Trabajo Práctico integrador: Programación 1

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: Bruselario, Sebastián; Nicolás Quirós

Tutores: Gubiotti, Florencia; Neyén Bianchi

Fecha de Entrega: 20 de junio de 2025

Índice **Páginas**

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:

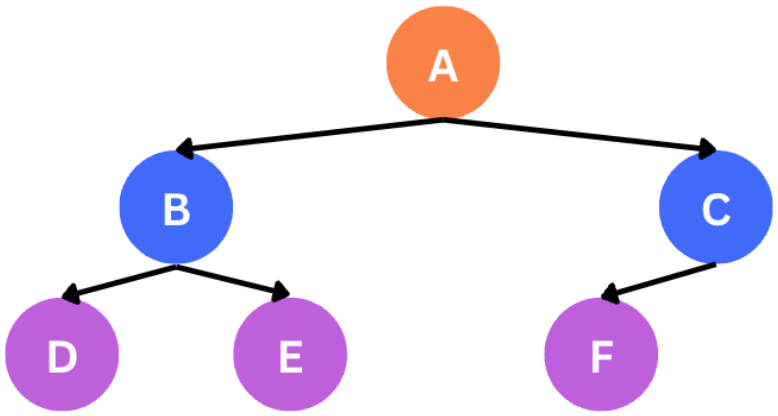
- Comprender qué es un árbol como estructura de datos.
- Desarrollar un programa interactivo en Python que permita al usuario construir un árbol genealógico utilizando la estructura de datos de árbol binario.
- Desarrollar habilidades básicas de pensamiento lógico y programación.
- Que el programa permita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

Operaciones que se pueden hacer en un árbol binario:

- Inserción
- Eliminación
- Localización
- Recorrido

Para nuestro caso práctico nos centraremos en la operación del tipo recorrido.

Tipos de recorridos de un árbol binario:

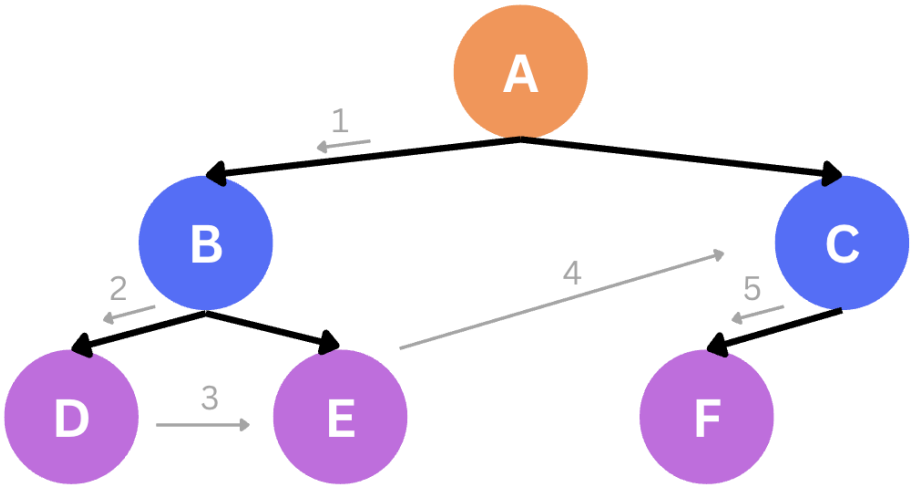
Cuando trabajamos con árboles binarios, es común recorrer sus nodos en cierto orden. Los tres recorridos principales son:

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.

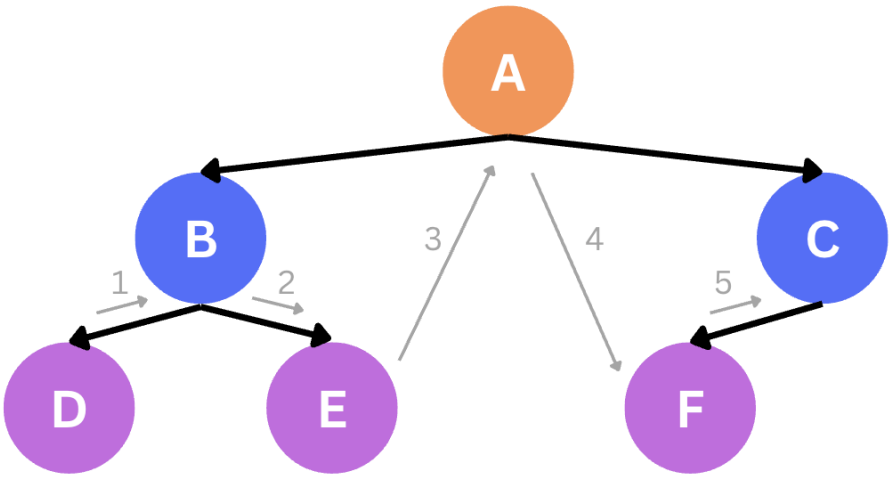


Inorden

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.

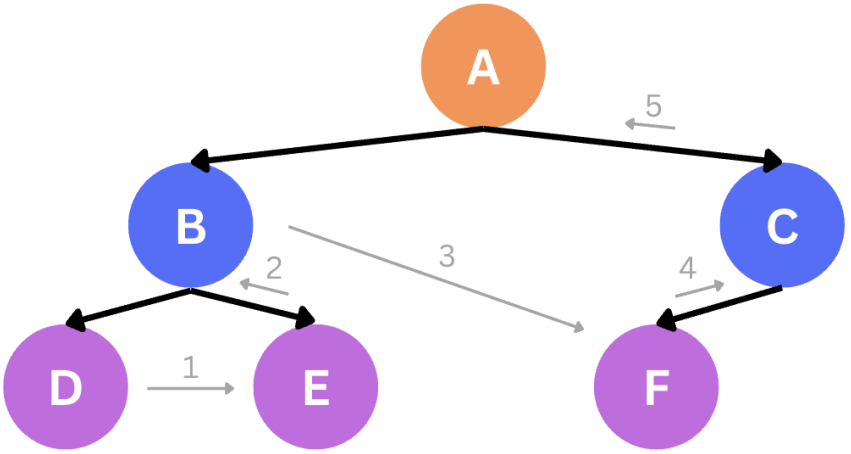


Postorden

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.

Recursividad en estructuras de datos

La recursividad es una técnica de programación en la que una función se llama a sí misma para resolver un problema más grande dividiéndolo en subproblemas más pequeños. Es especialmente útil en estructuras jerárquicas como los árboles, ya que permite recorrer nodos de manera natural y ordenada.

En un árbol binario, cada nodo tiene subárboles (izquierdo y derecho), que a su vez también pueden tener sus propios subárboles. Esto hace que la recursión sea una forma eficiente y clara de recorrer o buscar información dentro del árbol.

4. Caso Práctico

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

```
  #Función que crea una persona representada como un diccionario
def crear_persona(nombre):
    return {
"nombre": nombre.lower(),#Guarda el nombre en minúsculas para mantener uniformidad
"hijo_izquierdo": None,    #Inicialmente no tiene hijo izquierdo
"hijo_derecho": None       #Inicialmente no tiene hijo derecho
    }
   #Función recursiva para imprimir el árbol genealógico de manera jerárquica
def imprimir_arbol(persona, nivel=0):
    if persona:
     #Imprime el nombre con una indentación proporcional al nivel en el árbol
        print("  " * nivel + f"- {persona['nombre']}")
     #Llama recursivamente para imprimir el hijo izquierdo
        imprimir_arbol(persona["hijo_izquierdo"], nivel + 1)
     #Llama recursivamente para imprimir el hijo derecho
        imprimir_arbol(persona["hijo_derecho"], nivel + 1)
```

```
    #Función para buscar a una persona en el árbol por su nombre
def buscar_persona(raiz, nombre):
    if raiz is None:
            return None   #Si el nodo actual está vacío, termina búsqueda
    if raiz["nombre"] == nombre.lower():
          return raiz  #Si el nombre coincide (en minúsculas), se devuelve
    #Busca recursivamente en el hijo izquierdo
    izquierda = buscar_persona(raiz["hijo_izquierdo"], nombre)
    if izquierda:
      return izquierda   #Si se encuentra en la izquierda, se devuelve
    #Si no se encontró en la izquierda, busca en la derecha
    return buscar_persona(raiz["hijo_derecho"], nombre)


    #Recorrido INORDEN: primero el hijo izquierdo, luego la persona, luego el derecho
def recorrido_inorden(persona):
    if persona:
        recorrido_inorden(persona["hijo_izquierdo"])
        print(persona["nombre"])  #Visita el nodo actual
```

```
        recorrido_inorden(persona["hijo_derecho"])


    #Recorrido PREORDEN: primero la persona, luego izquierda, luego derecha
def recorrido_preorden(persona):
    if persona:
        print(persona["nombre"])  #Visita el nodo actual
        recorrido_preorden(persona["hijo_izquierdo"])
        recorrido_preorden(persona["hijo_derecho"])


    #Recorrido POSTORDEN: primero los hijos, luego la persona
def recorrido_postorden(persona):
    if persona:
        recorrido_postorden(persona["hijo_izquierdo"])
        recorrido_postorden(persona["hijo_derecho"])
        print(persona["nombre"])  #Visita el nodo actual
    #Solicita al usuario el nombre del ancestro principal del árbol
nombre_raiz = input("Ingrese el nombre del ancestro principal (raíz): ")
raiz = crear_persona(nombre_raiz)  #Crea la raíz del árbol
```

```
    #Bucle principal del programa con menú interactivo
while True:
    #Muestra las opciones disponibles
    print("\nOpciones:")
    print("1. Agregar hijo")
    print("2. Mostrar árbol genealógico")
    print("3. Recorrido INORDEN")
    print("4. Recorrido PREORDEN")
    print("5. Recorrido POSTORDEN")
    print("6. Salir")

    #Lee la opción elegida por el usuario
    opcion = input("Seleccione una opción: ")

    if opcion == "1":
     #Solicita los nombres del padre/madre y del nuevo hijo/hija
  padre_nombre = input("Ingrese el nombre del padre/madre: ")
   nuevo_nombre = input("Ingrese el nombre del nuevo hijo/hija:
")

      #Busca a la persona correspondiente al padre/madre
```

```
  padre = buscar_persona(raiz, padre_nombre)

   if padre is None:

  print("Persona no encontrada.") #Si no existe, muestra mensaje
de error

   else:

  nuevo_hijo = crear_persona(nuevo_nombre) #Crea el nuevo hijo

      #Intenta agregarlo como hijo izquierdo si está vacío

    if padre["hijo_izquierdo"] is None:

    padre["hijo_izquierdo"] = nuevo_hijo

    print(f"Hijo agregado a la izquierda de {padre['nombre']}.")

      #Si no, intenta como hijo derecho

    elif padre["hijo_derecho"] is None:

    padre["hijo_derecho"] = nuevo_hijo

    print(f"Hijo agregado a la derecha de {padre['nombre']}.")

      #Si ya tiene dos hijos, no se puede agregar más

    else:

     print("Esta persona ya tiene dos hijos.")


    elif opcion == "2":

      #Imprime el árbol completo en formato jerárquico

        print("\nÁrbol Genealógico:")
```

```
        imprimir_arbol(raiz)
    elif opcion == "3":
      #Muestra los nombres en recorrido INORDEN
        print("\nRecorrido INORDEN:")
        recorrido_inorden(raiz)
    elif opcion == "4":
      #Muestra los nombres en recorrido PREORDEN
        print("\nRecorrido PREORDEN:")
        recorrido_preorden(raiz)
    elif opcion == "5":
      #Muestra los nombres en recorrido POSTORDEN
        print("\nRecorrido POSTORDEN:")
        recorrido_postorden(raiz)
    elif opcion == "6":
     #Finaliza el programa
       print("Saliendo del programa.")
        break
    else:
     #Si la opción ingresada no es válida
       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 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 claves 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. Conclusión

Pudimos llegar a la conclusión de que los árboles binarios son estructuras fundamentales en informática, ampliamente utilizadas en áreas como los sistemas de archivos, algoritmos de búsqueda y representación jerárquica de datos. A través del desarrollo del trabajo práctico, se logró implementar un árbol genealógico binario en Python, lo que permitió aplicar conceptos claves como nodos, referencias a hijos, y recorridos recursivos (inorden, preorden y postorden).

Durante la implementación, se emplearon elementos esenciales de programación como funciones, condicionales, bucles (while) para el menú interactivo, y diccionarios como forma sencilla de representar personas, lo que facilitó la comprensión de cómo manipular estructuras de datos. También se abordaron aspectos importantes como la recursividad y el manejo de entradas del usuario.

Si bien el programa cumple su objetivo, se identificaron posibles mejoras, como permitir más de dos hijos por nodo, validar nombres duplicados o incorporar persistencia de datos mediante archivos. Esto refleja una limitación del árbol binario en contextos reales como la genealogía, donde una persona puede tener más de dos descendientes.

En resumen, este trabajo no solo fortaleció habilidades técnicas, sino que también permitió reflexionar sobre la importancia de seleccionar la estructura de datos adecuada según el problema a resolver, y abrió la puerta a futuras mejoras o extensiones más complejas del proyecto.

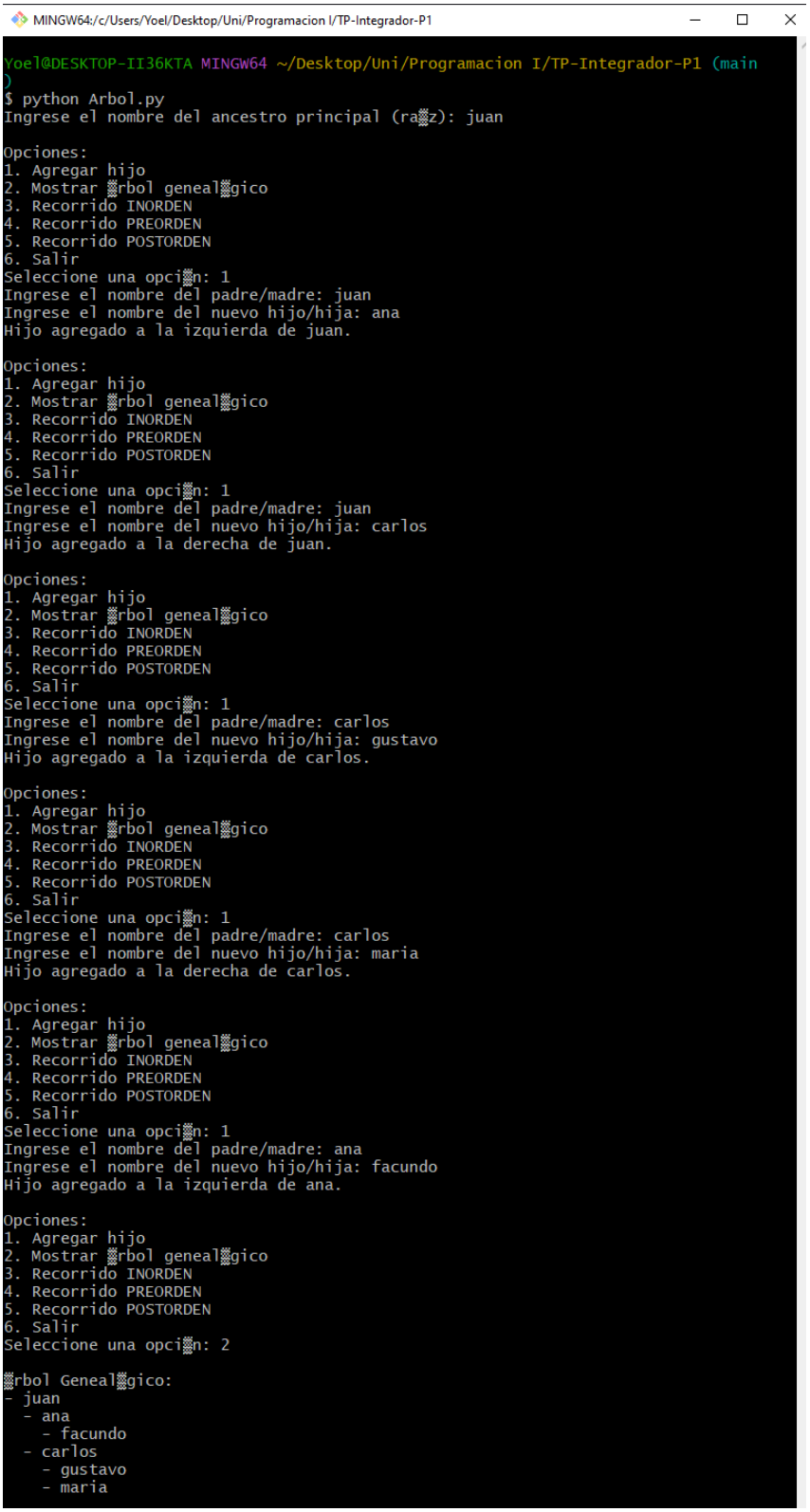
Bibliografía

- 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/
- Geeks for Geeks. (n.d.). *Recursion in Python*. https://www.geeksforgeeks.org/recursion-in-python/
- Documento de “Árboles” proporcionado por la UTN.

8. Anexos

- Capturas de pantalla del código funcionando en consola:

  Realizamos el árbol genealógico de ejemplo



En este caso vemos como quedan todos los recorridos posibles

```
Opciones:
1. Agregar hijo
2. Mostrar ▒rbol geneal▒gico
3. Recorrido INORDEN
4. Recorrido PREORDEN
5. Recorrido POSTORDEN
6. Salir
Seleccione una opci▒n: 3

Recorrido INORDEN:
facundo
ana
juan
gustavo
carlos
maria

Opciones:
1. Agregar hijo
2. Mostrar ▒rbol geneal▒gico
3. Recorrido INORDEN
4. Recorrido PREORDEN
5. Recorrido POSTORDEN
6. Salir
Seleccione una opci▒n: 4

Recorrido PREORDEN:
juan
ana
facundo
carlos
gustavo
maria

Opciones:
1. Agregar hijo
2. Mostrar ▒rbol geneal▒gico
3. Recorrido INORDEN
4. Recorrido PREORDEN
5. Recorrido POSTORDEN
6. Salir
Seleccione una opci▒n: 5

Recorrido POSTORDEN:
facundo
ana
gustavo
maria
carlos
juan
```

En caso de que le asignemos un hijo a alguien que tiene 2 hijos pasa esto

```
Opciones:
1. Agregar hijo
2. Mostrar ▒rbol geneal▒gico
3. Recorrido INORDEN
4. Recorrido PREORDEN
5. Recorrido POSTORDEN
6. Salir
Seleccione una opci▒n: 1
Ingrese el nombre del padre/madre: juan
Ingrese el nombre del nuevo hijo/hija: franco
Esta persona ya tiene dos hijos.
```

Y en el caso de que le queramos agregar un hijo a una persona q no existe

```
Opciones:
1. Agregar hijo
2. Mostrar ▒rbol geneal▒gico
3. Recorrido INORDEN
4. Recorrido PREORDEN
5. Recorrido POSTORDEN
6. Salir
Seleccione una opci▒n: 1
Ingrese el nombre del padre/madre: fabricio
Ingrese el nombre del nuevo hijo/hija: candelaria
Persona no encontrada.
```

Al elegir la opción 6 salimos del programa

```
Opciones:
1. Agregar hijo
2. Mostrar ▒rbol geneal▒gico
3. Recorrido INORDEN
4. Recorrido PREORDEN
5. Recorrido POSTORDEN
6. Salir
Seleccione una opci▒n: 6
Saliendo del programa.
```

- Repositorio en GitHub: https://github.com/GYoelSantarelli/TP-Integrador-P1.git
- Video explicativo mostrando la estructura del árbol, los recorridos y reflexión final: Trabajo práctico integrador: Programación 1