Título del proyecto: Estructuras de datos avanzados, Árboles y búsqueda binaria

Alumnos: Choque, Javier (ricardojavier.choque@gmail.com) Campana, Mario (mariol.campana@gmail.com)

Materia: Programación I

Profesor: Ariel Enferrel

Tutora: Martina Zabala

Fecha de Entrega: 9 de Junio de 2025

1 Introducción

Introducción

El presente trabajo práctico aborda la implementación y manipulación de estructuras jerárquicas mediante árboles binarios representados con listas anidadas en el lenguaje Python. El tema fue elegido por su valor didáctico al permitir comprender de forma visual y estructurada conceptos fundamentales de la programación estructurada y recursiva, así como también por su aplicabilidad en diversos contextos reales como jerarquías organizativas, árboles genealógicos, sistemas de archivos y estructuras de datos en informática.

En el ámbito de la programación, el estudio de árboles resulta esencial para la formación de una lógica robusta. Esta estructura permite modelar relaciones jerárquicas, aplicar recorridos sistemáticos (como pre-orden, in-orden y post-orden), y resolver problemas con métodos eficientes de búsqueda, como la búsqueda binaria. A través de este trabajo, se busca reforzar el uso de la recursividad, la manipulación de listas complejas y el análisis de estructuras con distintas profundidades.

El objetivo principal del desarrollo es construir un sistema capaz de representar una provincia, dividida en distritos, escuelas y aulas, mediante un árbol binario. A partir de esta estructura, se pretende implementar funcionalidades que permitan visualizar la jerarquía completa, buscar elementos y determinar su profundidad, localizar alumnos por ID, y aplicar distintos tipos de recorridos sobre el árbol. De este modo, se logra integrar conocimientos de estructuras de datos, algoritmos de búsqueda y diseño lógico, fortaleciendo competencias clave en la formación del programador.

2. Marco teórico

Un árbol es una estructura de datos jerárquica fundamental en informática y programación. Está compuesto por nodos conectados, donde cada nodo almacena un valor y puede estar vinculado a otros nodos descendientes llamados hijos y/o a uno superior llamado padre. El primer nodo del árbol se denomina “raíz”, mientras que aquellos sin hijos se conocen como “hojas” (Goodrich, Tamassia & Goldwasser, 2013)[1]. En los árboles binarios, cada nodo puede tener como máximo dos hijos.

[1] Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Estructuras de datos y algoritmos en Python*. Wiley.

Los árboles poseen varias propiedades estructurales esenciales. Estas propiedades permiten analizar y optimizar operaciones sobre árboles, como búsquedas, inserciones o recorridos.

- **Camino**: secuencia de nodos desde un nodo origen hasta un nodo destino.
- **Longitud del camino**: cantidad de aristas o ramas en ese camino.
- **Nivel**: distancia desde la raíz hasta un nodo (la raíz está en el nivel 0).
- **Profundidad**: número de niveles desde la raíz hasta un nodo dado.
- **Altura**: distancia máxima desde la raíz hasta una hoja.
- **Grado de un nodo**: número de hijos directos que posee.
- **Orden del árbol**: número máximo de hijos que puede tener un nodo (2 en el caso de un árbol binario).
- **Peso**: cantidad total de nodos que contiene el árbol.

En cuanto a los recorridos de un árbol binario, existen tres formas principales que permiten visitar sistemáticamente todos los nodos del árbol. Cada tipo de recorrido responde a diferentes necesidades según el tipo de operación que se desee realizar:

- **Recorrido In-Orden (Izquierda - Raíz - Derecha)**: este tipo de recorrido primero recorre el subárbol izquierdo, luego procesa la raíz y finalmente recorre el subárbol derecho. Es especialmente útil cuando se desea obtener los elementos del árbol en orden ascendente, como ocurre en los árboles binarios de búsqueda, donde este recorrido devuelve una secuencia ordenada de los valores almacenados. Según Cormen et al. (2009)[2], este recorrido es clave en estructuras de datos que requieren ordenamiento dinámico eficiente.
- **Recorrido Pre-Orden (Raíz - Izquierda - Derecha)**: en este caso se procesa primero la raíz, luego se recorre el subárbol izquierdo y finalmente el derecho. Es utilizado comúnmente para clonar árboles, ya que permite copiar primero la estructura de cada nodo antes de visitar a sus hijos. También es útil en la serialización de estructuras jerárquicas para su almacenamiento o transmisión.
- **Recorrido Post-Orden (Izquierda - Derecha - Raíz)**: en este tipo de recorrido se visitan primero ambos subárboles (izquierdo y derecho) y al final se procesa la raíz. Es muy utilizado en operaciones que requieren la eliminación segura de todos los nodos del árbol, como al liberar memoria. Además, es útil en la evaluación de expresiones matemáticas representadas en árboles sintácticos, ya que respeta el orden correcto de operaciones (jerárquicas) al evaluar desde los operadores hacia la raíz del árbol (el operador principal).

Estos recorridos forman la base de numerosas aplicaciones informáticas, desde el análisis de código fuente en compiladores hasta algoritmos de búsqueda en sistemas complejos. El dominio

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

de estos recorridos es esencial para cualquier programador que trabaje con estructuras jerárquicas o grafos (Knuth, 1997)[3].

Los árboles tienen múltiples aplicaciones prácticas, como representación jerárquica en bases de datos, compiladores donde se usan árboles de sintaxis abstracta para analizar código fuente, algoritmos en inteligencia artificial como árboles de decisión, estructuras de índice y búsqueda en motores de búsqueda y sistemas de archivos. Como indica Cormen et al. (2009), los árboles permiten organizar datos complejos de manera eficiente, reduciendo la complejidad algorítmica en diversas tareas. Además, si bien Python no cuenta con una estructura de árbol incorporada, su implementación mediante clases y estructuras enlazadas es una herramienta educativa valiosa (Python Software Foundation, 2024)[4].

3 Caso Práctico

El caso práctico desarrollado en este trabajo consiste en la implementación de una estructura de árbol jerárquico utilizando listas anidadas en el lenguaje de programación Python. La estructura representa la organización territorial y educativa de una provincia ficticia denominada "Neuquén", dividida en distritos, escuelas y aulas. A su vez, se incorpora una lista de alumnos identificados por un ID único y apellido, con el objetivo de aplicar algoritmos de búsqueda y análisis estructural.

La estructura general del árbol parte desde la raíz, que representa la provincia. Esta contiene dos nodos hijos correspondientes a los distritos, y cada distrito contiene dos escuelas. Finalmente, cada escuela posee dos aulas. Esta jerarquía binaria fue elegida para facilitar la implementación de funciones recursivas que permitan recorrer y analizar la estructura completa.

Entre las funcionalidades desarrolladas se incluyen:

- **Impresión jerárquica del árbol**: mediante una función recursiva, se muestra visualmente la estructura del árbol con indentaciones y guiones que reflejan los niveles jerárquicos.

- **Búsqueda de elementos**: se implementó una función de búsqueda recursiva que permite encontrar un nodo dado (por ejemplo, una escuela o aula) dentro del árbol y devuelve el camino completo desde la raíz hasta el nodo. Esto permite calcular automáticamente propiedades como profundidad, nivel y cantidad de nodos involucrados.

- **Recorridos clásicos del árbol binario**: se añadieron funciones para realizar los recorridos in-orden, pre-orden y post-orden, fundamentales para diferentes tipos de procesamiento sobre estructuras jerárquicas. Estas funciones permiten, por ejemplo, visualizar el árbol en distintos órdenes de visita según los requerimientos de análisis.

- **Búsqueda de alumnos**: utilizando estructuras tipo diccionario, se aplicaron dos algoritmos de búsqueda —lineal y binaria— para encontrar un alumno por su ID. Se midió el tiempo de ejecución de ambas búsquedas, lo que permitió comparar su eficiencia en términos de tiempo de procesamiento.

Este caso práctico permitió aplicar conceptos teóricos fundamentales sobre árboles binarios y búsquedas, reforzando su comprensión a través de la implementación en un lenguaje de propósito general. A su vez, permitió observar diferencias en rendimiento algorítmico, y representar una

[3] Knuth, D. E. (1997). *The art of computer programming, Volume 1: Fundamental algorithms* (3rd ed.). Addison-Wesley.
[4] Python Software Foundation. (2024). *The Python language reference*. https://docs.python.org/3/

estructura jerárquica realista, como lo es el sistema educativo, con fines didácticos y computacionales.

(Ver Código anexado)

4. Metodología Utilizada

El desarrollo del presente trabajo se llevó a cabo siguiendo una metodología estructurada en varias etapas, combinando la investigación teórica con la implementación práctica en el lenguaje de programación Python. A continuación se describen los principales pasos y recursos empleados durante su realización:

4.1 Investigación previa

En una primera instancia, se realizó una revisión teórica de los conceptos fundamentales relacionados con árboles binarios, sus propiedades y tipos de recorridos. Las principales fuentes consultadas fueron libros de referencia en el área de estructuras de datos como *Data Structures and Algorithms in Python* (Goodrich, Tamassia & Goldwasser, 2013) y *Introduction to Algorithms* (Cormen et al., 2009), además de recursos oficiales como la documentación de Python (Python Software Foundation, 2024). Esta etapa permitió consolidar el marco teórico necesario para el diseño de la solución. Además, fueron fundamentales los usos de las unidades sobre listas, datos complejos, y recursividad vistos en la materia anteriormente, sin los cuáles no se hubiera tenido la base para entender cómo funcionan las estructuras de árboles en Python.

4.2 Diseño de la estructura jerárquica

Se diseñó una representación jerárquica que simula una estructura administrativa de un sistema educativo provincial, en la cual se definió una provincia compuesta por distritos, escuelas y aulas. Se utilizaron listas anidadas para reflejar esta jerarquía de forma binaria, para hacer sencillo su recorrido y el procesamiento mediante funciones recursivas.

4.3 Implementación y pruebas del código

La programación se realizó utilizando Python 3. Durante esta etapa se implementaron diversas funciones:

- Impresión visual del árbol con indentación.
- Búsqueda recursiva de nodos.
- Cálculo de propiedades como nivel y profundidad.
- Recorridos in-orden, pre-orden y post-orden del árbol.
- Búsqueda de alumnos mediante algoritmos de búsqueda lineal y binaria, incluyendo la medición de tiempos de ejecución para comparar eficiencia.

Cada función fue probada de forma individual y luego integrada al sistema principal mediante un menú interactivo que permite al usuario seleccionar distintas acciones. Se utilizó el método de prueba y error para validar el funcionamiento correcto de la lógica recursiva y los algoritmos de búsqueda.

4.4 Herramientas y recursos utilizados

El desarrollo se realizó en el software Visual Studio Code, con el intérprete de Python configurado localmente. Las pruebas y ejecución también se hicieron utilizando este mismo. Las funciones de medición de tiempo se realizaron con la librería estándar time. Para limpiar la pantalla durante la ejecución, se empleó la función os.system("cls") propia del módulo os.

4.5 Trabajo colaborativo

El desarrollo del trabajo se realizó en parejas, promoviendo una división equilibrada de tareas y fomentando el aprendizaje colaborativo. Uno de los integrantes se enfocó principalmente en el diseño de la estructura jerárquica, el armado del menú interactivo y las funciones de búsqueda. Mientras que el otro se encargó de la medición de eficiencia de los algoritmos y de la implementación de las funciones de de recorridos del árbol.

Ambos participantes colaboraron en la redacción del informe y en la validación conjunta del funcionamiento del código, revisando y probando cada componente para asegurar su correcto comportamiento.

Del mismo modo, se utilizaron herramientas de repositorios como Github cuando se consideró necesario. Otras veces, por ser un trabajo de dos integrantes y con un código relativamente sencillo de separar y trabajar modularmente, se decidió ir compartiendo los avances por whatsapp (enviando el archivo entero con las modificaciones) para que el otro integrante valide los cambios. Asimismo, se trabajó en videollamadas cuando se presentaban problemas en la ejecución del código y no podían resolverse individualmente.

5 Resultados obtenidos

A partir del desarrollo del programa y su ejecución, se obtuvieron resultados correctos que validan el correcto diseño y la implementación de la estructura jerárquica binaria y de los algoritmos de recorrido y búsqueda.

Se logró representar con éxito una estructura jerárquica compuesta por una provincia, distritos, escuelas y aulas, utilizando listas anidadas con lógica binaria. A través de la función de impresión recursiva (imprimir_arbol), fue posible visualizar la jerarquía completa de forma ordenada y comprensible, utilizando indentación y guiones para reflejar niveles de profundidad.

Mediante la función buscar_elemento, el sistema permitió localizar cualquier nodo de la jerarquía y devolver el camino desde la raíz hasta dicho nodo, junto con sus propiedades estructurales como nivel, profundidad y cantidad de nodos involucrados en el trayecto. Esto permite al usuario comprender la ubicación exacta de un elemento dentro del árbol.

También se implementaron correctamente las tres formas clásicas de recorrido de árboles binarios:

- Mostrar el contenido ordenado (in-orden),
- Obtener una copia estructurada del árbol (pre-orden),
- Recorrer desde los nodos hoja hacia la raíz (post-orden).

La ejecución de estas funciones confirmó el correcto comportamiento de la lógica recursiva aplicada.

Sin embargo, es necesario aclarar que por el tipo de sistema desarrollado (sistema educativo en una provincia) quizás el recorrido más útil es el recorrido pre-orden que estructura el árbol comenzando por la provincia, siguiendo con el distrito 1, sus escuelas y sus aulas y luego el distrito 2 de la misma manera. A los efectos de mostrar el resto de los recorridos, se decidió también incorporarlos, pero se acordó en que, en este caso, no resultan tan útiles.

En cuanto a la búsqueda de alumnos por ID, se implementaron y evaluaron dos enfoques: búsqueda lineal y búsqueda binaria. Ambas lograron localizar al alumno correspondiente correctamente. La búsqueda binaria resultó ser más rápida y eficiente cumpliendo así con las expectativas teóricas respecto a su desempeño.

El programa principal incluyó un menú interactivo que permite al usuario seleccionar entre distintas funcionalidades (ver jerarquía, buscar un nodo o buscar un alumno). Esto facilitó la prueba de todas las funciones implementadas y brindó una experiencia clara accesible para el usuario.

6 Conclusiones

La realización de este trabajo permitió al grupo afianzar conocimientos fundamentales sobre estructuras de datos, particularmente sobre árboles binarios y su aplicación en entornos prácticos. A través del desarrollo de una jerarquía simulada de aulas, escuelas y distritos, se profundizó en el diseño recursivo, el manejo de listas anidadas en Python y la lógica detrás de recorridos estructurados.

Se comprendió la utilidad de los árboles como herramienta clave en programación, ya que permiten representar información jerárquica de manera ordenada y eficiente. Además, se evidenció la importancia de los algoritmos de búsqueda, tanto secuenciales como binarios, y su impacto en la eficiencia del procesamiento de datos. Estos conceptos tienen aplicaciones directas en proyectos que requieren organización de información, como sistemas educativos, motores de búsqueda, inteligencia artificial o bases de datos.

Durante el trabajo se identificaron posibles mejoras futuras que por cuestiones de tiempo (necesarias para investigar y probar en el código) no se pudieron llevar adelante, como:

- La implementación de una clase Nodo para una representación más robusta del árbol.
- La adición de funciones para insertar y eliminar nodos dinámicamente.
- Una interfaz gráfica que facilite aún más la interacción con el usuario.

Una de las principales dificultades surgió en el manejo recursivo de las listas anidadas y en la visualización clara de la estructura jerárquica. Estas situaciones se resolvieron mediante pruebas iterativas, el uso de la función print() para depurar o poder seguir mejor el flujo de información, revisión colaborativa del código en videollamadas y algunas consultas a documentación oficial, bibliografía, videos sugeridos por la cátedra para el trabajo realizado.

En definitiva, el trabajo no solo cumplió con los objetivos propuestos, sino que también fortaleció las habilidades de programación, resolución de problemas y colaboración entre los integrantes del grupo.

7 Bibliografía

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Estructuras de datos y algoritmos en Python*. Wiley.
- Python Software Foundation. (2024). *Python 3 Documentation*. Disponible en: https://docs.python.org/3
- W3Schools. (2024). Python Classes and Objects. Disponible en: https://www.w3schools.com/python/python_classes.asp
- Materiales y videos brindados por la cátedra de Programación I. Tecnicatura en Programación a Distancia. UTN.

8 Anexos

Capturas del programa funcionando (pdf aparte)

Enlace al video explicativo:

- Parte 1: https://youtu.be/0O8c6M6T9eg
- Parte 2: https://youtu.be/qAfALohpKlM

Código en archivo externo (enlace a Github):

Archivo README (subido a Github)