Árboles binarios de búsqueda

Institución:

- Universidad Tecnológica Nacional

Carrera:

- Tecnicatura Universitaria en Programación a distancia (TUP)

Materia:

- Programación I

Docente:

- Nicolás Quirós

Integrantes:

- Elian Rivoira - rivoiraelian@gmail.com
- Juan Francisco Reales - juanfrareales14@gmail.com

Fecha de entrega:

- 9/6/2025

1. Introducción

El presente trabajo integrador se enmarca en la asignatura **Programación I** de la **Tecnicatura Universitaria en Programación**, y tiene como objetivo aplicar de forma práctica los contenidos teóricos estudiados durante el cuatrimestre, profundizando en estructuras de datos avanzadas utilizando el lenguaje Python.

En particular, se decidió trabajar con **árboles binarios de búsqueda (ABB)**, una estructura jerárquica ampliamente utilizada en informática por su eficiencia en operaciones como búsqueda, inserción y recorrido ordenado de datos. El eje central del trabajo fue desarrollar un caso práctico funcional que represente un problema real, aplicando esta estructura para gestionar una **agenda de contactos**.

A través de esta experiencia buscamos no solo reforzar los contenidos académicos, sino también simular un entorno de desarrollo real en el que teoría y práctica se articulan para dar respuesta a una necesidad concreta mediante software.

---

2. Marco teórico

🔷 ¿Qué es un árbol?

Un **árbol** es una estructura de datos no lineal jerárquica, compuesta por nodos conectados entre sí. Un árbol tiene una raíz (nodo inicial) y cada nodo puede tener **hijos**, formando una estructura tipo grafo acíclico dirigido.

Terminología básica:

- **Nodo**: unidad fundamental del árbol que contiene un valor o dato.
- **Raíz (root)**: el nodo principal desde donde parte el árbol.
- **Hojas (leaves)**: nodos que no tienen hijos.
- **Padre e hijo**: relación entre nodos conectados jerárquicamente.
- **Subárbol**: cualquier rama que parte de un nodo.

🔷 Tipos de árboles comunes

- **Árbol Binario**: cada nodo tiene como máximo dos hijos.
- **Árbol Binario de Búsqueda (ABB o BST - Binary Search Tree)**.
- **Árbol AVL**: BST autobalanceado.
- **Árbol B**: usado en bases de datos, permite múltiples hijos.
- **Heap**: árbol que cumple la propiedad de ordenamiento (min/max).

🔷 Árbol Binario de Búsqueda (ABB)

Un **árbol binario de búsqueda** es un tipo especial de árbol binario en el que se cumple una propiedad clave:

Para cada nodo, todos los valores del subárbol izquierdo son menores, y todos los del subárbol derecho son mayores.

Esto permite una **búsqueda eficiente**, ya que podemos descartar la mitad del árbol en cada paso (similar a la búsqueda binaria).

Operaciones básicas:

1. **Inserción**: se compara el valor a insertar con los nodos, recorriendo hacia la izquierda o derecha hasta encontrar una posición vacía.
2. **Búsqueda**: se sigue el mismo camino que la inserción, comparando el valor deseado.
3. **Recorridos**:
    - **Inorden** (izquierda → nodo → derecha): da los elementos ordenados alfabéticamente.
    - **Preorden** (nodo → izquierda → derecha). Visitamos: primero la **raíz**, luego el subárbol **izquierdo**, después el subárbol **derecho.**
    - **Postorden** (izquierda → derecha → nodo). Visitamos: primero el subárbol **izquierdo**, luego el **derecho**, finalmente la **raíz**
4. **Eliminación**:
    - Caso 1: el nodo es hoja. Simplemente se elimina el nodo
    - Caso 2: el nodo tiene un solo hijo. El hijo reemplaza al nodo eliminado
    - Caso 3: el nodo tiene dos hijos.

      El PROGRAMADOR decide la estrategia:

i. **Opción A**: Usar el **sucesor inorden** (mínimo del subárbol derecho)
ii. **Opción B**: Usar el **predecesor inorden** (máximo del subárbol izquierdo)

🔷 Ventajas de los ABB

- **Búsqueda, inserción y eliminación** en **O(log n)** en promedio.
- Organización de datos jerárquica y ordenada.
- Fácil de recorrer para mostrar resultados ordenados.

🔷 Desventajas

- En su forma básica **no se auto balancea**. Si los datos llegan en orden creciente o decreciente, se convierte en una **lista enlazada**, perdiendo eficiencia (O(n)).
- Para mantener el rendimiento óptimo, pueden usarse versiones autobalanceadas como AVL o Red-Black Trees.

🔷 Aplicaciones reales

- Implementación de diccionarios y árboles de expresión.
- Motores de bases de datos (índices y árboles B/B+).
- Compiladores (análisis sintáctico).
- Inteligencia artificial (árboles de decisión).
- Sistemas de archivos (como en Linux o NTFS).

---

3. Caso Práctico

Se diseñó una agenda digital donde los contactos se insertan en un ABB según su nombre. Esta agenda debe permitir almacenar miles de contactos y brindar funcionalidades como agregar, buscar y listar contactos por orden alfabético. Para lograrlo de manera eficiente, un ABB es ideal:

- Inserta automáticamente los contactos en orden.
- Permite búsquedas rápidas sin recorrer todos los datos.

- Genera una lista ordenada sin requerir ordenamiento adicional.

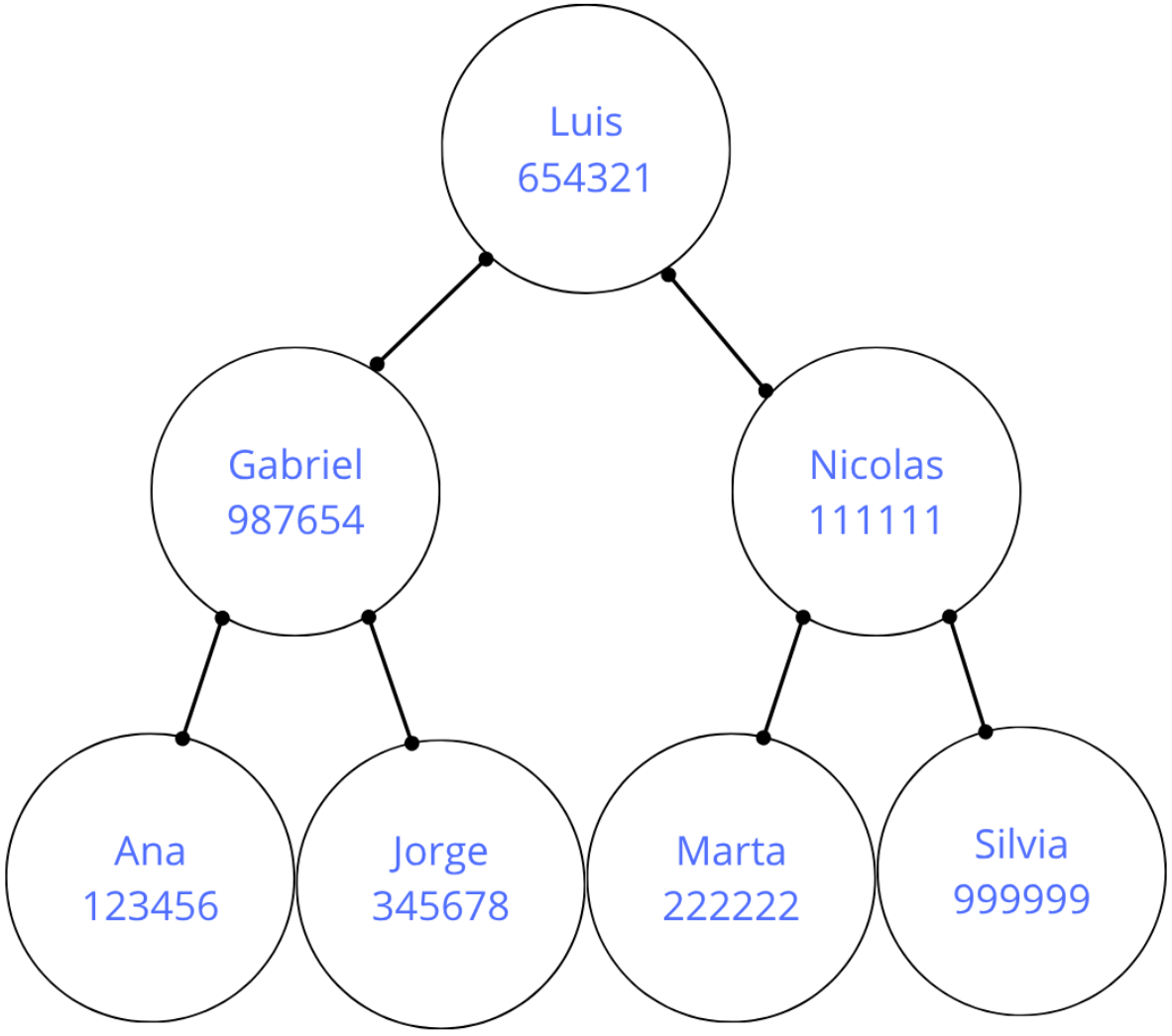
Este tipo de estructura optimiza el rendimiento y escalabilidad del sistema.

La estructura contiene las siguientes clases:

- ***Contact***: representa un contacto (nombre y teléfono).
- ***Node***: nodo del árbol que almacena un contacto.
- ***ContactTree***: contiene la lógica del ABB (insertar, buscar, recorrer).

Se agregaron los siguientes contactos para probar el sistema:

- Ana (123456)
- Luis (654321)
- Nicolas (111111)
- Marta (222222)
- Jorge (345678)
- Gabriel(987654)
- Silvia (999999)



4. Conclusiones

- El ABB permite almacenar y organizar datos de forma jerárquica y eficiente.
- Aplicar esta estructura en una agenda digital permite búsquedas e inserciones rápidas.
- La estructura puede extenderse con funcionalidades como eliminación, persistencia y GUI.

Este trabajo nos permitió comprender en profundidad la importancia de las estructuras de datos avanzadas, en particular el Árbol Binario de Búsqueda (ABB), como herramienta fundamental para organizar y acceder eficientemente a la información.

Aprendizajes clave

- Comprendimos cómo aplicar árboles en contextos reales.
- Reforzamos programación orientada a objetos y ordenamiento.
- Exploramos cómo organizar datos de manera escalable.

Desafíos y soluciones

- Diseñamos la lógica del árbol para mantener el orden.
- Aplicamos recursividad para insertar y recorrer.
- Usamos Git para trabajo colaborativo.

Link video explicativo

- https://www.youtube.com/watch?v=8mXYOfO2xXE

📚 Bibliografía y fuentes

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Python Software Foundation. (2024). https://docs.python.org/3/tutorial/classes.html
- Geeks for Geeks. Árboles binarios: https://www.geeksforgeeks.org/binary-tree-data-structure/ (Ultimo acceso: 3/6/2025)