Trabajo Práctico Integrador
Programación I

Árboles con listas en Python

Alumnos:
Lautaro Lucero - lautalucero@gmail.com
Gabriela Machin - gabrielamachin.gm@gmail.com

Materia: Programación I

Fecha de Entrega: 09/06/2025

UNIVERSIDAD TECNOLÓGICA NACIONAL

Índice:

Introducción

En el campo de la informática y particularmente en las estructuras de datos, los árboles ocupan un lugar fundamental debido a su capacidad para organizar la información de manera jerárquica y eficiente. Permiten representar relaciones entre elementos, como jerarquías organizacionales, estructuras de archivos o expresiones matemáticas. Además, se utilizan en diversas áreas, como sistemas operativos, gráficos, bases de datos y redes de computadoras. Una de las formas más didácticas y accesibles de implementar árboles es mediante el uso de listas en Python, un lenguaje que suele ser elegido para el desarrollo de software por su simplicidad y versatilidad.

A diferencia de las implementaciones más formales con clases y nodos enlazados, los árboles con listas ofrecen una manera estructurada pero sencilla de comprender los principios básicos del funcionamiento de un árbol binario. Esta aproximación permite visualizar con claridad conceptos como raíz, hijos, recorridos y profundidad. El objetivo de este trabajo es analizar esta forma de implementación y presentar un caso práctico que demuestre cómo se construye y manipula un árbol utilizando listas en Python, destacando sus aplicaciones y sus posibles limitaciones.

Objetivos

- Demostrar nuestro entendimiento sobre cómo se crean, cómo funcionan y cómo se pueden representar los árboles binarios hechos con listas en Python.
- Desarrollar funciones fundamentales para trabajar con árboles, como agregar nuevos elementos y recorrer sus nodos de distintas formas.
- Comparar los árboles binarios implementados con listas con los que se hacen usando clases y objetos para comprender sus diferencias y ventajas.

Marco Teórico

Un árbol es una estructura de datos abstracta que organiza elementos de manera jerárquica. A diferencia de estructuras lineales como listas o arreglos, los árboles permiten una organización en niveles, en donde cada elemento (o nodo) puede tener múltiples hijos, pero solo un padre, con excepción del nodo raíz, que no tiene antecesor (Miller & Ranum, 2011a).

Formalmente, un árbol T es un conjunto de nodos que almacena elementos con una relación padre-hijo que cumple con las siguientes propiedades:

- Si T no está vacío, contiene un nodo especial llamado raíz, que no tiene padre.
- Cada nodo distinto de la raíz tiene un único padre.
- Un árbol puede estar vacío, lo cual permite una definición recursiva en la que un árbol es o bien vacío, o bien un nodo raíz con un conjunto (posiblemente vacío) de subárboles cuyos nodos raíz son los hijos de este (Miller & Ranum, 2011a).

En el estudio de los árboles, existen conceptos clave que permiten su comprensión y uso en algoritmos:

- **Nodo**: Parte fundamental del árbol. Puede tener una clave (“key”) y opcionalmente un valor asociado (“payload”).
- **Raíz (Root)**: Nodo sin aristas entrantes, es decir, sin padre.
- **Hijos (Children)**: Nodos con aristas entrantes desde un mismo nodo superior.
- **Padre (Parent)**: Nodo que tiene aristas salientes hacia sus hijos.
- **Hermanos (Siblings)**: Nodos que comparten un mismo padre.
- **Subárbol (Subtree)**: Conjunto de un nodo y todos sus descendientes.
- **Nodo hoja (Leaf)**: Nodo que no tiene hijos.
- **Nivel (Level)**: Número de aristas entre la raíz y un nodo dado.
- **Altura del árbol (Height)**: Nivel máximo que alcanza cualquier nodo en el árbol (Miller & Ranum, 2011b).

Uno de los tipos de árboles más relevantes en programación es el **árbol de búsqueda binaria** (BST, por sus siglas en inglés), que permite realizar eficientemente operaciones dinámicas sobre conjuntos de datos, tales como búsqueda, inserción, eliminación, y recuperación de mínimos y máximos (Cormen et al., 2022).

Además de su estructura, los árboles binarios permiten diferentes formas de recorridos o traversals, los cuales son fundamentales para procesar o visitar sistemáticamente los nodos. Existen tres recorridos principales definidos para árboles binarios:

Recorrido inorden (inorder traversal): En este recorrido se visita primero el subárbol izquierdo, luego el nodo actual, y finalmente el subárbol derecho. En árboles de búsqueda binaria, este recorrido devuelve los elementos en orden ascendente.

Recorrido preorden (preorder traversal): Primero se visita el nodo actual, seguido del subárbol izquierdo y finalmente el subárbol derecho. Es útil para copiar árboles o generar expresiones en notación prefija, que es una forma de escribir operaciones sin paréntesis, en la que los operadores van antes que sus operandos que se usa en algunos lenguajes de programación, compiladores y calculadoras sin necesidad de paréntesis.

Recorrido postorden (postorder traversal): En este caso, se visita primero el subárbol izquierdo, luego el derecho, y al final el nodo actual. Es comúnmente usado para eliminar o liberar memoria de árboles y para evaluar expresiones en notación postfija, la cual es otra forma de escribir expresiones matemáticas sin usar paréntesis, en la que los operadores van después de los operandos.

Cada uno de estos recorridos tiene aplicaciones específicas dependiendo del problema a resolver. Por ejemplo, el recorrido inorden es clave en algoritmos de ordenación sobre árboles, mientras que el postorden es útil en evaluación de expresiones aritméticas representadas como árboles binarios (Knuth, 1997).

Metodología Utilizada

El desarrollo del trabajo se llevó a cabo en varias etapas previamente definidas. Comenzamos realizando una investigación teórica sobre el tema de implementación de árboles utilizando listas en Python. Consultamos diversas fuentes de información, comenzando por la bibliografía propuesta por la cátedra, la cual nos brindó una base conceptual sólida. Luego profundizamos la investigación con material bibliográfico adicional encontrado por nuestra cuenta, complementando la información con videos explicativos de YouTube. Cada integrante del grupo contribuyó activamente en esta etapa

La etapa siguiente fue la implementación práctica en Python. Lautaro fue el encargado de crear un repositorio para el proyecto. Utilizamos Visual Studio Code como entorno de desarrollo y Git como herramienta de control de versiones y colaboración. Ambos integrantes del grupo, Lautaro y Gabriela, realizaron diferentes commits al repositorio, contribuyendo de manera continua con mejoras al código, incorporación de nuevas funcionalidades y documentación en el archivo Readme. Durante este proceso, fuimos testeando el código con distintos ejemplos y casos de prueba. Esto nos permitió verificar el funcionamiento correcto e ir corrigiendo errores a medida que surgían.

Finalmente, como cierre del trabajo, realizamos de forma conjunta la creación del video explicativo que resume el desarrollo y funcionamiento del proyecto.

Desarrollo / Implementación

La implementación del árbol binario se realizó utilizando listas anidadas en Python, lo que nos permitió representar cada nodo como una lista de tres elementos: el valor del nodo, el hijo izquierdo y el hijo derecho. Esta decisión responde a la necesidad de mantener el enfoque didáctico del trabajo, priorizando la claridad en la construcción y manipulación de la estructura, sin introducir aún el uso de clases u objetos.

Uno de los aspectos más importantes del diseño fue la forma en la que el árbol se construye. Optamos por una función recursiva (construir_arbol) que guía al usuario en el proceso de creación, solicitando la información de manera secuencial

y contextual. Primero se pide el valor del nodo, y luego se consulta si ese nodo tendrá hijo izquierdo y/o derecho. En caso afirmativo, la función se vuelve a llamar para construir el subárbol correspondiente.

Este enfoque interactivo tiene dos ventajas principales: Por un lado, resulta amigable para el usuario, ya que no requiere conocimientos previos sobre estructuras complejas ni escritura de sintaxis específica. El árbol se forma a partir de preguntas simples, lo cual favorece la comprensión.

Por otro lado, nos simplificó la lógica desde el punto de vista del desarrollo, ya que evitamos manejar estructuras previas o validaciones adicionales complejas, trabajando con una recursividad natural y progresiva.

Además de la construcción del árbol, implementamos funciones para recorrerlo en tres órdenes clásicos: preorden, inorden y postorden. Cada una de estas funciones aplica un algoritmo recursivo que refleja las definiciones teóricas tratadas previamente, y permite observar cómo se procesan los nodos en cada caso.

Finalmente, desarrollamos una función adicional (imprimir_arbol) que imprime la estructura del árbol de manera jerárquica y visual, utilizando sangrías y prefijos para identificar la posición de cada nodo (raíz, hijo izquierdo, hijo derecho). Esta herramienta fue clave para validar la estructura creada y para facilitar su interpretación, especialmente al trabajar desde consola.

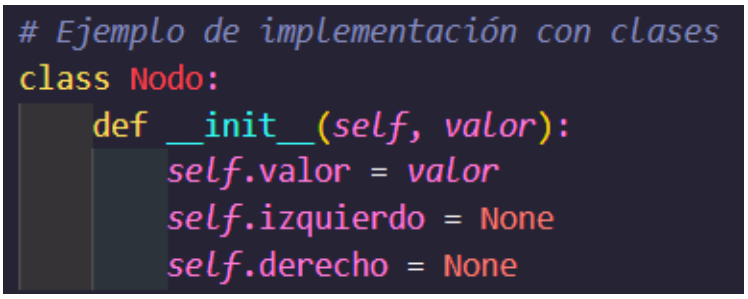
Resultados Obtenidos

Con la investigación realizada para este trabajo práctico, se aprendió cómo funcionan, para qué sirven y cómo se implementan los árboles en Python. Descubrimos que los árboles son estructuras de datos muy útiles, especialmente para organizar información jerárquica y realizar búsquedas, recorridos o evaluaciones de expresiones.

Como somos principiantes en programación, decidimos implementar los árboles utilizando listas debido a su simplicidad y facilidad de visualización. Esto nos permitió concentrarnos en entender la lógica de los árboles sin complicarnos con detalles más avanzados de la programación orientada a objetos.

Sin embargo, también investigamos cómo se implementan los árboles utilizando clases y objetos, lo cual es el enfoque más utilizado en programación

profesional. Esta implementación define un nodo como una instancia de una clase Nodo, con atributos como valor, izquierdo y derecho. Aunque más compleja al principio, esta técnica ofrece más flexibilidad, legibilidad y escalabilidad, especialmente en proyectos grandes o con árboles dinámicos.



```
# Ejemplo de implementación con clases
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izquierdo = None
        self.derecho = None
```

En resumen, implementar árboles con listas es útil como primer acercamiento, ya que nos permitió experimentar con la estructura de forma rápida y directa. No obstante, a medida que vamos progresando en el aprendizaje de programación, la implementación con clases y objetos se vuelve mucho más recomendable, sobre todo para aplicaciones reales donde la estructura del árbol puede volverse compleja o requiere operaciones más avanzadas.

Conclusión

En conclusión, la implementación de árboles binarios utilizando listas nos resultó una herramienta didáctica muy útil para comprender los conceptos fundamentales de esta estructura de datos. Esta forma de trabajo nos permitió enfocarnos en la lógica del árbol, es decir, cómo se construye, cómo se recorre y cómo se representa.

Si bien el uso de listas es apropiado para proyectos simples o con fines de aprendizaje, también reconocemos sus limitaciones, especialmente cuando se trata de árboles más grandes, desbalanceados o que requieren modificaciones dinámicas. En esos casos, la implementación con clases y objetos ofrece una solución más robusta y mantenible.

Este trabajo no solo nos permitió adquirir los conocimientos de árboles con listas en Python, sino también continuar poniendo en práctica el trabajo colaborativo, control de versiones y documentación de código. A medida que avancemos en nuestro aprendizaje de programación, nos proponemos adoptar enfoques más avanzados como el uso de clases para aprovechar al máximo el potencial de las estructuras de datos en aplicaciones reales.

Bibliografía

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). The MIT Press.
- Miller, B. N., & Ranum, D. L. (2013). *Problem solving with algorithms and data structures using Python* (3th ed.). Franklin, Beedle & Associates.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Data structures and algorithms in Python*. Wiley.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3rd ed.). Addison-Wesley.
- Vida MRR - Programación Web. (2020, 27 mayo). ¿Qué son y cómo funcionan los árboles? [Video]. YouTube. https://youtu.be/tBaOQeyXYqg
- Tecnicatura Universitaria en Programación. (2020, 10 octubre). Implementación de árboles como listas anidadas [Video]. YouTube. https://youtu.be/-D4SxeHQGIg
- BitBoss. (2021, 1 abril). Estructuras de datos con Python en 8 minutos: Listas, Tuplas, Conjuntos y Diccionarios [Video]. YouTube. https://youtu.be/v25-m1LOUiU