UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico Integrador

Estructuras de datos avanzadas

Árboles

Comisión 25

Macarena Marinoni <*marinonimacarena@gmail.com*>

Marianela Valletto <*maruvalletto@gmail.com*>

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional

Arquitectura y Sistemas Operativos

Docente Titular

Bruselario, Sebastian

Docente Tutor

Carbonari, Veronica

08/06/2025

ÍNDICE

Introducción.....3
Marco teórico..... 4
Caso práctico..... 8
Metodología utilizada.....9
Resultados obtenidos..... 10
Conclusiones.....11
Bibliografía.....12
Anexos..... 13

Introducción

Este trabajo práctico tiene como objetivo aplicar de forma concreta el conocimiento adquirido sobre estructuras de datos avanzadas, específicamente árboles, mediante su desarrollo e implementación en Python.

La propuesta consiste en organizar la planificación académica de un cuatrimestre utilizando un árbol jerárquico. Esta estructura permitirá representar relaciones entre materias, entregas, evaluaciones y reuniones grupales, facilitando así una mejor visualización y gestión de los compromisos académicos.

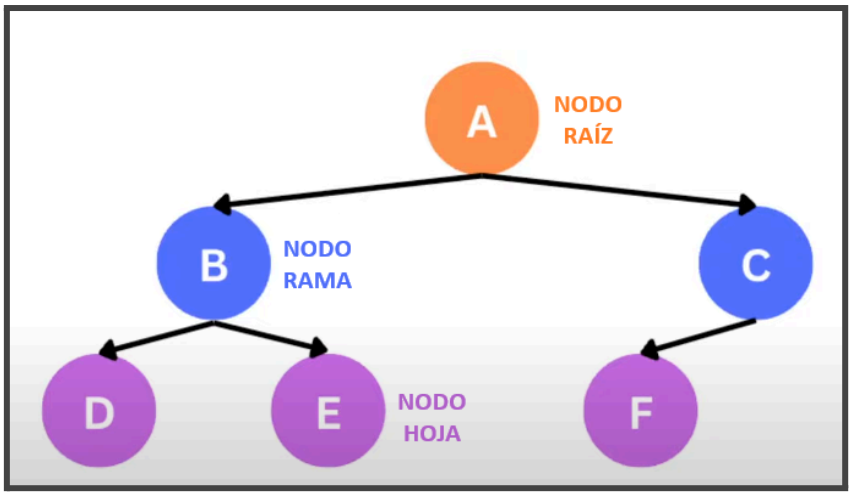
Marco teórico

Un árbol es una estructura de datos jerárquica que se compone de nodos conectados entre sí mediante ramas. Un nodo es un centro de datos, pueden contener cualquier tipo de datos. Una rama es la conexión de los nodos entre sí.

Cada árbol posee un nodo raíz, que es el punto inicial, y a partir de él se ramifican los nodos hijos. Estos pueden ser a su vez nodos internos (si tienen hijos) o nodos hoja (si no tienen hijos).

Hay diferentes tipos de nodos y se clasifican según su ubicación en el árbol como:

- **Raíz**: el nodo principal.
- **Interno o rama**: nodo con padre y al menos un hijo.
- **Hoja**: nodo sin hijos.

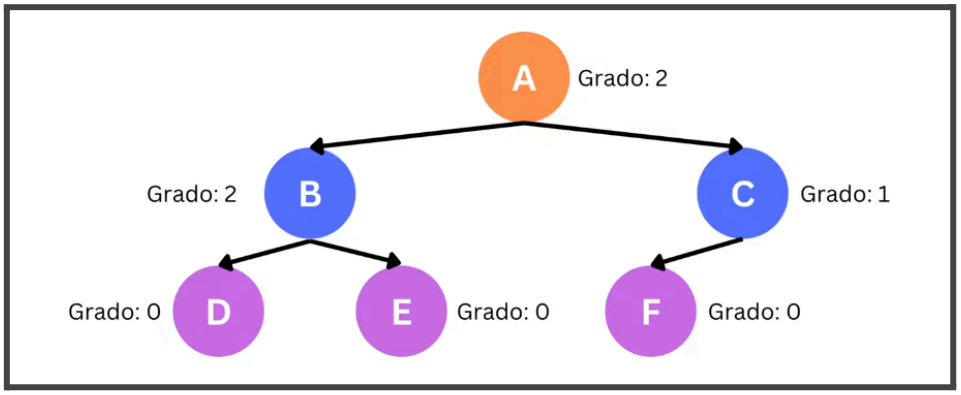


📌Representación gráfica tipos Nodo. Origen: Presentación Docente Julieta Trape.

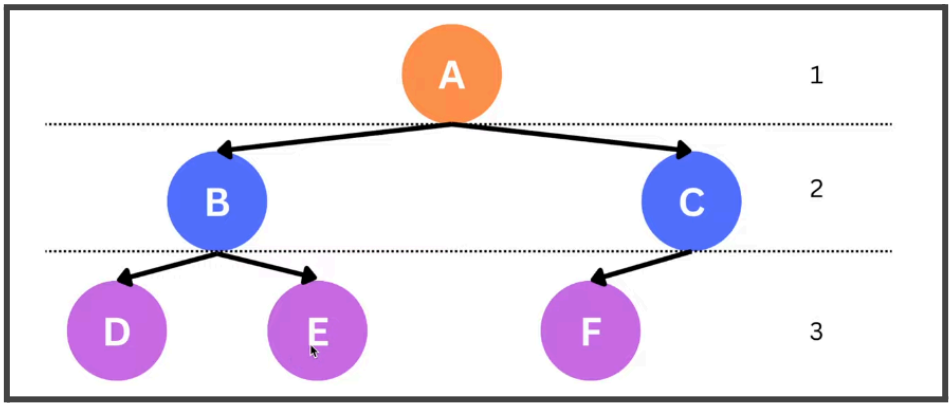
Los árboles permiten representar relaciones jerárquicas que no pueden modelarse eficientemente con listas o diccionarios, ya que estos no permiten múltiples niveles de dependencia.

Entre las propiedades clave de los árboles se encuentran:

- **Altura**: máximo nivel de profundidad.
- **Grado**: cantidad de hijos de un nodo.
- **Peso**: total de nodos del árbol.



📌Representación gráfica grado árbol. Fuente: Presentación Docente Julieta Trape.



📌Representación gráfica peso parbol. Fuente: Presentación Docente Julieta Trape.

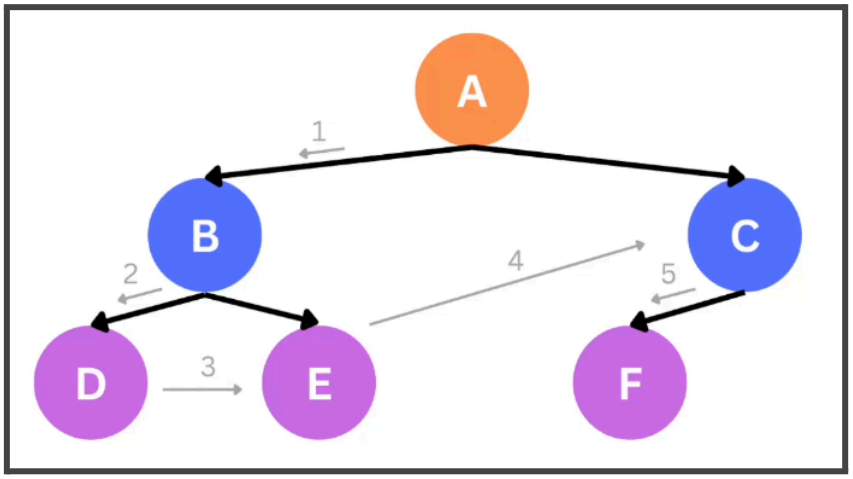
Árbol binario:

Es un tipo especial de árbol en el que cada nodo puede tener como máximo dos hijos: uno izquierdo y uno derecho. Permite representar estructuras jerárquicas simples de forma eficiente.

Recorridos:

- **Preorden**: raiz → izquierda → derecha

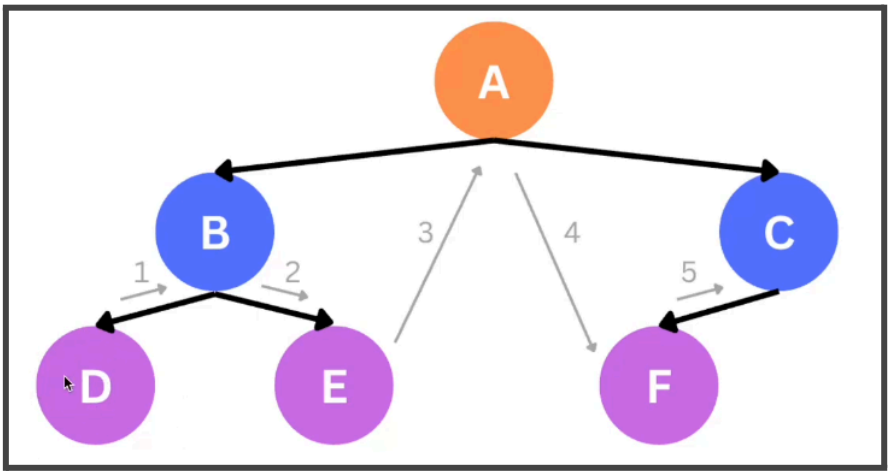
  En este recorrido, primero visitamos la raíz, luego recursivamente visitamos el subárbol izquierdo y finalmente el subárbol derecho.



📌Representación gráfica Preorden: A, B, D, E, C, F. Fuente: Presentación Docente Julieta Trape.

- **Inorden**: izquierda ⟶ raíz ⟶ derecha

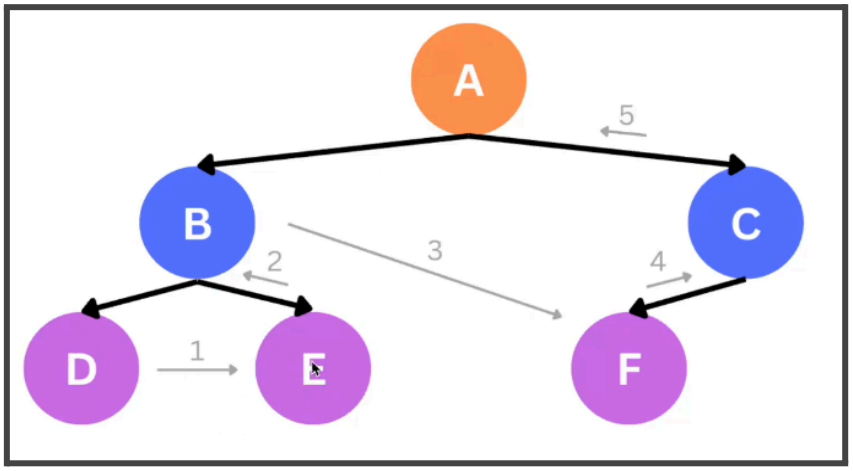
  En este recorrido, primero recursivamente visitamos el subárbol izquierdo, se inicia por su hoja más a la izquierda, luego visitamos la raíz y finalmente el subárbol derecho por la hoja más a la izquierda.



📌Representación gráfica Inorden: B, D, E, A, F, C. Fuente: Presentación Docente Julieta Trape.

- **Postorden**: izquierda ⟶ derecha ⟶ raíz

  En este recorrido, primero recursivamente visitamos el subárbol izquierdo, luego el subárbol derecho y finalmente la raíz.



📌Representación gráfica Postorden: D, E, B, F, C, A. Fuente: Presentación Docente Julieta Trape.

Estos recorridos son fundamentales para procesar, imprimir o analizar la información contenida en los nodos.

Caso práctico

El trabajo parte de un caso práctico centrado en la experiencia de un estudiante que necesita organizar su año académico de forma estructurada y visual. La propuesta consiste en modelar esta planificación utilizando un árbol jerárquico, donde cada nivel de la estructura representa un componente clave del cursado.

En la raíz del árbol se encuentra el **Año 1**, que agrupa los dos cuatrimestres académicos. De cada cuatrimestre se desprenden las **materias correspondientes** como nodos hijos. A su vez, cada materia contiene como hijos los **compromisos académicos** tales como parciales, trabajos prácticos, entregas integradoras y reuniones de grupo. Finalmente, en el nivel más bajo de la jerarquía, se incluyen los **detalles adicionales**, como fechas, modalidad (presencial o a distancia), o el tipo de entrega.

La estructura completa puede visualizarse en los siguientes niveles:

- **Nivel 0:** Año académico (raíz del árbol)
- **Nivel 1:** Cuatrimestres (1° y 2°)
- **Nivel 2:** Materias de cada cuatrimestre (Programación I, Matemática, etc.)
- **Nivel 3:** Compromisos por materia (parciales, TPs, entregas, reuniones)
- **Nivel 4:** Detalles específicos de cada compromiso (fecha, modalidad, grupo)

Esta jerarquía fue implementada en **Python 3** mediante una clase personalizada llamada Nodo, que permite construir la estructura mediante listas anidadas de hijos. La clase incluye un método **agregar_hijo()** para incorporar descendientes a cada nodo y un método imprimir() que recorre el árbol de forma recursiva utilizando un recorrido **preorden**. Esta elección permite mostrar cada nivel del árbol de arriba hacia abajo, respetando el orden lógico de planificación.

Gracias al uso de Python 3, el código resultante es claro, legible y fácilmente extensible, y permite simular operaciones como inserciones, listados jerárquicos y visualización en consola con formato estructurado. La flexibilidad del árbol lo convierte en una solución adecuada para modelar relaciones académicas complejas de forma intuitiva y ordenada.

Metodología utilizada

El desarrollo del trabajo se estructuró en distintas etapas, comenzando con una revisión teórica sobre estructuras de datos no lineales, en particular los árboles. Se analizaron conceptos como nodos, jerarquías, tipos de árboles (binarios y n-arios) y recorridos, para luego definir un caso práctico con aplicación directa al ámbito académico de los estudiantes.

La idea central fue representar de forma estructurada y jerárquica la organización del cursado del primer año de la carrera, abarcando los dos cuatrimestres, sus respectivas materias y los compromisos asociados (parciales, trabajos prácticos, entregas grupales, etc.).

Con ese objetivo, se diseñó un árbol general (n-ario), donde el nodo raíz representa el **Año 1**, y de él se desprenden los nodos correspondientes al **Primer Cuatrimestre** y **Segundo Cuatrimestre**. Cada cuatrimestre contiene como hijos las **materias cursadas**, y cada materia, a su vez, tiene como hijos los **compromisos académicos** con detalles incluidos en sus nombres (fecha, modalidad, tipo de entrega).

Para implementar esta estructura se desarrolló en Python una clase personalizada llamada **Nodo**. Esta clase contiene un atributo nombre y una lista de hijos, lo que permite construir una estructura jerárquica flexible sin restricciones de cantidad de descendientes por nodo. Además, se definió un método **agregar_hijo()** para vincular los nodos y un método **imprimir()** que recorre recursivamente el árbol y lo muestra en consola con formato visual de árbol, usando caracteres como ├── y └── para representar la jerarquía.

El método **imprimir()** realiza un recorrido **preorden**, es decir, imprime primero el nodo actual y luego recorre e imprime recursivamente todos sus hijos, de izquierda a derecha. Esta estrategia permite que la visualización respete la lógica de organización: primero se ve el año, luego el cuatrimestre, seguido de cada materia y finalmente sus actividades.

Durante la etapa de pruebas, se verificó que el árbol imprimiera correctamente la estructura completa, y se probaron distintos escenarios de materias y actividades para asegurar que el diseño fuera escalable y representara con claridad la planificación académica. Por último, se documentó el proceso de desarrollo y se reflexionó sobre los conocimientos aplicados, destacando el valor de las estructuras de datos jerárquicas para representar información real y compleja de manera clara y ordenada.

Resultados obtenidos

El programa desarrollado permitió visualizar correctamente la estructura jerárquica del cuatrimestre y reflejó el uso exitoso de árboles para:

- Organizar información de forma no lineal.
- Agregar nodos con múltiples niveles de dependencia.
- Representar relaciones jerárquicas mediante programación orientada a objetos.

La impresión del árbol recursivamente facilitó la comprensión de la estructura general y ayudó a validar el funcionamiento del código implementado.

Ventajas observadas:

- Claridad visual al representar información compleja.
- Flexibilidad para agregar y organizar nuevos elementos.
- Reutilización del código en otros contextos (organización personal, tareas, proyectos).
- Aplicación directa de recursividad.

Desventajas o limitaciones:

- No permite búsqueda directa de información sin funciones adicionales.
- La visualización se limita al entorno de consola (no es gráfica ni interactiva).
- Puede volverse complejo de recorrer si la estructura crece mucho sin control.

Conclusiones

La realización de este trabajo permitió al equipo aplicar en la práctica los conocimientos teóricos sobre árboles como estructura de datos. Se aprendió a construir jerarquías utilizando programación orientada a objetos y a recorrer estructuras mediante métodos recursivos, lo que reforzó particularmente la comprensión del concepto de recursividad.

Durante el desarrollo, se descubrió que los árboles resultan muy útiles no solo en problemas técnicos, como compiladores o algoritmos de búsqueda, sino también en contextos cotidianos, como la organización académica o de proyectos. Esta versatilidad los convierte en una herramienta clave para desarrollar sistemas estructurados y eficientes.

A futuro, el código podría ampliarse incorporando funcionalidades como búsqueda de compromisos, filtrado por fechas o exportación a otros formatos, como JSON.

Una de las principales dificultades iniciales fue comprender cómo representar relaciones jerárquicas con múltiples niveles en el código, especialmente en lo referente al diseño recursivo del método de impresión. Esta problemática se resolvió mediante el análisis de ejemplos, la realización de pruebas iterativas y el estudio de los materiales de clase.

En conclusión, este trabajo integrador no solo permitió consolidar los contenidos sobre estructuras de datos, sino que también motivó al equipo a pensar en aplicaciones concretas y escalables de los árboles en contextos de programación real.

Bibliografía

Bibliografía Python Software Foundation. (s.f.). Welcome to Python.org. Recuperado el 06 de junio de 2025, de https://www.python.org

Universidad Tecnológica Nacional – Facultad Regional Rosario. (2025). Material de clase: Árboles como estructura de datos [Apuntes]. Cátedra de Programación I.

Marinoni, M. (2025). Trabajo práctico integrador UTN TUPaD [Repositorio GitHub]. Recuperado de https://github.com/MaquiMarinoni/TUPaD_P1-C12025

Valletto, M. (2025). Trabajo práctico integrador UTN TUPaD [Repositorio GitHub]. Recuperado de https://github.com/MaruValletto/UTN-TUPaD-P1_Valletto

Anexos

- Presentación y video

https://drive.google.com/drive/folders/1ztoRGr-KqQMPlLP_vrujtYHYZvCNFwUg?usp=drive_link

- Script

```
# Trabajo Práctico Integrador
# Tecnicatura Universitaria en Programación a Distancia - UTN
# Alumno: Marinoni Macarena, Valleto Marianela
# Comisión 25


# Definicion para la clase Nodo que representa un nodo en el árbol
# Cada nodo tiene un nombre y una lista de hijos.
class Nodo:
    def __init__(self, nombre):
        self.nombre = nombre
        self.hijos = []


# El método agregar_hijo permite añadir un hijo al nodo actual.
    def agregar_hijo(self, hijo):
        self.hijos.append(hijo)


# El método imprimir permite mostrar el árbol de forma jerárquica.
    def imprimir(self, prefijo="", es_ultimo=True):
```

```
        conector = "└── " if es_ultimo else "├── "
        print(prefijo + conector + self.nombre)
        nuevo_prefijo = prefijo + ("    " if es_ultimo else "│   ")
        for i, hijo in enumerate(self.hijos):
            ultimo = i == len(self.hijos) - 1
            hijo.imprimir(nuevo_prefijo, ultimo)

# Crear estructura raíz del árbol
anio1 = Nodo("Año 1")

# Cuatrimestres
cuatrimestre1 = Nodo("Primer Cuatrimestre")
cuatrimestre2 = Nodo("Segundo Cuatrimestre")
anio1.agregar_hijo(cuatrimestre1)
anio1.agregar_hijo(cuatrimestre2)

# Materias Cuatrimestre 1
prog1 = Nodo("Programación I")
ayso = Nodo("Arquitectura y Sistemas Operativos")
mate = Nodo("Matemática")
orga = Nodo("Organización Empresarial")
cuatrimestre1.agregar_hijo(prog1)
cuatrimestre1.agregar_hijo(ayso)
cuatrimestre1.agregar_hijo(mate)
```

```
cuatrimestre1.agregar_hijo(orga)


# Compromisos por materia Cuatrimestre 1


prog1.agregar_hijo(Nodo("Parcial 1 - Fecha: 10/05 - Modalidad: A
distancia"))

prog1.agregar_hijo(Nodo("TP Integrador - Fecha: 09/06"))

prog1.agregar_hijo(Nodo("Parcial 2 - Fecha: 14/06 - Modalidad: A
distancia"))


ayso.agregar_hijo(Nodo("Parcial 1 - Fecha: 03/05 - Modalidad: A
distancia"))

ayso.agregar_hijo(Nodo("Entrega Proyecto Virtualización - Fecha:
05/06"))

ayso.agregar_hijo(Nodo("Parcial 2 - Fecha: 11/06 - Modalidad: A
distancia"))


mate.agregar_hijo(Nodo("TP Integrador - Fecha: 28/04"))

mate.agregar_hijo(Nodo("Parcial 1 - Fecha: 14/05 - Modalidad: A
distancia"))

mate.agregar_hijo(Nodo("TP Integrador 2 - Fecha: 13/06"))

mate.agregar_hijo(Nodo("Parcial 2 - Fecha: 25/06 - Modalidad: A
distancia"))

mate.agregar_hijo(Nodo("TP Integrador 3 - Fecha: 30/06"))


orga.agregar_hijo(Nodo("TP Individual 1 - Fecha: "))

orga.agregar_hijo(Nodo("TP Individual 2 - Fecha: 27/04"))
```

```
orga.agregar_hijo(Nodo("Parcial 1 - Fecha: 17/05 - Modalidad: A
distancia"))

orga.agregar_hijo(Nodo("Parcial 2 - Fecha: 07/06 - Modalidad: A
distancia"))

orga.agregar_hijo(Nodo("TP Integrador - Fecha: 19/06 "))


cuatrimestre1.agregar_hijo(Nodo("Reunión con grupo de Programación -
Fecha: 11/06"))



# Materias Cuatrimestre 2

prog2 = Nodo("Programación II")

estad = Nodo("Probabilidad y Estadística")

bdd1 = Nodo("Base de Datos I")

ingles1 = Nodo("Inglés I")

cuatrimestre2.agregar_hijo(prog2)

cuatrimestre2.agregar_hijo(estad)

cuatrimestre2.agregar_hijo(bdd1)

cuatrimestre2.agregar_hijo(ingles1)


cuatrimestre2.agregar_hijo(Nodo("Mesa de examen - Fecha tentativa: 26
al 28/11"))


# Compromisos por materia Cuatrimestre 2
```

```
prog2.agregar_hijo(Nodo("Inicio tentativo cursado - Fecha: 01/08"))

estad.agregar_hijo(Nodo("Inicio tentativo cursado - Fecha: 01/08"))

bdd1.agregar_hijo(Nodo("Inicio tentativo cursado - Fecha: 01/08"))

ingles1.agregar_hijo(Nodo("Inicio tentativo cursado - Fecha: 01/08"))


# Imprimir árbol completo

anio1.imprimir()
```