Trabajo integrador 1 – datos avanzados

Alumnos:

Ciro Giorgini - giorginiciro@gmail.com

Fernando Chacon – fernandochaconamati@gmail.com

Materia: Sistemas Operativos

Profesor: Ariel Enferrel

Fecha de Entrega: 09 de junio de 2025

Índice:

1. Introducción

2. Marco Teórico

3. Caso Práctico

4. Metodología Utilizada

5. Resultados Obtenidos

6. Conclusiones

7. Bibliografía

8. Anexos

1. Introducción:

En este trabajo se aborda el estudio y la implementación de árboles como estructuras de datos avanzadas en programación, particularmente los árboles binarios de búsqueda.

La elección de este tema surge de su gran relevancia tanto teórica como práctica: los árboles permiten organizar datos de forma jerárquica, facilitando operaciones eficientes de búsqueda, inserción y eliminación.

Este tipo de estructura se utiliza ampliamente en sistemas de archivos, algoritmos de inteligencia artificial, motores de bases de datos y procesamiento de expresiones matemáticas, entre otros.

El objetivo principal del trabajo es comprender en profundidad el funcionamiento de los árboles binarios y desarrollar una aplicación práctica que refleje sus principios fundamentales, utilizando Python como lenguaje de implementación.

2. Marco Teórico

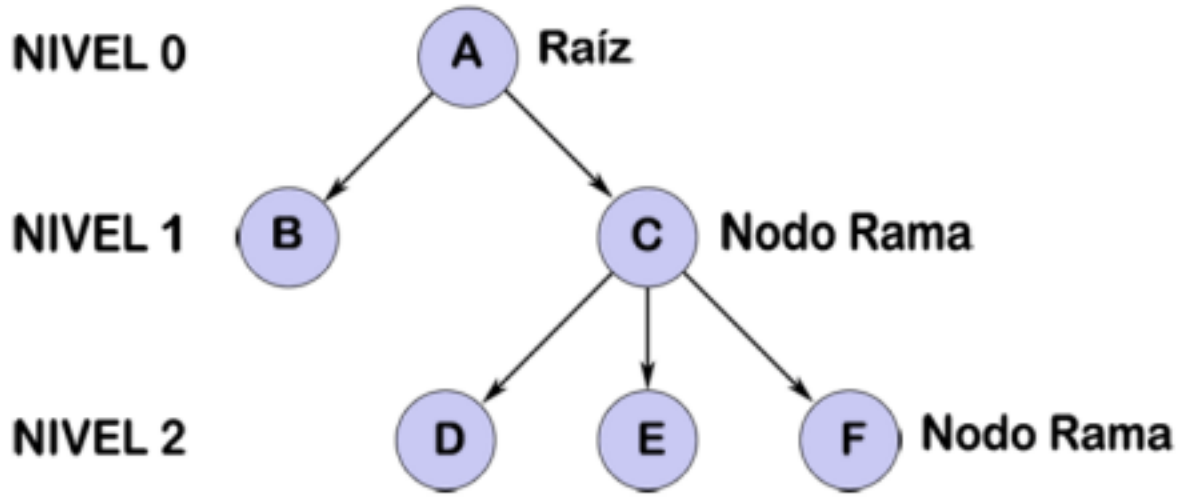
Los árboles son estructuras de datos no lineales ampliamente utilizadas en informática para representar relaciones jerárquicas entre elementos. A diferencia de listas o diccionarios, los árboles permiten organizar los datos en niveles y jerarquías, siendo fundamentales en la representación de sistemas de archivos, árboles genealógicos, bases de datos jerárquicas y algoritmos de toma de decisiones.

Un árbol está compuesto por nodos conectados entre sí por ramas. Cada nodo puede tener uno o varios hijos, excepto el nodo raíz, que es el origen de la estructura, y los nodos hoja, que no tienen descendientes. Existen distintos tipos de nodos (raíz, hoja, internos) y relaciones (padre, hijo, hermano) que definen su estructura.

Entre sus propiedades fundamentales se encuentran la profundidad (distancia desde la raíz hasta un nodo), el nivel (distancia desde un nodo hasta la raíz más uno), la altura del árbol (nivel más alto alcanzado), el grado (número de hijos de un nodo) y el peso (cantidad total de nodos).

Además, existen árboles binarios, donde cada nodo puede tener como máximo dos hijos, y árboles binarios de búsqueda, que mantienen un orden específico que facilita búsquedas eficientes. Estos árboles se pueden recorrer de diferentes formas: preorden, inorden y postorden, cada una con propósitos distintos según el tipo de procesamiento que se desee realizar.

El conocimiento de estas estructuras y su aplicación es esencial para optimizar el uso de memoria, reducir los tiempos de ejecución y mejorar la eficiencia de los algoritmos en numerosos contextos informáticos.



3. Caso Práctico

Descripción del problema

Se planteó como caso práctico la simulación de un árbol binario de búsqueda (ABB), estructura útil para organizar y acceder rápidamente a datos ordenados. El objetivo fue implementar este tipo de árbol en Python para gestionar un conjunto de valores enteros, permitiendo su inserción, búsqueda, eliminación y recorrido de forma eficiente.

Decisiones de diseño

Se optó por implementar un árbol binario de búsqueda debido a su eficiencia en operaciones de inserción, búsqueda y eliminación (tiempo promedio O(log n) si el árbol está balanceado). Se prefirió el recorrido inorden para verificar el orden creciente de los valores, y el preorden y postorden para visualizar el flujo jerárquico del árbol. Además, se eligió un diseño modular, con
funciones internas recursivas, lo que permitió una implementación clara y reutilizable. **Validación del funcionamiento**

El árbol fue validado mediante:

• Pruebas de inserción con múltiples valores enteros.

• Búsqueda del valor 60, que arrojó el resultado esperado.

• Recorridos inorden, preorden y postorden, cuyos resultados confirmaron la correcta estructura del árbol.

• Eliminación del nodo 30, caso en el que se comprobó que el árbol ajusta correctamente su estructura al reemplazar el nodo eliminado con su sucesor inorden (nodo 40 en este caso).

Estas pruebas confirman que la implementación funciona según lo esperado y que los conceptos teóricos fueron aplicados correctamente en la práctica.

4. Metodología Utilizada

Para el desarrollo de este trabajo se siguió una metodología estructurada, basada en la combinación de investigación teórica y desarrollo práctico. A continuación, se detallan los pasos seguidos:

1. Investigación previa

Se inició el trabajo con una etapa de recopilación y análisis de información sobre estructuras de datos, haciendo especial énfasis en los árboles y sus distintas variantes. Las fuentes
utilizadas incluyeron material bibliográfico de cursos de algoritmos y estructuras de datos, documentación oficial de Python y recursos educativos en línea. También se consultaron artículos y apuntes provistos por los docentes.

2. Diseño y prueba del código

Una vez comprendidos los conceptos teóricos, se procedió al diseño de un árbol binario de búsqueda utilizando el lenguaje Python. La implementación incluyó funcionalidades clave como inserción, búsqueda, eliminación y distintos recorridos (inorden, preorden y postorden). El código se fue desarrollando de forma modular y probado de manera

incremental, verificando su correcto funcionamiento con conjuntos de datos concretos.

3. Herramientas y recursos utilizados

El entorno de desarrollo utilizado fue Visual Studio Code, por su compatibilidad con Python, autocompletado y facilidad de depuración. Se utilizó el intérprete de Python 3.11. Además, se recurrió a Jupyter Notebook para realizar pruebas interactivas. No se requirieron librerías externas, ya que el foco estuvo en comprender el funcionamiento interno de las estructuras. Para control de versiones y respaldo del trabajo, se usó GitHub, permitiendo mantener un historial organizado del proyecto.

4. Trabajo colaborativo

El trabajo fue realizado en grupo, dividiéndose las tareas según las fortalezas de cada integrante. Algunos se enfocaron en la investigación teórica y redacción del marco conceptual, mientras que otros se dedicaron al diseño, programación y validación del código. Se trabajó de manera colaborativa mediante reuniones virtuales y el uso de herramientas como Google Docs y GitHub, lo cual permitió mantener una comunicación fluida y una integración efectiva de las partes del trabajo.

Esta metodología permitió alcanzar un desarrollo sólido y comprensivo del tema, asegurando la calidad tanto del contenido teórico como de la aplicación práctica.

5. Resultados Obtenidos

A través de la implementación práctica de un árbol binario de búsqueda (ABB) en Python, se lograron observar los fundamentos teóricos aplicados a una estructura concreta y funcional. Se construyó un ABB con los siguientes valores: 50, 30, 70, 20, 40, 60, 80, 10, respetando las reglas de ordenamiento que esta estructura impone: todos los nodos del subárbol izquierdo son menores que el nodo raíz, y los del subárbol derecho son mayores.
Se realizaron tres tipos de recorridos sobre el árbol resultante:

• Recorrido inorden: devolvió los valores ordenados ascendentemente ([10, 20, 30, 40,

50, 60, 70, 80]), confirmando que el árbol fue construido correctamente.

• Recorrido preorden: evidenció el proceso de construcción del árbol desde la raíz hacia los hijos ([50, 30, 20, 10, 40, 70, 60, 80]).

• Recorrido postorden: mostró cómo se recorren primero los subárboles antes de procesar el nodo principal ([10, 20, 40, 30, 60, 80, 70, 50]).

También se evaluó la funcionalidad de búsqueda, comprobando la presencia del valor 60 en el árbol con éxito.

Finalmente, se probó la eliminación de un nodo con dos hijos (el nodo 30). Tras esta operación, el recorrido inorden resultante fue [10, 20, 40, 50, 60, 70, 80], lo cual demuestra que la eliminación fue realizada correctamente, manteniendo la propiedad estructural del ABB. El nodo 30 fue reemplazado por su sucesor inorden (el menor valor del subárbol derecho), en este caso el nodo 40.

En conclusión, el código permitió validar de forma empírica el funcionamiento de las operaciones básicas sobre árboles binarios de búsqueda: inserción, búsqueda, recorrido y eliminación, reflejando fielmente los conceptos teóricos aprendidos.

6. Conclusiones

La realización de este trabajo permitió comprender en profundidad cómo funcionan los árboles como estructuras de datos jerárquicas, y su ventaja frente a otras formas de almacenamiento lineal.

A través del desarrollo práctico, se pudo observar cómo operaciones como la búsqueda o la eliminación se optimizan considerablemente gracias a la organización interna del árbol.

Además, se reforzaron conceptos clave como la recursividad, la organización por nodos, y la toma de decisiones en función de comparaciones entre valores.

Se identificó como área de mejora la implementación de árboles balanceados o AVL, que podrían garantizar un mejor rendimiento ante casos de inserciones desbalanceadas.

7. Bibliografía

Material de estudio: Arboles.docx

8. Anexos

OpenAI. (2025). Imagenes generadas por ChatGPT sobre recorridos de árboles binarios. ChatGPT.