Trabajo Práctico Integrador: Programación I

Datos Avanzados: árboles y grados

Árbol binario con clases y recorridos en preorden, inorden y postorden

Alumnas

Alterio Micaela, Ambrosio Laura

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación I

Docente Titular

Cinthia Rigoni

Docente Tutor

Martín A. García

09 de Junio de 2025

Tabla de contenido

1. Introducción

El propósito de este trabajo es construir un árbol binario que represente relaciones genealógicas, utilizando clases en Python. A partir de la estructura creada, se implementan los tres tipos clásicos de recorrido de árboles: preorden, inorden y postorden.

Los árboles son estructuras jerárquicas ideales para representar relaciones entre elementos con un origen común, como los miembros de una familia. Los árboles binarios permiten modelar los vínculos entre una persona y sus dos progenitores de forma ordenada, facilitando la visualización, análisis y recorrido de la información desde distintos puntos de vista. Usar clases brinda flexibilidad para recorrer el árbol desde cualquier persona (nodo raíz).

2. Objetivo del trabajo

- ➢ Implementar un árbol binario para representar una genealogía enfocada en los progenitores y así comprender el proceso de creación de árboles binarios
- ➢ Explorar las tres formas para recorrer un árbol binario
- ➢ Afianzar el uso de clases y recursividad en python

3. Marco Teórico

¿Qué es un árbol binario?

Un árbol binario es una estructura de datos jerárquica compuesta por nodos, donde cada nodo tiene a lo sumo dos nodos hijos, comúnmente denominados hijo izquierdo e hijo derecho. Esta estructura es especialmente útil para representar relaciones jerárquicas o bifurcaciones lógicas. El nodo superior se denomina raíz, y los nodos sin hijos se denominan hojas. Cada nodo puede contener información (por ejemplo, un nombre) y referencias a sus progenitores o descendientes, dependiendo de la aplicación.

Propiedades clave de los árboles binarios:

• Raíz: nodo inicial.

• Hojas: nodos sin hijos.

• Altura: número de niveles desde la raíz hasta la hoja más profunda.

• Subárboles: cualquier nodo junto a sus descendientes forma un subárbol.

¿Cómo se representa un árbol binario en Python?

Existen varias formas de representar un árbol binario en Python, siendo las más comunes:

- **Con listas de hijos:** Cada nodo se representa como una lista donde el primer elemento es el dato y los siguientes elementos son sus hijos. Es útil para árboles más generales o con más de dos hijos.

- **Con clases:** Es la forma más estructurada y flexible. Se define una clase Nodo (o Persona, como en este caso) con atributos para almacenar el valor y referencias a sus hijos. Esta técnica facilita la implementación de métodos como recorridos (preorden, inorden, postorden) y la visualización del árbol.

En el caso práctico presentado, se utiliza una clase Persona que modela a cada nodo. La clase contiene un atributo progenitores (una lista de nodos padres) y métodos para agregar progenitores, imprimir el árbol e implementar recorridos clásicos de árboles binarios.

Comparación con árboles no binarios

Un **árbol no binario** permite que cada nodo tenga más de dos hijos, lo cual es común en aplicaciones como menús de navegación. Aunque más flexibles, los árboles no binarios requieren estructuras de datos más complejas para su manejo.

En cambio, los **árboles binarios** están limitados a dos hijos por nodo, lo que permite simplificar algoritmos de recorrido y estructuras de representación. Esto los hace ideales para situaciones donde las relaciones padre-hijo son estrictamente binarias, como árboles genealógicos ascendentes, árboles de decisión o árboles de expresión matemática.

Aplicaciones reales

Los árboles binarios tienen múltiples aplicaciones en el mundo real, entre ellas:

- Genealogía: Como en este caso, donde cada persona tiene hasta dos progenitores, el árbol binario permite representar de forma clara la ascendencia de un individuo.

- Estructuras jerárquicas: Se utilizan en organizaciones, empresas o sistemas operativos para representar estructuras de mando o archivos.

- Algoritmos y estructuras de datos: Los árboles binarios son la base de estructuras como árboles de búsqueda binaria (BST), heaps, árboles AVL, entre otros.

- Sistemas expertos y lógica: Se usan en árboles de decisión o árboles de juego para modelar elecciones y consecuencias.

- Compresión de datos: Como en el algoritmo de Huffman, que utiliza árboles binarios para codificar datos de forma eficiente.

4. Desarrollo/Implementación

```
#Árbol genealógico binario de progenitores

#La raíz es la persona de menor edad, las ramas llevan a sus padres

#Se utilizó una lista para guardar la información de los nodos

#Usamos clases para poder obtener el orden dependiendo del nodo (cambiar el root)


class Persona:

    def __init__(self, nombre):

        self.nombre = nombre

        self.progenitores = []


    def agregar_progenitor(self, progenitor):

        self.progenitores.append(progenitor)


    def imprimir_arbol(self, nivel=0):

        print("  " * nivel + self.nombre)

        for progenitores in self.progenitores:

            progenitores.imprimir_arbol(nivel + 1)


    def preorden(self):

        resultado = [self.nombre]

        for p in self.progenitores:
```

```
            resultado.extend(p.preorden())
        return resultado


    def inorden(self):
        resultado = []
        if len(self.progenitores) > 0:
            resultado.extend(self.progenitores[0].inorden())
        resultado.append(self.nombre)
        if len(self.progenitores) > 1:
            resultado.extend(self.progenitores[1].inorden())
        return resultado


    def postorden(self):
        resultado = []
        for p in self.progenitores:
                resultado.extend(p.postorden()) #llama hasta que encuentra al que no tiene progenitor
        resultado.append(self.nombre)
        return resultado


#Crear personas/nodos
ana = Persona("Ana")
berta = Persona("Berta")
```

```
carlos = Persona("Carlos")

denis = Persona("Denís")

esteban= Persona("Esteban")

florencia = Persona("Florencia")

gaston= Persona("Gastón")

horacio=Persona("Horacio")

ileana=Persona("Ileana")



# Definir relaciones entre nodos/crear relacion hijos-progenitores

ana.agregar_progenitor(berta)

ana.agregar_progenitor(carlos)

berta.agregar_progenitor(denis)

berta.agregar_progenitor(esteban)

carlos.agregar_progenitor(florencia)

carlos.agregar_progenitor(gaston)

denis.agregar_progenitor(horacio)

gaston.agregar_progenitor(ileana)



#Mostrar arbol (root: Ana)

print("""

Arbol de progenitores con Ana como root

Las indentaciones representan el nivel del nodo

""")
```

```
ana.imprimir_arbol()



print("""

Formas de recorrer árboles binario

""")



print("Preorden:", ana.preorden())

print("Inorden:", ana.inorden())

print("Postorden:", ana.postorden())
```

6. Resultados

```
Arbol de progenitores con Ana como root
Las indentaciones representan el nivel del nodo

Ana
  Berta
    Denís
      Horacio
    Esteban
  Carlos
    Florencia
    Gastón
      Ileana

Formas de recorrer árboles binario

Preorden: ['Ana', 'Berta', 'Denís', 'Horacio', 'Esteban', 'Carlos', 'Florencia', 'Gastón', 'Ileana']
Inorden: ['Horacio', 'Denís', 'Berta', 'Esteban', 'Ana', 'Florencia', 'Carlos', 'Ileana', 'Gastón']
Postorden: ['Horacio', 'Denís', 'Esteban', 'Berta', 'Florencia', 'Ileana', 'Gastón', 'Carlos', 'Ana']
```

El árbol se construyó correctamente usando instancias de la clase Persona, reflejando una estructura genealógica binaria donde cada persona tiene hasta dos progenitores.

El método imprimir_arbol muestra claramente la jerarquía genealógica con indentaciones proporcionales al nivel del nodo.

Los métodos de recorrido (preorden, inorden, postorden) funcionan de manera adecuada y devuelven las listas de nombres en el orden correspondiente:

- ➢ Preorden visita primero el nodo actual, luego los progenitores.
- ➢ Inorden sigue el patrón izquierdo - raíz - derecho (útil en árboles de búsqueda).
- ➢ Postorden primero visita los progenitores y luego el nodo actual, ideal para análisis estructurales como eliminación o evaluación.

7. Conclusiones

- Comprendimos el funcionamiento interno de los **árboles binarios**, en especial su representación mediante **clases en Python**.

- También aprendimos los distintos tipos de **recorridos** y cómo aplicarlos en estructuras jerárquicas.

- Visualizamos cómo modelar relaciones reales (como un árbol genealógico) usando programación orientada a objetos y estructuras de datos.

- Una dificultad inicial que tuvimos fue decidir cómo representar a los progenitores: como atributos directos (izquierdo y derecho) o como una lista. Optamos por una lista para permitir una estructura más dinámica y escalable.

- También se nos presentó el reto de implementar los métodos de recorrido de forma correcta cuando los nodos no siempre tienen dos progenitores.

Posibilidades de mejora o ampliación

- Incluir más atributos en los nodos: Como fecha de nacimiento, lugar, profesión, etc., enriqueciendo la información de cada persona.

- Incorporar búsqueda de ancestros: Implementar funciones para buscar un nodo por nombre y listar todos sus ancestros.

- Validación de relaciones: Asegurar que no se ingresen más de dos progenitores por persona y evitar ciclos.

Enlace al video:

https://youtu.be/1pi9g78Ue1o