## Programación I

# Trabajo Práctico Integrador

# Árboles

**Profesora:** Prof. Cinthia Rigoni

**Tutor:** Prof. Walter Pintos

**Alumnas**

Alexia Abi Rubin ( alexiarubin23@gmail.com)

Maria Victoria Volpe (mvictoriavolpe@gmail.com)

## 1. Introducción

Los árboles binarios constituyen una estructura de datos fundamental en programación, utilizada para representar relaciones jerárquicas entre elementos. Su aplicación se extiende desde contextos cotidianos como árboles genealógicos y estructuras de archivos, hasta sistemas complejos como algoritmos de búsqueda, compiladores y sistemas de decisión.

Este trabajo aborda la implementación de árboles binarios de decisión en Python, utilizando listas anidadas como estructura de representación. Se desarrollaron dos casos prácticos: un test interactivo que determina el tipo de estudiante mediante preguntas binarias y una simulación simplificada del juego BlackJack que utiliza árboles de decisión para representar las opciones del jugador.

---

## 2. Objetivos

- Comprender cómo se puede modelar un árbol de decisiones binario utilizando listas anidadas en Python.
- Aplicar funciones recursivas para simular decisiones binarias dentro de un flujo de juego interactivo.
- Calcular propiedades estructurales de un árbol binario, como su peso (cantidad de nodos) y altura (profundidad máxima), a partir de su representación en listas.

---

## 3. Marco Teórico

### Definición y Estructura

Un árbol binario es una estructura de datos no lineal compuesta por nodos, donde cada nodo tiene como máximo dos hijos denominados izquierdo y derecho. Según Knuth (1997), esta estructura se caracteriza por tener un nodo raíz único desde el cual se ramifican todos los demás elementos.

### Propiedades Fundamentales

- **Raíz:** nodo inicial del árbol
- **Hojas:** nodos sin descendientes

- **Nodos internos:** nodos con al menos un hijo
- **Subárboles:** cualquier nodo con sus descendientes

**Medidas estructurales:**

- **Altura:** longitud del camino más largo desde la raíz hasta una hoja
- **Peso:** cantidad total de nodos en el árbol
- **Nivel:** profundidad de un nodo contando desde la raíz
- **Grado:** número de hijos de un nodo específico

**Implementación en Python**

La representación mediante listas anidadas utiliza el formato [valor, subárbol_izquierdo, subárbol_derecho], donde cada subárbol mantiene la misma estructura recursiva. Esta aproximación, aunque menos eficiente que las implementaciones con punteros, facilita la comprensión conceptual y resulta adecuada para estructuras de tamaño moderado.

**Árboles de Decisión**

Los árboles de decisión representan procesos de toma de decisiones mediante estructura jerárquica, donde cada nodo interno corresponde a una decisión y las hojas a resultados finales. Esta aplicación resulta especialmente útil en sistemas expertos, juegos y procesos de clasificación.

---

**4. Metodología Utilizada**

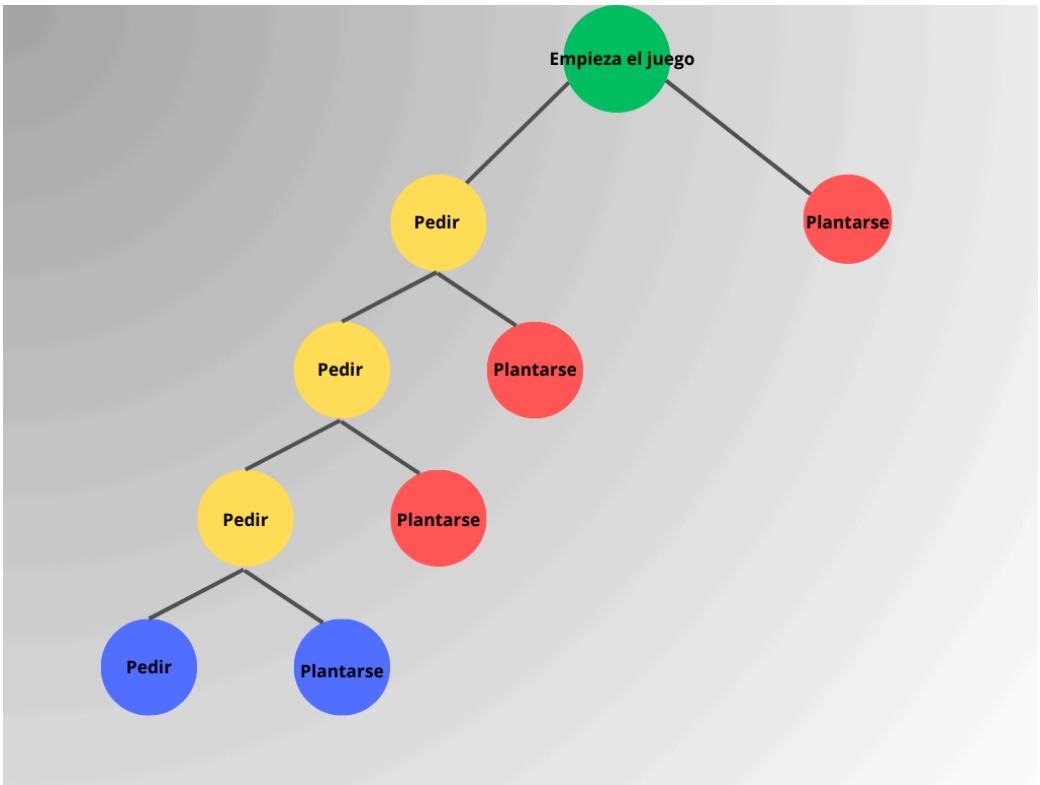
**Etapas de Desarrollo**

**Investigación previa:** Se consultó documentación sobre estructuras de datos y árboles binarios, utilizando fuentes académicas y documentación oficial de Python.

**Diseño de la estructura:** Se definió la representación mediante listas anidadas, estableciendo la convención [valor, hijo_izquierdo, hijo_derecho] para mantener consistencia.

**Implementación iterativa:** El desarrollo se realizó en etapas progresivas, comenzando con funciones básicas de recorrido y añadiendo gradualmente funcionalidades de análisis estructural.

1. **Modelado del árbol de decisiones**: se creó un grafo donde cada elemento contiene:
   - Una etiqueta de estado (por ejemplo, “Pedir carta”).
   - Un subárbol izquierdo (para la acción de pedir otra carta).
   - Un subárbol derecho (para la acción de plantarse).




2. **Implementación en Python**: se codificó el árbol utilizando listas y funciones recursivas que permiten simular el avance del juego según las elecciones del usuario:

```
arbol_juego = [
    "Empezar juego",
    ["Pedir carta",
        ["Pedir carta",
            ["Pedir carta", ["Pedir carta", [], []], ["Plantarse", [], []]],
            ["Plantarse", [], []]
        ],
        ["Plantarse", [], []]
    ],
    ["Plantarse", [], []]
]
```

3. **Simulación del juego**: se desarrollaron funciones para manejar la lógica del juego de cartas (sacar cartas, sumar puntos, mostrar cartas) y para recorrer el árbol según las decisiones del jugador:

```
def recorrer_arbol(arbol_juego, primera_mano=False):
    if arbol_juego[1]==[] and arbol_juego[2]==[]: #condición para detectar hojas #además es el caso de corte para la recursión
        print("\n🔴 No hay más nodos.\n")

    else:
        if primera_mano:
            print(f"\n🎲 {arbol_juego[0]}") #si es la primera mano imprimo el nodo padre y reparto 2
            print(f"Te tocó {sacar_carta()} y un {sacar_carta()}.")

        print(f"Puntos totales: {sumar_mano()}.") #sumo las cartas de la mano.
        empezar=input(f"¿Quieres pedir otra carta (1)?, o 'Plantarte'(2)? ")

        if empezar == "1": #si el usuario "pide" carta:
            print(f"Te tocó un {sacar_carta()}.") #saca 1

            if sumar_mano()>21: #verifica si se pasa de 21
                print(f"💥 ¡Te pasaste! Perdiste con {sumar_mano()} puntos.")
                return

            recorrer_arbol(arbol_juego[1]) #Al recorrer recursivamente va a ir sacando cartas y sumándolas.

        elif empezar == "2":
            print(f"Te plantaste con {sumar_mano()} puntos.")
            mostrar_puntuaciones(sumar_mano())
```

Por otro lado, se realizó un test que realiza preguntas al usuario y devuelve un resultado. Se construyó un árbol binario donde cada nodo representa una **pregunta** o un **resultado final**, con la siguiente estructura:

- Un texto que actúa como pregunta.
- Una **rama izquierda** que representa la respuesta *"Sí"*.
- Una **rama derecha** que representa la respuesta *"No"*.

El recorrido por el árbol se define en función de las respuestas del usuario, hasta llegar a un nodo hoja con el resultado final (tipo de estudiante).

El árbol fue modelado como una lista anidada de tres elementos por nodo:  [valor, subárbol izquierdo, subárbol derecho]

Se programó una función recursiva llamada recorrer_test() que:

Muestra una pregunta.

Espera la elección del usuario: 1 para Sí, 2 para No.

Según la respuesta, recorre la rama correspondiente.

Cuando llega a un nodo sin subárboles, muestra un resultado.

**Herramientas Utilizadas**

- **Lenguaje:** Python 3.x
- **Github**
- **Metodología de trabajo:** Programación colaborativa con división de tareas

**Trabajo Colaborativo**

Las tareas se distribuyeron equitativamente: una integrante se enfocó en la teoría e implementación del test de clasificación mientras la otra desarrolló el simulador de BlackJack. Las funciones de análisis estructural se implementaron conjuntamente.

**5. Análisis estructural del árbol**

- contar_nodos(arbol): calcula el número total de nodos recorriendo el árbol en profundidad. Cada nodo cuenta como una unidad, y la función se aplica recursivamente sobre los subárboles izquierdo y derecho.
- calcular_altura(arbol): determina la altura máxima del árbol, definida como la longitud del camino más largo desde la raíz hasta una hoja. La función compara recursivamente la altura de ambos subárboles y retorna el valor máximo más uno.
- contar_hojas(arbol): contabiliza la cantidad de hojas (nodos sin hijos). La función verifica si ambos hijos del nodo actual están vacíos y, en ese caso, incrementa el contador. En caso contrario, continúa el recorrido por ambos subárboles.

```python
def contar_nodos(arbol):
    if arbol == []:
        return 0
    return 1 + contar_nodos(arbol[1]) + contar_nodos(arbol[2])

def calcular_altura(arbol):
    if arbol == []:
        return 0
    return 1 + max(calcular_altura(arbol[1]), calcular_altura(arbol[2]))

def contar_hojas(arbol):
    if arbol[1] == [] and arbol[2] == []:
        return 1
    return contar_hojas(arbol[1]) + contar_hojas(arbol[2])
```

Estas funciones permiten obtener una caracterización estructural básica del árbol, relevante para el análisis del comportamiento y la eficiencia de algoritmos que operan sobre estructuras jerárquicas.

**6. Resultados obtenidos**

La implementación realizada permitió construir y recorrer un árbol binario de forma clara y funcional, utilizando listas anidadas en Python. Esta aproximación, si bien presenta limitaciones en términos de escalabilidad, resultó adecuada para introducir los conceptos fundamentales de árboles binarios. Entre sus ventajas, se identificó la simplicidad conceptual que favorece la comprensión del modelo.

Los resultados validan esta forma de representación como una herramienta eficaz para el abordaje inicial de árboles binarios permitiendo una comprensión progresiva antes de avanzar hacia modelos más robustos.

---

**7. Conclusiones**

El desarrollo de este trabajo permitió comprender los fundamentos de los árboles binarios desde una perspectiva tanto teórica como práctica. La representación mediante listas anidadas demostró ser una herramienta pedagógica efectiva para introducir conceptos estructurales complejos.

La implementación de casos prácticos reforzó la comprensión de la recursividad y el manejo de estructuras jerárquicas. Los árboles de decisión resultaron especialmente útiles para modelar procesos de toma de decisiones binarias, demostrando la versatilidad de esta estructura de datos.

Las funciones de análisis estructural proporcionaron herramientas valiosas para validar y caracterizar las estructuras creadas, evidenciando cómo Python permite trabajar con conceptos complejos mediante sintaxis accesible.

**Aprendizajes clave:**

- Comprensión profunda de la estructura y navegación de árboles binarios
- Aplicación práctica de recursividad en problemas reales
- Experiencia en implementación colaborativa de sistemas interactivos

**8. Bibliografía**


- Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). *Introduction to Algorithms*.
- Miller, B. & Ranum, D. *Problem Solving with Algorithms and Data Structures using Python*.
- Python Software Foundation. (s.f.). *The Python Standard Library*. Recuperado el 9 de junio de 2025, de https://docs.python.org/3/