilidades como en un diseño orientado a objetos.

- Dificultad de mantenimiento: la manipulación directa de índices dentro de listas puede generar errores difíciles de depurar.

- Carencia de funcionalidades avanzadas: no permite fácilmente la implementación de árboles balanceados, árboles de búsqueda o mecanismos automáticos de reorganización.

Posibles mejoras

- Migración a una implementación orientada a objetos: utilizando clases para representar nodos y el árbol como estructura global.

- Incorporación de validaciones y manejo de errores: para garantizar que las inserciones y búsquedas se realicen correctamente.

- Visualización gráfica del árbol: mediante herramientas externas o bibliotecas de visualización para reforzar la comprensión visual.

- Extensión a árboles de búsqueda o balanceados: para explorar aplicaciones más avanzadas de esta estructura y su uso en algoritmos eficientes.

Complicaciones que pudieran surgir

Durante el desarrollo del trabajo, se identificaron posibles dificultades inherentes tanto al enfoque elegido como al proceso de implementación. Entre ellas, se destacan:

- **Gestión de la estructura anidada:** La representación del árbol mediante listas exige una atención constante a los niveles de anidamiento, lo cual puede dificultar la lectura, depuración y modificación del árbol, especialmente en estructuras profundas.

- **Localización de nodos internos:** La ausencia de referencias directas a objetos o punteros obliga a realizar búsquedas recursivas para localizar nodos, lo que puede complejizar operaciones como la inserción o verificación de existencia.

- **Limitaciones del entorno textual:** La interacción a través de consola restringe la visualización intuitiva del árbol, lo que puede dificultar la comprensión de su forma jerárquica para usuarios sin experiencia.

- **Errores por manipulación directa de índices:** Dado que los nodos se acceden mediante índices fijos, una alteración accidental en la estructura de las listas puede romper el árbol o generar inconsistencias difíciles de detectar.

Estas complicaciones, aunque propias de un enfoque minimalista, constituyen a su vez oportunidades de aprendizaje valiosas para la identificación y solución de problemas estructurales en programación.

Bibliografía

Cursa. (s. f.). *Estructuras de datos en Python: árboles*. Recuperado de https://cursa.app/es/pagina/estructuras-de-datos-en-python-arboles ocw.uc3m.es+2es.scribd.com+2runestone.academy+2cursa.app

Informática UNAHUR. (2019). *Estructuras de datos – Árboles binarios de búsqueda en Python* [Video]. YouTube. Recuperado de https://www.youtube.com/watch?v=ljJz5fqhe2I youtube.com

Python Software Foundation. (2025). *Estructuras de datos — Documentación de Python* (sección 5.1.4: listas anidadas). Recuperado de https://docs.python.org/es/3.13/tutorial/datastructures.html datacamp.com+9docs.python.org+9labex.io+9

Scribd. (s. f.). *Lab 07 – Árboles*. Recuperado de https://es.scribd.com/document/595584156/Lab-07-Arboles-1 es.scribd.com

Wikipedia. (2025, abril). *Árbol binario* [Artículo]. Recuperado de https://es.wikipedia.org/wiki/Árbol_binario runestone.academy+6es.wikipedia.org+6cursa.app+6

Wikipedia. (2024). *Recorrido de árboles* [Artículo]. Recuperado de https://es.wikipedia.org/wiki/Recorrido_de_árboles