Trabajo Práctico

Investigación aplicada en Python

Estructuras de datos avanzadas: árboles

Alumnos:
Rocío Santarelli – santarellirocio@gmail.com Comisión 9
Yoel Santarelli – santarelli114@gmail.com Comisión 21

Materia: Programación I
Profesor: AUS Bruselario, Sebastián; Nicolás Quirós
Tutores: Gubiotti, Florencia; Neyén Bianchi
Fecha de Entrega: 09 de junio de 2025

Índice

1. Introducción

En la programación y la informática, existen muchas formas de organizar los datos. Una de ellas son las estructuras de datos avanzadas, que permiten guardar y acceder a la información de manera más eficiente. En este trabajo nos centraremos en una estructura llamada árbol, que se puede usar para representar jerarquías. Aunque normalmente se implementan con clases o nodos, en este caso aprenderemos cómo hacerlo de forma más simple utilizando listas en Python.

2. Objetivo General:

Llevar adelante una investigación práctica y aplicada sobre conceptos fundamentales y avanzados del lenguaje Python, integrando teoría, casos de uso reales, desarrollo de software y difusión de los resultados obtenidos.

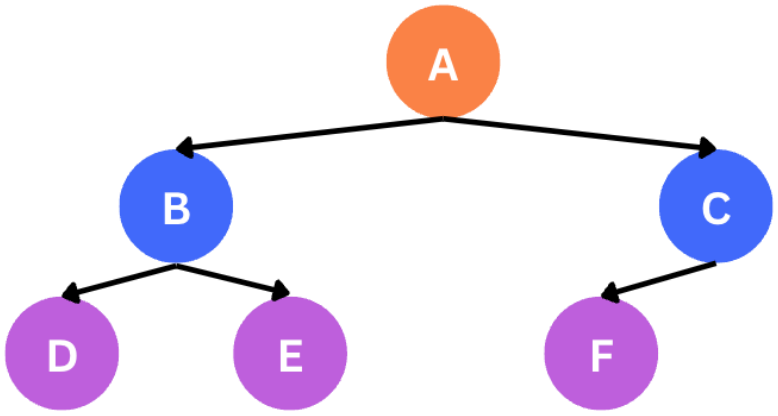
2.1. Objetivos del trabajo práctico

Con el desarrollo de este trabajo buscamos:

- Comprender qué es un árbol como estructura de datos.
- Desarrollar un programa interactivo en Python que permita al usuario construir un árbol genealógico utilizando la estructura de datos de árbol binario.
- Desarrollar habilidades básicas de pensamiento lógico y programación.
- Que el programa permita la incorporación de miembros, búsqueda de nodos, y visualización jerárquica del árbol.

3. Marco Teórico

Un **árbol** es una estructura de datos jerárquica compuesta por **nodos**. Tiene un nodo principal llamado **raíz**, del cual pueden salir otros nodos llamados **hijos**. Cada nodo puede tener **subnodos** (más hijos), y así se forma una estructura en forma de árbol invertido.
Normalmente, los árboles se implementan usando clases y objetos en Python, pero también es posible representarlos con listas anidadas, que es una forma más accesible para quienes están empezando.
Por ejemplo, el siguiente árbol:



Se puede representar como:

["A", ["B", [“D” [ ], [ ]], [“E”, [ ],[ ]]], ["C", ["F", [], []], [ ]]]

Cada nodo tiene esta forma: [valor (llamado padre), [hijo_izquierdo], [hijo_derecho]].

En un árbol (como estructura de datos), cada elemento se llama nodo. Según dónde está ubicado y con quién se relaciona, un nodo puede tener distintos nombres.

1. ***Según su posición en el árbol***

Nodo raíz

Es el nodo principal, el que está arriba de todo. Es el punto de entrada al árbol y no tiene "padre". Ejemplo: A es la raíz.

Nodo hoja

Es cualquier nodo que no tiene hijos. Son los que están en las puntas del árbol. Ejemplo: D, E y F son hojas.

Nodo rama (o nodo interno)

Es un nodo que tiene padre y también hijos. Está en el medio del árbol, conectando niveles. Ejemplo: B y C son nodos rama.

2. ***Según su relación con otros nodos***

Nodo padre

Es el nodo que tiene hijos conectados a él. Ejemplo: A es padre de B y C. B es padre de D y E.

Nodo hijo

Es el que está debajo de un padre. Todos los nodos (menos la raíz) tienen un padre. Ejemplo: B y C son hijos de A. D y E son hijos de B.

Nodo hermano

Son nodos que tienen el mismo padre. Ejemplo: B y C son hermanos. D y E también son hermanos.

Resumen rápido en modo familia:

- El árbol es como una familia.
- El primer nodo es como un abuelo (la raíz).
- Los nodos con hijos son padres.
- Los que no tienen hijos son como niños pequeños (hojas).
- Y los que comparten al mismo padre son hermanos.

Árboles binarios

Es un árbol en el que cada nodo puede tener, como máximo, 2 hijos. Dicho de otra manera, es un árbol grado dos. En un árbol binario el nodo de la izquierda representa al hijo izquierdo y el nodo de la derecha representa al hijo derecho.

Tipos de recorridos de un árbol binario:

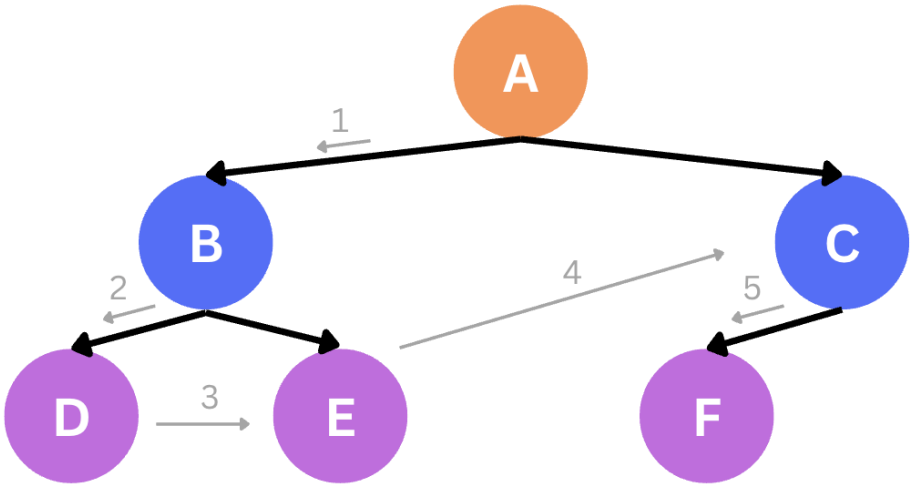
Cuando trabajamos con árboles binarios, es común recorrer sus nodos en cierto orden. Los tres recorridos principales son:

Preorden

El recorrido preorden es útil cuando necesitas realizar alguna operación en el nodo antes de procesar a sus hijos. Un caso típico es cuando estás copiando un árbol o clonando su estructura.

Para recorrer un árbol binario no vacío en preorden se ejecutan los siguientes pasos:

1. Se comienza por la raíz.
2. Se baja hacia el hijo izquierdo de la raíz.
3. Se recorre recursivamente el subárbol izquierdo.
4. Se sube hasta el hijo derecho de la raíz.
5. Se recorre recursivamente el subárbol derecho.



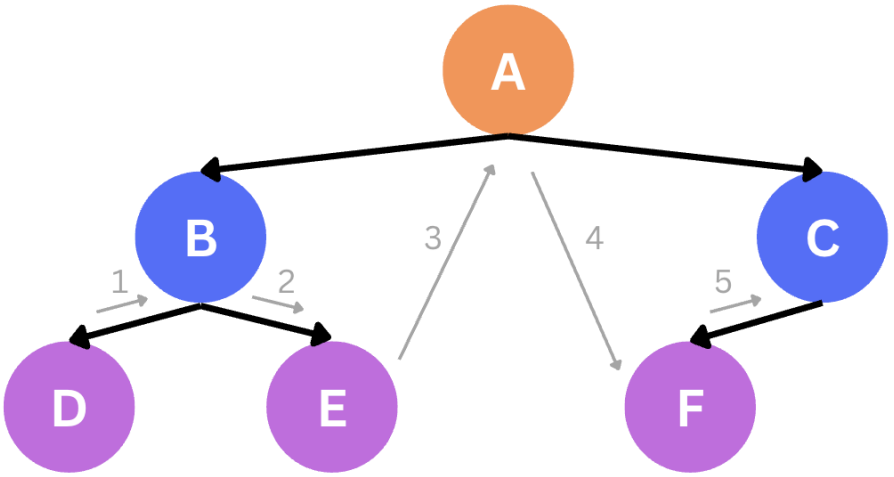
Inorden

El recorrido inorden es especialmente útil para los árboles de búsqueda binaria, ya que garantiza que los nodos se visiten en orden ascendente.

Para recorrer un árbol binario no vacío en inorden se ejecutan los siguientes pasos:

1. Se comienza por el nodo hoja que se encuentre más a la izquierda de todos.
2. Se sube hacia su nodo padre.
3. Se baja hacia el hijo derecho del nodo recorrido en el paso 2.

4. Se repiten los pasos 2 y 3 hasta terminar de recorrer el subárbol izquierdo.
5. Se visita el nodo raíz.
6. Se recorre el subárbol derecho de la misma manera que se recorrió el subárbol izquierdo.

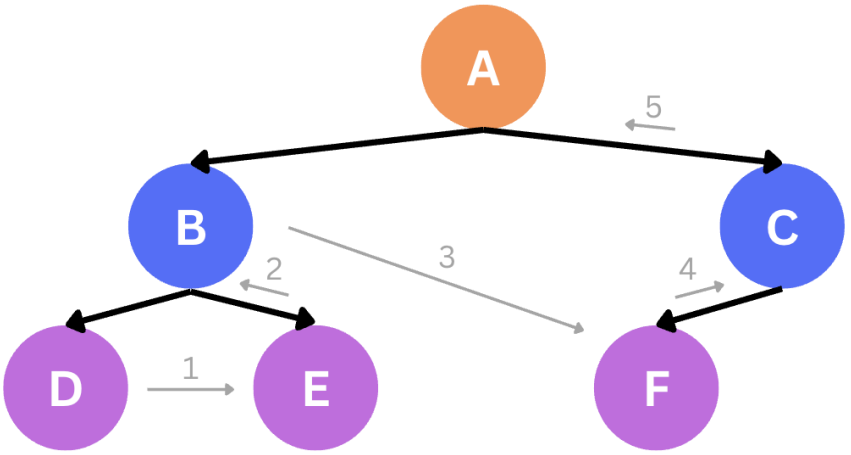


Postorden

El recorrido postorden es útil cuando quieres realizar alguna acción en los nodos hijos de un nodo antes de procesar el nodo mismo. Esto es común cuando se trabaja con estructuras jerárquicas donde necesitas procesar primero las subestructuras.

Para recorrer un árbol binario no vacío en postorden se ejecutan los siguientes pasos:

1. Se comienza por el nodo hoja que se encuentre más a la izquierda de todos.
2. Se visita su nodo hermano.
3. Se sube hacia el padre de ambos.
4. Si tuviera hermanos, se visita su nodo hermano.
5. Se repiten los pasos 3 y 4 hasta terminar de recorrer el subárbol izquierdo.
6. Se recorre el subárbol derecho, comenzando por el nodo hoja que se encuentre más a la izquierda y siguiendo el mismo procedimiento que con el subárbol izquierdo
7. Se visita el nodo raíz.



Un árbol puede servir para modelar estructuras tales como:

- Sistemas de archivos.
- Árboles genealógicos.
- Bases de datos jerárquicas.
- Algoritmos de búsqueda y toma de decisiones.
- Organización jerárquica de una empresa.
- Entre otros.

4. Caso Práctico

A continuación, se presenta el código completo para implementar un árbol binario:

```
# Definimos la clase Persona para representar a cada miembro del árbol genealógico

class Persona:

    def __init__(self, nombre):

        self.nombre = nombre # Nombre de la persona

        self.hijo_izquierdo = None # Referencia al primer hijo (izquierdo)

        self.hijo_derecho = None # Referencia al segundo hijo (derecho)



# Función para imprimir el árbol genealógico en forma jerárquica
```

```python
def imprimir_arbol(persona, nivel=0):
    if persona:
        # Imprime el nombre de la persona con indentación según su nivel
        print("  " * nivel + f"- {persona.nombre}")
      # Llama recursivamente para imprimir los hijos (izquierda y derecha)
        imprimir_arbol(persona.hijo_izquierdo, nivel + 1)
        imprimir_arbol(persona.hijo_derecho, nivel + 1)


def buscar_persona(raiz, nombre):
    if raiz is None:
        return None
    if raiz.nombre == nombre:
        return raiz
    izquierda = buscar_persona(raiz.hijo_izquierdo, nombre)
    if izquierda:
        return izquierda
    return buscar_persona(raiz.hijo_derecho, nombre)


# Se pide al usuario que ingrese el nombre del ancestro principal (raíz del árbol)
nombre_raiz = input("Ingrese el nombre del ancestro principal (raíz): ")
raiz = Persona(nombre_raiz)  # Se crea la raíz del árbol
```


```
while True:

    print("\nOpciones:")

    print("1. Agregar hijo")

    print("2. Mostrar árbol genealógico")

    print("3. Salir")



    opcion = input("Seleccione una opción: ")

    if opcion == "1":

        # Se ingresan los datos del padre/madre y del nuevo hijo

        padre_nombre = input("Ingrese el nombre del padre/madre: ")

        nuevo_nombre = input("Ingrese el nombre del nuevo hijo/hija: ")



        # Se busca la persona a quien se le quiere agregar un hijo

        padre = buscar_persona(raiz, padre_nombre)

        if padre is None:

                print("❌ Persona no encontrada.")  # No se encontró a esa
persona en el árbol

        else:

            # Si se encuentra, se crea el nuevo hijo

            nuevo_hijo = Persona(nuevo_nombre)

            if padre.hijo_izquierdo is None:

                  padre.hijo_izquierdo = nuevo_hijo  # Se agrega como hijo
izquierdo

                           print(f"✅ Hijo agregado a la izquierda de
{padre.nombre}.")
```

```
            elif padre.hijo_derecho is None:
                padre.hijo_derecho = nuevo_hijo  # Se agrega como hijo derecho
                print(f"✅ Hijo agregado a la derecha de {padre.nombre}.")
            else:
                print("⚠️ Esta persona ya tiene dos hijos.")  # Ya tiene el máximo de hijos permitidos
    elif opcion == "2":
        # Muestra el árbol actual con todos sus miembros
        print("\nÁrbol Genealógico:")
        imprimir_arbol(raiz)

    elif opcion == "3":
        # Finaliza el programa
        print("👋 Saliendo del programa.")
        break
    else:
        # Opción inválida del menú
        print("❌ Opción no válida.")
```

5. Metodología Utilizada

Para el desarrollo de este trabajo se siguieron los siguientes pasos:

- Se investigó cómo funcionan los árboles binarios y cómo representarlos con listas en Python.
- Buscamos información de en qué contexto se puede implementar árboles para aplicarlo en un caso práctico de la realidad.

- Se escribió un programa en Python para construir un árbol binario basándonos en un árbol genealógico.
- Se ejecutó el adecuado funcionamiento del mismo.

6. Resultados Obtenidos

El sistema permite:

- Registrar un ancestro raíz (por ejemplo: abuelo).
- Agregar descendientes de forma binaria (máximo dos hijos por persona).
- Imprimir el árbol genealógico con formato jerárquico.
- Prevenir el exceso de hijos (más de 2) por persona.
- Interactuar con el usuario mediante un menú claro y funcional.

Ventajas:

- Implementación clara y escalable.
- Interfaz simple para el usuario.
- Uso de conceptos clave como árboles, búsqueda y recursividad.

Desventajas:

- Solo permite dos hijos por persona, lo que no representa correctamente familias con más hijos.
- El programa no guarda los datos al cerrar
- No se controla que los nombres ingresados estén duplicados.

7. Conclusiones

Pudimos llegar a la conclusión de que los árboles son estructuras muy importantes que aparecen en muchos ámbitos de la informática, como en los sistemas de archivos o en algoritmos de búsqueda y en cuanto al trabajo práctico se logró construir un árbol genealógico binario en Python usando programación orientada a objetos, donde la implementación permite comprender conceptos fundamentales como estructuras de datos, clases, recursividad y manejo de entrada del usuario aunque su utilidad en aplicaciones reales es limitada debido a la restricción de solo dos hijos por nodo.

Bibliografía

- Programiz. “Binary Tree in Python.”: https://www.programiz.com/dsa/binary-tree

- W3Schools. “Python Lists.”: https://www.w3schools.com/python/python_lists.asp
- Geeks for Geeks. “Tree Data Structure.”: https://www.geeksforgeeks.org/binary-tree-data-structure/
- Documento de “Árboles” proporcionado por la UTN.

8. Anexos

- Capturas de pantalla del código funcionando en consola:

```
Ingrese el nombre del ancestro principal (raíz): abuelo

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: abuelo
Opciones:                              padre
Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: abuelo
Ingrese el nombre del nuevo hijo/hija: tio
✅ Hijo agregado a la derecha de abuelo.

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 2

Árbol Genealógico:
- abuelo
  - padre
  - tio

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: padre
Ingrese el nombre del nuevo hijo/hija: yo
✅ Hijo agregado a la izquierda de padre.

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: padre
Ingrese el nombre del nuevo hijo/hija: hermano
✅ Hijo agregado a la derecha de padre.

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: tio
Ingrese el nombre del nuevo hijo/hija: primo
✅ Hijo agregado a la izquierda de tio.

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 2

Árbol Genealógico:
- abuelo
  - padre
    - yo
    - hermano
  - tio
    - primo
```


```
Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: padre
Ingrese el nombre del nuevo hijo/hija: hermano
⚠️Esta persona ya tiene dos hijos.

Opciones:
1. Agregar hijo
2. Mostrar árbol genealógico
3. Salir
Seleccione una opción: 1
Ingrese el nombre del padre/madre: madre
Ingrese el nombre del nuevo hijo/hija: hermano
❌ Persona no encontrada.
```

- Repositorio en GitHub: https://github.com/GYoelSantarelli/TP-Integrador-P1.git
- Video explicativo mostrando la estructura del árbol, los recorridos y reflexión final: