

Árbol de Directorios

Implementación de Árboles en Python

Alumnos:

BRIÑÓCCOLI, Adrian Nahuel – email: adrianbrinoccoli@gmail.com
BUJALDON DUARTE, Octavio Carlos – email: octa.buja@gmail.com

Materia:

Programación 1

Profesor:

Cinthia Rigoni

Profesor Tutor:

Oscar Londero

09 de Junio de 2025

Introducción:

El siguiente trabajo integrador trata sobre el desarrollo de una estructura de datos de árbol para la representación de sistemas de archivos jerárquicos. La elección de este tema se fundamenta en su relevancia directa y observable en la organización de la información digital cotidiana, dado que en los distintos tipos de sistemas operativos se gestionan archivos y directorios siguiendo un modelo de árbol.

La importancia de este tema para la programación reside en que las estructuras de árbol son fundamentales para la resolución de diversos problemas computacionales más allá de la gestión de archivos. Permiten organizar datos de manera jerárquica, facilitando operaciones eficientes de búsquedas, inserción y eliminación. Son aplicables en ámbitos como bases de datos, compiladores, inteligencia artificial y representación de expresiones matemática, lo que demuestra su versatilidad y eficiencia como herramienta algorítmica.

Con la elaboración de este trabajo integrador, se propone alcanzar los siguientes objetivos: implementar una clase que represente los nodos de un árbol, diferenciar entre nodos de tipo “carpeta” y “archivo”, establecer la capacidad de agregar elementos hijos a los nodos tipo “carpeta” y, finalmente, desarrollar una función recursiva que permita visualizar la estructura jerárquica del árbol completo. De esta manera, se busca demostrar la comprensión y aplicación práctica de los principios de las estructuras de datos en un contexto familiar y relevante.

Marco Teórico: Estructuras de Datos de Árbol:

En la ciencia de la computación, las estructuras de datos son clave para organizar y almacenar información de forma eficiente para su posterior procesamiento. Dentro de este campo, los arboles representan una de las estructuras no lineales y dinámicas más importantes y versátiles. A diferencia de las estructuras lineales o los arreglos, los arboles organizan la información de forma jerárquica, reflejando relaciones de uno a muchos.

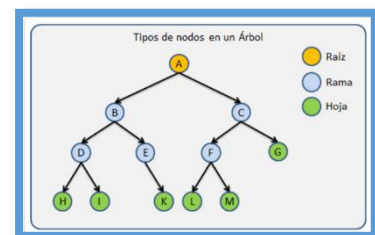
¿Qué son los Árboles?

Los árboles son una estructura de datos más fundamentales y poderosos en la informática. Son utilizados en una amplia variedad de aplicaciones del mundo real, como la organización de archivos en sistemas operativos, la gestión de bases de datos y en algoritmos de optimización. Un árbol puede definirse formalmente como una colección de elementos, denominados “Nodos” conectados por “aristas o ramas”, un árbol no vacío posee un único nodo raíz, el cual no tiene padre, a partir de la raíz, se desprenden otros nodos, estableciendo una jerarquía donde cada nodo (excepto la raíz) tiene exactamente un nodo padre y puede tener cero o más nodos hijos.



Clasificación de nodos según su ubicación en el árbol.

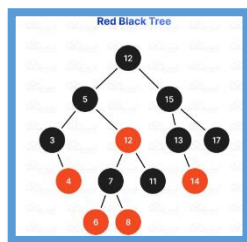
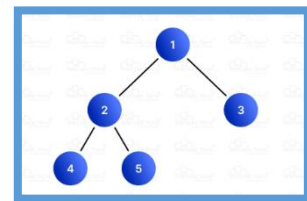
- **Nodo Raíz:** Es el primer nodo del árbol, desde el cual cuelgan todos sus descendientes. Solo un nodo puede ser la raíz.
- **Nodo Hoja:** es un nodo que no tiene hijos, se encuentran en el nivel más bajo del árbol.
- **Nodo / Rama:** Cada elemento individual en un árbol se lo denomina nodo, cada “nodo” tiene un valor o clave que lo identifica de manera única dentro del árbol.



Tipos de Árboles:

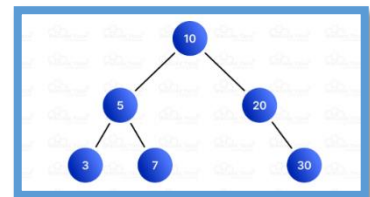
Existen diversas clasificaciones de árboles, cada una con características y aplicaciones específicas:

- Árboles Generales: Son aquellos árboles donde un nodo puede tener un número limitado de hijos.
- Árboles Binarios: Cada nodo puede tener un máximo de dos hijos (izquierdo y derecho). Son de especial interés y muy utilizados en informática, dentro de los árboles binarios, se distinguen:
- Árbol Binario Completo: Cada nivel, excepto posiblemente el último, está completamente lleno, y todos los nodos del último nivel están a la izquierda posible



Árbol Binario Equilibrado: Árboles donde la diferencia de alturas entre los subárboles izquierdo y derecho de cualquier nodo es como máximo uno. Esto garantiza la eficiencia en la operaciones, por ejemplo incluyen los árboles AVL y Rojo-Negro.

- Árboles de Búsqueda Binaria (BST): Son los árboles binarios donde para cada nodo, todos los valores en su subárbol izquierdo son menores que el valor del nodo, y todos los valores en su subárbol derecho son mayores, esto permite búsquedas eficientes.



Operaciones fundamentales en Árboles:

Las operaciones básicas realizadas en árboles son:

- *Inserción de Nodos*: Añadir un nuevo elemento al árbol, manteniendo sus propiedades. Por ejemplo en un BST
- *Eliminación de Nodos*: Remover un elemento del árbol y ajustar la estructura para mantener la coherencia.
- *Búsquedas de Elementos*: Localizar un nodo específico dentro del árbol.
- *Recorridos (Traversals)*: Visitar cada nodo del árbol una única vez de manera sistemática. Los recorridos más comunes son:
 - *Preorden (Pre-orden)*: Visita la raíz, luego el subárbol izquierdo y finalmente el subárbol derecho.
 - *Inorden (In-orden)*: Visita el subárbol izquierdo, luego la raíz y finalmente el subárbol derecho (útil para obtener elementos ordenados en BST).
 - *Postorden (Post-order)*: Visita el subárbol izquierdo, luego el subárbol derecho y finalmente la raíz.

Aplicaciones de las Estructuras de Datos de Árbol:

La naturaleza jerárquica y eficiente de los arboles los hace indispensables en numerosas aplicaciones informáticas.

- Sistemas de Archivos y Directorios: Los árboles son la base de como los sistemas operativos organizan archivos y carpetas.
- Bases de Datos: Optimizan la recuperación de información y el almacenamiento de datos indexados.
- Compiladores: Representan la estructura sintáctica de programas, por ejemplo árboles de análisis sintáctico.
- Árboles de Decisión: son utilizados en la inteligencia artificial y machine learning para decisiones y sus posibles resultados.
- Árboles genealógicos y Organigrama: Modelado de relaciones jerárquicas en diversos contextos.
- Redes de Computadoras: son los enrutamientos y topología de red.

Caso Práctico:

El caso práctico desarrollado consiste en la simulación de un árbol de directorios mediante la programación orientada a objetos en Python. El objetivo principal fue representar de manera estructurada la jerarquía de carpetas y archivos, tal como ocurre en un sistema de archivos real, por ejemplo en Windows o Linux.

Para lograr esto, se creó una clase Nodo, que representa tanto carpetas como archivos. Cada nodo tiene un nombre, un tipo (carpeta o archivo) y, en el caso de las carpetas, una lista de hijos. Esta estructura permite construir árboles n-arios, donde cada carpeta puede contener múltiples elementos, sin estar limitada a dos hijos como en los árboles binarios.

Se implementaron métodos para agregar nodos y para visualizar la estructura completa del directorio de forma jerárquica, utilizando recursividad para recorrer los distintos niveles del árbol. El código se desarrolló íntegramente en Python, haciendo uso de listas, condicionales, clases y funciones recursivas.

```
class Nodo:
    def __init__(self, nombre, tipo="carpeta"):
        self.nombre = nombre          # Nombre del archivo o carpeta
        self.tipo = tipo              # Puede ser "carpeta" o "archivo"
        self.hijos = []              # Lista de hijos (otros nodos)

    def agregar_hijo(self, hijo):
        self.hijos.append(hijo)      # Agrega un hijo a la lista (si es carpeta)

    def mostrar_estructura(self, nivel=0):
        sangria = " " * nivel
        print(sangria + (self.nombre + "/" if self.tipo == "carpeta" else self.nombre))

        # Si es carpeta, imprime sus hijos de forma recursiva
        if self.tipo == "carpeta":
            for hijo in self.hijos:
                hijo.mostrar_estructura(nivel + 1)

# Crear el nodo raíz
root = Nodo("root")

# Carpeta: documentos
documentos = Nodo("documentos")
documentos.agregar_hijo(Nodo("tp_final.pdf", tipo="archivo"))
documentos.agregar_hijo(Nodo("apuntes.txt", tipo="archivo"))

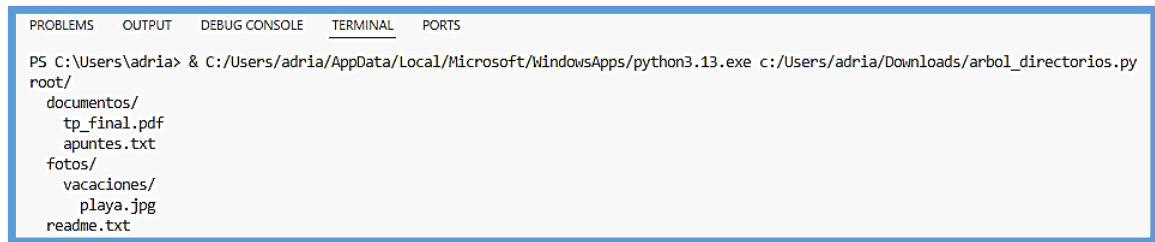
# Carpeta: fotos/vacaciones
fotos = Nodo("fotos")
vacaciones = Nodo("vacaciones")
vacaciones.agregar_hijo(Nodo("playa.jpg", tipo="archivo"))
fotos.agregar_hijo(vacaciones)

# Archivo suelto en raíz
readme = Nodo("readme.txt", tipo="archivo")

# Armar el árbol
root.agregar_hijo(documentos)
root.agregar_hijo(fotos)
root.agregar_hijo(readme)

# Mostrar el árbol
root.mostrar_estructura()
```


Validación del funcionamiento



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\adria> & C:/Users/adria/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/adria/Downloads/arbol_directorios.py
root/
  documentos/
    tp_final.pdf
    apuntes.txt
  fotos/
    vacaciones/
      playa.jpg
    readme.txt
```

Metodología Utilizada:

Para llevar adelante este trabajo, primero se realizó una investigación sobre el concepto de árboles en estructuras de datos y su aplicación en Python. Además se contaba con conocimientos de programación orientada en lenguaje JAVA, lo cual se utilizó para realizar el caso práctico de árbol de directorios, ya que consideramos más apropiado realizar el trabajo con el enfoque a objetos aunque también se puede realizar con listas anidadas. Una vez comprendido el funcionamiento de los árboles n-arios y como representarlos en Python, se procedió al diseño de la clase Nodo, que cumple la función de representar los elementos del árbol (carpetas o archivos). Luego, se definió una estructura de ejemplos que simula un árbol de directorios real, incluyendo carpetas anidadas y archivos.

El código fue construido de forma incremental, probando en cada paso las funciones implementadas. Se utilizaron técnicas de recursión para imprimir la estructura en pantalla.

En cuanto al trabajo colaborativo, el grupo se organizó dividiendo las tareas de forma equitativa, la elección del caso, el desarrollo en Python y la prueba del código fue hecho en conjunto mientras que la parte teórica o explicativa fue dividida en partes iguales, la coordinación general se llevó a cabo mediante reuniones virtuales (Zoom) y mensajería, fomentando así la activa participación de ambos integrantes.

Resultados Obtenidos:

El resultado final fue un programa funcional que permite construir un árbol de directorios personalizado y visualizar su estructura de forma jerárquica. Se logró simular correctamente un sistema de archivos con carpetas que contienen tanto subcarpetas como archivos, replicando el comportamiento de un entorno real.

El programa permite:

- Crear nodos de tipo “carpeta” o “archivo”
- Agregar elementos de forma dinámica.
- Visualizar toda la estructura del árbol usando recursividad.

La estructura generada como ejemplo incluyó una raíz con tres carpetas, archivos dentro de subdirectorios, y un archivo suelto. La salida del programa mostro correctamente la jerarquía completa, lo que permitió verificar que el modelo utilizado es adecuado para representar árboles de directorios.

Este trabajo permitió afianzar conocimientos de programación orientadas a objetos, uso de listas, lógica recursiva y estructura de datos jerárquicas en Python.

Conclusiones:

La realización de este trabajo integrador, que fue desarrollado en la implementación de una estructura de datos de árbol para la representación de sistemas de archivos, nos ha permitido conocer más el ámbito de las estructuras de datos no lineales y la programación orientada a objetos. Comprendimos en detalle la definición y los componentes fundamentales de un árbol, como los nodos, raíz, padres, hijos y hojas, así como la importancia de la recursividad para el desarrollo y la visualización de este tipo de estructura jerárquica.

La utilidad del tema para la programación es fundamental, las estructuras de árbol son esenciales para la organización eficiente de datos jerárquico en numerosos dominios, no solo limitándose solo a sistemas de archivos. Su aplicación se extiende a la creación de índices en bases de datos, la implementación de árboles de búsqueda binaria para optimizar la recuperación de información, la representación de expresiones matemáticas, y como base para algoritmos de inteligencia artificial como los árboles de decisión. Esta versatilidad demuestra la importancia como herramienta algorítmica y de diseño en cualquier proyecto de software que requiera manejar relaciones jerárquicas de datos.

En cuanto a posibles mejoras o extensiones futuras al caso presentado se podría agregar funcionalidades de manipulación para que tenga la capacidad de mover elementos entre directorios o renombrar nodos, la búsqueda avanzada, esto permitiría desarrollar algoritmos de búsqueda más complejos que permitan encontrar archivos por nombre, y una interfaz para el usuario que permita interacción visual con el árbol de directorios.

Bibliografía:

Sitios Web con fecha de acceso:

- <https://conclase.net/c/edd/cap6> (fecha de acceso 04/06/2025)
- <https://www.fing.edu.uy/tecnoinf/mvd/cursos/eda/material/teo/EDA-teorico15.pdf> (fecha de acceso 04/06/2025)
- <https://www.wscubetech.com/resources/dsa/tree-data-structure> (05/06/2025)
- <https://www.apinem.com/arboles-programacion/> (06/06/2025)
- <https://cursa.app/es/pagina/estructuras-de-datos-en-python-arboles> (06/06/2025)
- <https://builtin.com/articles/tree-python> (06/06/2025)
- <https://keepcoding.io/blog/arboles-binarios-en-programacion/> (06/06/2025)

Artículos recomendados por el docente:

- **Datos Avanzados:**
 - Video: Introducción a árboles
 - Propiedades de árboles
 - Formas de recorrer un árbol
 - Árboles de búsqueda binaria
 - **Apunte teórico sobre árboles**

Anexo:

Se adjunta el código de Python:

```
class Nodo:
    def __init__(self, nombre, tipo="carpeta"):
        self.nombre = nombre          # Nombre del archivo o carpeta
        self.tipo = tipo              # Puede ser "carpeta" o "archivo"
        self.hijos = []              # Lista de hijos (otros nodos)

    def agregar_hijo(self, hijo):
        self.hijos.append(hijo)      # Agrega un hijo a la lista (si es
carpeta)

    def mostrar_estructura(self, nivel=0):
        sangria = "  " * nivel
        print(sangria + (self.nombre + "/" if self.tipo == "carpeta" else
self.nombre))

        # Si es carpeta, imprime sus hijos de forma recursiva
        if self.tipo == "carpeta":
            for hijo in self.hijos:
                hijo.mostrar_estructura(nivel + 1)

# Crear el nodo raíz
root = Nodo("root")
# Carpeta: documentos
documentos = Nodo("documentos")
documentos.agregar_hijo(Nodo("tp_final.pdf", tipo="archivo"))
documentos.agregar_hijo(Nodo("apuntes.txt", tipo="archivo"))
# Carpeta: fotos/vacaciones
fotos = Nodo("fotos")
vacaciones = Nodo("vacaciones")
vacaciones.agregar_hijo(Nodo("playa.jpg", tipo="archivo"))
fotos.agregar_hijo(vacaciones)
# Archivo suelto en raíz
readme = Nodo("readme.txt", tipo="archivo")
# Armandos el árbol
root.agregar_hijo(documentos)
root.agregar_hijo(fotos)
root.agregar_hijo(readme)
# Mostrar el árbol
root.mostrar_estructura()
```



Link del Video: https://youtu.be/R_hQoSuN0Oo

Diagrama de Flujo:

