Implementación de árboles en Organización de stock utilizando Python

Alumnos:
Franco Siri
Pedro Antonio Sota Taier (psptaier@gmail.com)

Materia: Programación I

Profesor: Prof. Nicolás Quirós
Tutor: Matias Santiago Torres

Fecha de Entrega: 9 de junio de 2025

1. Introducción

En programación orientada a objetos es común utilizar clases, entendidas como *"Una plantilla para crear objetos definidos por el usuario. Las definiciones de clase normalmente contienen definiciones de métodos que operan una instancia de la clase."*[1]. Las clases nos permiten crear variables que al pertenecer a una clase heredan un conjunto funciones, parámetros y utilidades. Esto va de la mano con el concepto de árbol en programación, que es un tipo de estructura de datos jerárquica que utiliza la recursividad ya sea para recorrer sus elementos, como para crear, eliminar, y las clases es una de las opciones más claras, especialmente cuando el árbol tiene lógica asociada (como métodos para inserción, búsqueda, recorrido, etc.)

2. Marco Teórico

Este trabajo se centra en el uso de una estructura de datos complejas como son la de tipo árbol, también utilizaremos la definición de clases ya que nos va a permitir modelar los nodos del árbol.
Para eso es importante entender la definición de árbol: Un árbol es una estructura de datos que organiza sus elementos en forma jerárquica. Por lo tanto establece una relación padre/hijo entre los elementos en el que cada uno de estos puede tener más de un hijo, pero un solo padre. A cada uno de estos elementos se los denomina *Nodo*, y hay distintas formas de clasificarlas.
El primero es según su ubicación en el árbol, en el que encontramos tres tipos:
Nodo raíz: es el primer nodo, en la parte superior de la jerarquía. Por lo tanto es el único nodo que no tiene "padre", y por árbol sólo hay un nodo de este tipo.
Nodo interno/rama: son aquellos nodos que tienen un padre y al menos un hijo, por lo tanto son nodos intermedios en la jerarquía que siguen propagando la estructura.
Nodo hoja: son los nodos que están al final de la jerarquía y por lo tanto no tienen hijos, por lo que no tienen más ramificaciones.
Además de la relación de padre-hijo de los nodos podemos definir la relación de tipo **hermanos** que refiere a los nodos que comparten el nodo padre y entre sí están en la misma jerarquía ya que tienen la misma distancia al nodo raíz que a los nodos hoja.

Luego podemos definir una serie de propiedades de este tipo de estructuras, como la **profundidad** que es la longitud de camino (número de ramas que hay que transitar para llegar de un nodo a otro) entre el nodo raíz y un nodo específico. En cambio también tenemos el **nivel** que también es la longitud de camino del nodo raíz a un nodo, más uno. *"La diferencia entre*

[1] Python Documentation. (2025) *Glosario.* https://docs.python.org/es/3.13/glossary.html#term-class

nivel y profundidad es que el nivel incluye a la raíz mientras que la profundidad no la tiene en cuenta."[2]
Y tenemos **altura** que es el nivel máximo de un árbol.
Por otro lado tenemos el **grado** que es la cantidad de hijos de un nodo. El **orden** que es la cantidad máxima de hijos que pueden tener cada nodo (ej: un árbol de grado dos es lo que llamamos un árbol binario ya que cada nodo sólo puede tener dos hijos). Y el **peso** que es la cantidad total de nodos que tiene un árbol, nos provee información sobre el tamaño del árbol y de la cantidad de memoria que podría implicar en su ejecución.

Este tipo de estructura compleja y jerárquica implica necesariamente métodos de búsqueda para poder rastrear un nodo en particular. Hay varios métodos para recorrer árboles, por ejemplo **postorden** que comienza por la hoja que se encuentra más a la izquierda, recorre primero los hermanos, luego el padre, luego repite el proceso comenzando por el subárbol derecho y por último la raíz. Este tipo de recorrido es común para realizar alguna acción en los nodos hijos antes de procesar al padre. Otro método es **inorden**, que también comienza por el nodo hoja más a la izquierda de todo, luego al padre, luego el otro nodo hijo, repite esa lógica por todo el subárbol izquierdo hasta llegar al nodo raíz, y luego lo mismo desde el subárbol derecho. Este método es útil para las búsquedas en árboles binarios. Y por último está **preorden** que comienza el recorrido por la raíz y sigue por el hijo izquierdo recorriendolo recursivamente antes de recorrer el subárbol derecho. Este último es el método que utilizamos en nuestras funciones de búsqueda, primero se procesa el nodo actual (en este caso, se compara el nombre del nodo con el buscado) y luego se recorre recursivamente cada uno de sus hijos. Ejemplo la función interna _buscar.

Es necesario definir el concepto de **clase**, que es un tipo de objeto que permiten crear instancias (que son variables creadas con ese "molde") que tiene *atributos* que son propiedades o datos propios de esa instancia, y *métodos* que son funciones del objeto que le permiten acciones. Las clases son la base de la programación orientada a objetos, que permite estructuras reutilizables y modulares. [3]

Y por último un concepto clave para el desarrollo de este trabajo, y dónde reside la potencia de utilizar clases para una estructura de tipo árbol, son las **funciones recursivas**. Son aquellas funciones que se llaman a sí mismas en la propia función, para eso necesitan un *caso base* que es la condición que detiene el llamado al infinito de la función, y luego el *caso recursivo* que es la llamada a la función pero con el argumento modificado para que en la sucesivas recursiones el problema se vaya reduciendo.

[2] Programación 1 UTN. (2025) *Árboles. https://docs.google.com/document/d/10k16oL15EeyOaq92aoi4qwK3t_22X29-FSV2iV-8N1U/edit?tab=t.0*
[3] Clases - documentación de Python (2025) *https://docs.python.org/es/3.13/tutorial/classes.html*

3. Caso Práctico

Nuestro programa de Python se encarga de gestionar y categorizar productos. La posibilidad de trackear stock, precio, actualizar nombre, aplicar categorías y subcategorías, agregar y eliminar categorías así como también agregar y eliminar productos. Por lo tanto estamos planteando una estructura jerarquizada, un caso que es útil el uso de una estructura de tipo árbol.

Para este concepto lo primero fue definir la clase NodoProducto que representa un nodo dentro de la estructura de tipo árbol para organizar productos y categorías. También podríamos haber modularizado más esta idea haciendo una clase para NodoSistema y luego NodoCategoria y NodoProducto como clases derivadas que hereden de NodoSistema[4], optamos por la por abordaje actual para no complejizar de más.

Esta clase *NodoProducto* contiene los atributos:

- **nombre** (`str`): Nombre del producto o categoría.
- **precio** (`float`, opcional): Precio del producto. Puede ser `None` si es una categoría.
- **stock** (`int`, opcional): Cantidad en stock del producto. Puede ser `None` si es una categoría.
- **es_producto** (`bool`): Indica si el nodo es un producto (`True`) o una categoría (`False`).
- **hijos** (`dict`): Diccionario de nodos hijos, donde la key es el nombre y el value es el nodo hijo.

Y los siguientes métodos:

- **agregar_hijo**(nodo): Agrega un nodo hijo al nodo actual.
- **listar_productos**: Retorna un array con todos los productos hijos del nodo, e incluso a través de la recursividad utiliza una estructura iterativa *for* para evaluar en cada uno de sus hijos si hay productos y sumarlos al array.
- **actualizar_producto**(self, nuevo_nombre=None, nuevo_precio=None, nuevo_stock=None): actualiza los atributos de un producto, previa verificación del atributo es_producto.

La clase SistemaProductos actúa como el controlador principal que gestiona toda la estructura del árbol de productos. Al instanciarse, crea automáticamente un nodo raíz llamado "Catálogo de Productos" que sirve como punto de partida para toda la jerarquía.

A esta clase le proveimos los siguientes métodos para gestionar dicho sistema:

buscar_nodo(nombre, nodo_inicial=None): Implementa una búsqueda recursiva para localizar cualquier nodo en el árbol por su nombre. Utiliza una función anidada *_buscar* que recorre todos los nodos hasta encontrar una coincidencia exacta. Es fundamental para las operaciones posteriores ya que permite localizar tanto productos como categorías.

[4] Clases - documantación Python. Capítulo 9.5 Herencias (2025)
https://docs.python.org/es/3.13/tutorial/classes.html#inheritance

agregar_categoria(nombre, ruta_padre=None): Crea una nueva categoría, si se especifica un nodo padre lo anexa a ese nodo. Si no se proporciona *ruta_padre*, la categoría se agrega directamente como hijo de la raíz. El método incluye validaciones para verificar que la categoría padre exista y que no sea un producto, y también evitar duplicados en el mismo nivel.

eliminar_categoria(nombre_categoria): Elimina una categoría del sistema. Utiliza una función recursiva interna *_encontrar_y_eliminar_padre* que busca la categoría en todo el árbol y verifica que sea una categoría (no un producto) y que esté vacía antes de eliminarla.

agregar_producto(categoria, nombre, precio, stock): Añade un nuevo producto a una categoría. Primero encuentra la categoría de destino, valida que no sea un producto, y luego crea un nuevo nodo con *es_producto*=True.

eliminar_producto(nombre_producto): Similar al método de eliminar categorías, pero específicamente para productos. Busca en todo el árbol el producto y lo elimina de la categoría padre.

mostrar_arbol(nodo=None, prefijo=""): Genera una representación visual del árbol completo.

El código incluye una demostración completa que con todas las funcionalidades. Primero inicializamos creando una instancia del sistema, en la variable sistema. Al crear una instancia de la clase SistemaProductos se crea el primer nodo “Catálogo de Productos” a partir del cual creamos una primera estructura estableciendo tres categorías principales (Almacén, Lácteos, Limpieza). Para demostrar el uso de la prop *ruta_padre* en el método agregar_categoria, añadimos subcategorías dentro de Almacén, así como también al intentar agregar subcategoría dentro de categorías que no existen devuelve un mensaje de error.

Insertamos productos con todos sus datos utilizando el método *agregar_producto*. Una vez realizado esa operación testeamos el método de listar productos dentro de una categoría, para ello primero buscamos el nodo con el método *buscar_nodo* y luego llamamos al método *listar_productos* como éste método retorna un array lo guardamos en una variable *productos_lacteos* y vamos a utilizar una iteración sobre ese array para mostrar por consola.

La siguiente prueba es actualizar los datos de un producto, buscamos el nodo y llamamos el método pasando como argumento *nuevo_precio* y *nuevo_stock*. Y por último hacemos una demostración de los métodos *eliminar_producto* y *eliminar_categoria* que cada uno se encarga de un caso particular y debería mostrar un error en caso de que se esté llamando para eliminar un nodo incorrecto.

En este trabajo buscamos demostrar cómo las estructuras de árbol proporcionan una solución para problemas de categorización jerárquica, tratamos mantener una simplicidad del código al mismo tiempo que tratando de proporcionar funcionalidades completas de gestión. Para eso también nos ayudamos de Inteligencia Artificial sobre todo en hacer el código más compacto y

simplificado. Así como ayudándonos a hacer la prueba de implementación más estética y entendible para el usuario.

4. Metodología Utilizada

La elección de una estructura de árbol para este sistema de gestión de productos se fundamenta en varias razones técnicas y prácticas. La representación natural de jerarquías, ya que los catálogos de productos tienen una organización inherentemente jerárquica (Almacén > Bebidas > Gaseosas > Coca-Cola), que se mapea perfectamente con la estructura de árbol donde cada nodo puede tener múltiples hijos.

A diferencia de estructuras lineales como listas o arrays, los árboles permiten agregar nuevas categorías y subcategorías sin reestructurar todo el sistema. Además los árboles como presentan la relación padre-hijo, facilitan acciones como "listar todos los productos de una categoría" o "calcular el valor total del inventario por sección".

En ese sentido la implementación de nodos crea una estructura flexible en el que un mismo tipo de nodo puede representar tanto categorías como productos, diferenciados por el atributo booleano es_producto.

La recursividad es fundamental en este sistema para tres operaciones principales, el primero la búsqueda; la función *buscar_nodo*() explora cada rama del árbol hasta encontrar el nodo objetivo. Esta forma garantiza que se encuentre cualquier elemento sin importar su profundidad en la jerarquía. El método *listar_productos*() utiliza recursividad para recopilar todos los productos desde un nodo dado hacia abajo, incluyendo productos en subcategorías anidadas. Y por último las funciones de eliminación emplean recursividad para localizar elementos en cualquier nivel del árbol y verificar dependencias antes de la eliminación, asegurando que no se eliminen categorías con contenido.

5. Resultados Obtenidos

El sistema desarrollado demuestra capacidades completas de gestión de inventario. Se logró implementar un sistema que maneja categorías y subcategorías de profundidad arbitraria, como se evidencia en la estructura Almacén > Bebidas > productos específicos. También el sistema soporta todas las operaciones fundamentales, como crear, buscar y listar, actualizar datos y eliminar (CRUD).

Se intentó implementar validaciones, a un nivel básico, que previene eliminar categorías que no estén vacías, crear un nodo hijo en categorías que no existen.

6. Conclusiones

La implementación de un sistema de gestión de productos basado en árboles fue un ejercicio desafiante para implementar la estructura de los árboles. Esta arquitectura modular de clases facilita agregar nuevas funcionalidades como reportes, búsquedas avanzadas o integración con bases de datos externas.

En el trayecto aprendimos a implementar capaz de validación que prevengan errores, e hicimos uso de las funciones recursivas como soluciones para problemas de recorridos, aunque nos costó algunas veces diseñar el manejo de casos base para evitar stack overflow.

Algunas ideas que se podrían añadir a futuro podrían ser:

- Interfaz gráfica de usuario para facilitar la interacción
- Análisis de datos y generación de reportes automatizados
- Validación de rutas.
- Métodos para guardar y cargar la estructura en archivos (por ejemplo, JSON).
- Métodos para agregar/disminuir stock.

7. Bibliografía

Árboles. (n.d.). Google Docs.

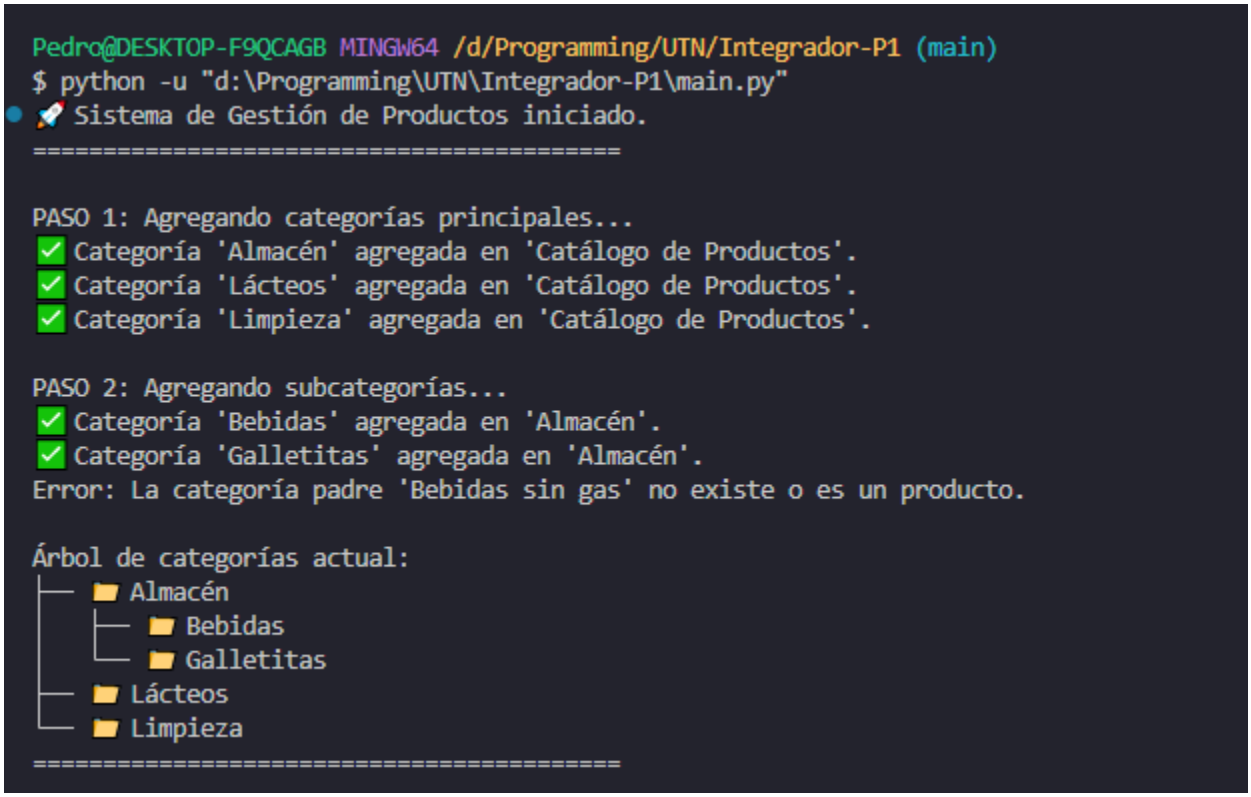
https://docs.google.com/document/d/10k16oL15EeyOaq92aoi4qwK3t_22X29-FSV2iV-8N1U/edit?tab=t.0

9. Clases — documentación de Python - 3.13.4. (n.d.). Python documentation. Retrieved June 9, 2025, from https://docs.python.org/es/3.13/tutorial/classes.html

Árboles. (s/f). Google Docs. Recuperado el 8 de junio de 2025, de https://docs.google.com/document/d/10k16oL15EeyOaq92aoi4qwK3t_22X29-FSV2iV-8N1U/edit?tab=t.0

8. Anexos

• Repositorio en GitHub: https://github.com/dibaxu/Integrador-P1
• Video: https://youtu.be/KO8ORZHhQRY



```
PASO 3: Agregando productos...
✅ Producto 'Leche Entera' agregado a la categoría 'Lácteos'.
✅ Producto 'Yogur de Frutilla' agregado a la categoría 'Lácteos'.
✅ Producto 'Gaseosa Cola' agregado a la categoría 'Bebidas'.
✅ Producto 'Lavandina' agregado a la categoría 'Limpieza'.

Árbol con productos:
├── 📁 Almacén
│   ├── 📁 Bebidas
│   │   └── 📦 Gaseosa Cola (Precio: $250.0, Stock: 100)
│   └── 📁 Galletitas
├── 📁 Lácteos
│   ├── 📦 Leche Entera (Precio: $150.5, Stock: 50)
│   └── 📦 Yogur de Frutilla (Precio: $99.99, Stock: 30)
└── 📁 Limpieza
    └── 📦 Lavandina (Precio: $120.0, Stock: 40)
========================================
```

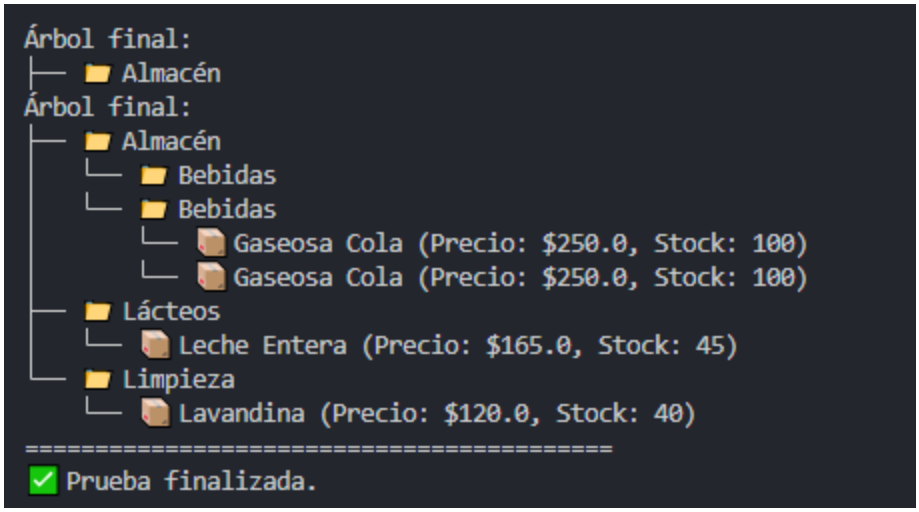
```
PASO 6: Eliminando el producto 'Yogur de Frutilla' y la categoría 'Galletitas'...
    └── 📦 Lavandina (Precio: $120.0, Stock: 40)
========================================

PASO 6: Eliminando el producto 'Yogur de Frutilla' y la categoría 'Galletitas'...
========================================

PASO 6: Eliminando el producto 'Yogur de Frutilla' y la categoría 'Galletitas'...
🗑️Producto 'Yogur de Frutilla' eliminado correctamente.
Error: No se puede eliminar la categoría 'Lácteos' porque no está vacía.
🗑️Categoría 'Galletitas' eliminada correctamente.

PASO 6: Eliminando el producto 'Yogur de Frutilla' y la categoría 'Galletitas'...
🗑️Producto 'Yogur de Frutilla' eliminado correctamente.
Error: No se puede eliminar la categoría 'Lácteos' porque no está vacía.
🗑️Categoría 'Galletitas' eliminada correctamente.

🗑️Producto 'Yogur de Frutilla' eliminado correctamente.
Error: No se puede eliminar la categoría 'Lácteos' porque no está vacía.
🗑️Categoría 'Galletitas' eliminada correctamente.
```


Árbol final:
Almacén
Árbol final:
Almacén
Bebidas
Bebidas
Gaseosa Cola (Precio: $250.0, Stock: 100)
Gaseosa Cola (Precio: $250.0, Stock: 100)
Lácteos
Leche Entera (Precio: $165.0, Stock: 45)
Limpieza
Lavandina (Precio: $120.0, Stock: 40)
Prueba finalizada.