

Arboles Binarios

- **Alumnos:**
 - Gonzalo Barrios – gonza.barrios.248@gmail.com
 - Agustín Caro – aguscaro88@gmail.com
 - **Materia:** Programación I
 - **Profesora:** Cinthia Rigoni **Tutor:** Oscar Londero.
 - **Fecha de Entrega:** 07/06/2025
-

Índice

Introducción	1
Marco Teórico	2
Caso Práctico	3
Metodología Utilizada	7
Resultados Obtenidos	8
Conclusiones	10
Bibliografía	10
Anexos	10

1. Introducción

Elegimos el tema de Árbol Binario de Búsqueda (ABB) en Python por su importante utilización fundamental en estructuras de datos y algoritmos, siendo una de las herramientas más claves para operaciones eficientes de inserción, búsqueda y recorrido. Utilizamos un caso práctico conocido para comprender su funcionamiento interno mediante utilización de clases y funciones que permiten al usuario clasificar y recorrer.



2. Marco Teórico

Un ABB es una estructura de datos jerárquica donde cada nodo tiene como máximo dos hijos: izquierdo y derecho. Se mantiene ordenado: los valores menores al nodo se colocan a la izquierda y los mayores o iguales a la derecha.

Ejemplo de ABB:



Raíz: 50

Rama: 30, 70

Hoja: 10, 60, 80

Para describirlo en palabras seria:

- 50 es el padre de 30 y 70.
- 30 es hijo izquierdo de 50, hermano de 70 y padre de 10.
- 10 es hijo izquierdo de 30.
- 70 es hijo derecho de 50, hermano de 30 y padre de 60 y 80.
- 60 es hijo izquierdo de 70 y hermano de 80.
- 80 es hijo derecho de 70 y hermano de 60.

Para el trabajo en Python esto se implementa de la siguiente manera:

Se crean dos **Clases** principales:



- *Nodo*: se utiliza para almacenar un valor y hacer referencias a sus hijos.
- *Árbol*: El cual contiene la raíz y los métodos para insertar ramas e hijos y métodos para recorrerlo:
 - *Preorden*: visita nodo raíz, luego subárbol izquierdo y derecho.
 - *Inorden*: visita subárbol izquierdo, luego nodo raíz y luego subárbol derecho.
 - *Postorden*: visita subárbol izquierdo y derecho, luego el nodo raíz.

Los árboles tienen **Propiedades**:

- Longitud de camino: número de ramas que hay que transitar para llegar de un nodo a otro.
- Profundidad: es la longitud de camino entre el nodo raíz y un nodo.
- Nivel: es la longitud del camino que conecta un nodo al nodo raíz más uno.
- Altura: es el máximo nivel de un árbol.
- Grado: es el número de hijos que tiene un nodo.
- Orden: es la máxima cantidad de hijos que puede tener cada nodo.
- Peso: Es el número total de nodos que tiene un árbol.

3. Caso Práctico

La simulación aplicada en nuestro caso práctico se realizó para resolverle al usuario el problema de tener que ordenar datos ya sea numéricos o letras.

Función para preguntar con qué trabajar

```
def obtener_tipo_dato():

    while True:

        tipo = input("¿Con qué deseas trabajar, letras(L) o números(N)? ").strip().lower()

        if tipo in ["l", "n"]:

            return "letras" if tipo == "l" else "numeros"

    print("Opción inválida. Escribe 'L' o 'N'.")
```



```
PS-07 (user3) ~ % cd /mnt/c/Users/3/Downloads/PyProyecto_Estructuras/; python3.7 main.py
¿Con qué deseas trabajar, letras(L) o números(N)?
```

Cuando el usuario indica que tipo de dato va a utilizar y se confirma que lo ingreso correctamente, se indicara que ingrese los datos a ordenar.

```
# Inicio
```

```
tipo_dato = obtener_tipo_dato()

entrada = input(f"Ingresá {'letras separadas' if tipo_dato == 'letras' else 'números separados'} por espacios: ").split()
```

```
Ingresá letras separadas por espacios:
```

Luego de que ingrese los datos a ordenar, se le consultara que quiere ver del árbol (nodo raiz, nodo rama, nodo hoja o todos) y como quiere verlos ordenados (Preorden, Inorden, Postorden, o todos).

```
# Opciones para que el usuario eliga que quiere ver raiz, ramas o hojas o todo.
```

```
opcion = int(input("\nPara ver cual es Raiz, cual es Rama y cual es Hoja ingresa:\n1. Ver Nodo Raiz.\n2. Ver Nodo Rama.\n3. Ver Nodo Hoja.\nPara ver todas las opciones presione cualquier otro número.\n"))
```

```
# Lista para almacenar los nodos
```

```
raiz = []
```

```
rama = []
```

```
hoja = []
```

```
# Funcion recursiva para clasifica los nodos del arbol y guardarlos en las listas
```

```
def clasificar_nodos(nodo, es_raiz=False):
```

```
    if nodo:
```

```
        # El primero es el nodo raiz, luego sigue con los rama y los ultimos son los hijos
```

```
        if es_raiz:
```

```
            raiz.append(nodo.valor)
```

```
        # Nodo con hijos
```

```
        elif nodo.izq or nodo.der:
```

```
            rama.append(nodo.valor)
```

```
# Nodo sin hijos
else:
    hoja.append(nodo.valor)
# Recorre hijo izquierdo
clasificar_nodos(nodo.izq)
# Recorre hijo derecho
clasificar_nodos(nodo.der)

# Se llama a la funcion con es_raiz True para que el primero sea la raiz
clasificar_nodos(arbol.raiz, es_raiz = True)
```

```
# Convierte numeros a string para imprimir porque .join no anda con int
if tipo_dato == "numeros":
    raiz = [str(v) for v in raiz]
    rama = [str(v) for v in rama]
    hoja = [str(v) for v in hoja]

# Muestra los nodos segun la opcion elegida
if opcion == 1:
    print(f"\nNodo Raiz: {' '.join(raiz)}")
elif opcion == 2:
    print(f"\nNodo/s Rama: {' '.join(rama)}")
elif opcion == 3:
    print(f"\nNodo/s Hoja: {' '.join(hoja)}")
else:
    print(f"\nNodo Raiz: {' '.join(raiz)}")
    print(f"Nodo/s Rama: {' '.join(rama)}")
    print(f"Nodo/s Hoja: {' '.join(hoja)}")
```



```
# Pregunta al usuario que tipo de recorrido mostrar

opcion = int(input("\nPara ver recorridos ingresa:\n1. Recorrido Preorden.\n2. Recorrido Inorden.\n3. Recorrido Postorden.\nPara ver todas las opciones presione cualquier otro número.\n"))

# Ejecuta el recorrido elegido

if opcion == 1:

    print("\nRecorrido Preorden:")

    arbol.preorden(arbol.raiz)

elif opcion == 2:

    print("\nRecorrido Inorden:")

    arbol.inorden(arbol.raiz)

elif opcion == 3:

    print("\nRecorrido Postorden:")

    arbol.postorden(arbol.raiz)

else:

    print("\nRecorrido Preorden:")

    arbol.preorden(arbol.raiz)

    print("\nRecorrido Inorden:")

    arbol.inorden(arbol.raiz)

    print("\nRecorrido Postorden:")

    arbol.postorden(arbol.raiz)
```

```
Para ver cual es Raiz, cual es Rama y cual es Hoja ingresa:
1. Ver Nodo Raiz.
2. Ver Nodo Rama.
3. Ver Nodo Hoja.
Para ver todas las opciones presione cualquier otro número.
3

Nodo/s Hoja: E Q

Para ver recorridos ingresa:
1. Recorrido Preorden.
2. Recorrido Inorden.
3. Recorrido Postorden.
Para ver todas las opciones presione cualquier otro número.
2

Recorrido Inorden:
D E E H M N Q U Z
```



4. Metodología Utilizada

Para comenzar el trabajo se investigó principalmente la documentación presentada por los profesores, con mirar los videos y leer el documento ya te queda bastante clara la importancia de la utilización de Arboles Binarios en programación.

- Lista de Videos:

https://www.youtube.com/watch?v=cVm51pO35qA&list=PLy5wpwhsM-2IY-ge_fALJ4K_XAhLZ2I-&index=6

- Documento:

https://docs.google.com/document/d/10k16oL15EeyOaq92aoi4qwK3t_22X29-FSV2iV-8N1U/edit?tab=t.0

Luego de tener clara su utilidad dentro de grandes tipos de datos, decidimos ir por el camino de tener las dos Clases principales previamente nombradas (Nodo y Árbol) para utilizar dentro del código de Python, sabiendo que el código se va a centrar en el trabajo automático para que el usuario tenga la información que necesite lo más clara y rápidamente presentada posible.

Las dos Clases las realizamos en conjunto entre los dos y con la gran influencia de lo mostrado por los profesores.

Cuando llego la parte de crear las primeras interacciones con el usuario, determinar que tipo de dato se utilizara, verificar que todo lo ingresado sea correcto y continuar, Agustín decidió la opción de poner un bucle While para que si o si se ingrese lo necesario y su verificación posterior, con lo cual si todo va correcto, se creara el Árbol.

Con el Árbol Binario ya creado, entre ambos se implementó la función de poder almacenar el listas cada tipo de Nodo(def clasificar_nodos()).

Teniendo ya los datos ingresados, verificados, y el árbol ya creado y los datos almacenados, Gonzalo se centro en las interacciones finales con el usuario, agregando las opciones de mostrar los Nodos(raíz, ramas y hojas) por separado o todos juntos, y mostrar los Recorridos (Preorden, Inorden y Postorden) necesarios o todos.

5. Resultados Obtenidos

Luego de ya tener la idea totalmente implementada en el código, se logró una rápida ordenanza de Arboles Binarios con tipos de datos a elección del usuario.

Se tuvo problemas típicos de incompatibilidades de funciones de Python con tipos de datos, como por ejemplo:

.join() no acepta números enteros (int) pero se tenía que utilizar para eliminar los corchetes de las listas y tener un espacio entre los datos mostrados, con lo cual pusimos lo siguiente:

```
# Convierte numeros a string para imprimir porque .join no anda con int
if tipo_dato == "numeros":
    raiz = [str(v) for v in raiz]
    rama = [str(v) for v in rama]
    hoja = [str(v) for v in hoja]
```

En ese bloque de código se verifica si el tipo de dato utilizado son números y luego se pasan a string con un simple bucle for, así .join funciona correctamente.

Por suerte, las clases funcionaron correctamente casi desde el principio, tuvimos complicaciones con la Recursividad de las funciones, pero luego de revisar videos y documentación previamente utilizada en Programación 1, se corrigió rápido.

Casos de prueba utilizando ambos tipos de datos (int/str):

```
¿Con qué deseas trabajar, letras(L) o números(N)? n
Ingresá números separados por espacios: 40 60 10 50 20 30 70
```

Para ver cual es Raiz, cual es Rama y cual es Hoja ingresa:

1. Ver Nodo Raiz.
2. Ver Nodo Rama.
3. Ver Nodo Hoja.

Para ver todas las opciones presione cualquier otro número.

2

Nodo/s Rama: 10 20 60

Para ver recorridos ingresa:

1. Recorrido Preorden.
2. Recorrido Inorden.
3. Recorrido Postorden.

Para ver todas las opciones presione cualquier otro número.

6

Recorrido Preorden:

40 10 20 30 60 50 70

Recorrido Inorden:

10 20 30 40 50 60 70

Recorrido Postorden:

30 20 10 50 70 60 40

```
¿Con qué deseas trabajar, letras(L) o números(N)? l
```

```
Ingresá letras separadas por espacios: y d a b i m x
```

Para ver cual es Raiz, cual es Rama y cual es Hoja ingresa:

1. Ver Nodo Raiz.
2. Ver Nodo Rama.
3. Ver Nodo Hoja.

Para ver todas las opciones presione cualquier otro número.

5

Nodo Raiz: Y

Nodo/s Rama: D A I M

Nodo/s Hoja: B X

Para ver recorridos ingresa:

1. Recorrido Preorden.
2. Recorrido Inorden.
3. Recorrido Postorden.

Para ver todas las opciones presione cualquier otro número.

1

Recorrido Preorden:

Y D A B I M X



6. Conclusiones

Gracias al desarrollo de este Trabajo Practico se logro entender mas a profundidad el lógica detrás de Nodos, Arboles Binarios, como recorrerlos y la importancia de saber trabajar con diferentes tipos de datos.

Destacamos la importancia de la recursividad en la funciones y lo que logra tener una buena validación de datos de entrada para tener un código entendible y ordenado la menor cantidad de cabos sueltos posibles.

Para mejorar el proyecto se podrían implementar más funciones, como eliminación de nodos, visualizaciones graficas, mostrar propiedades como Nivel, Altura, Grado, etc.

7. Bibliografía

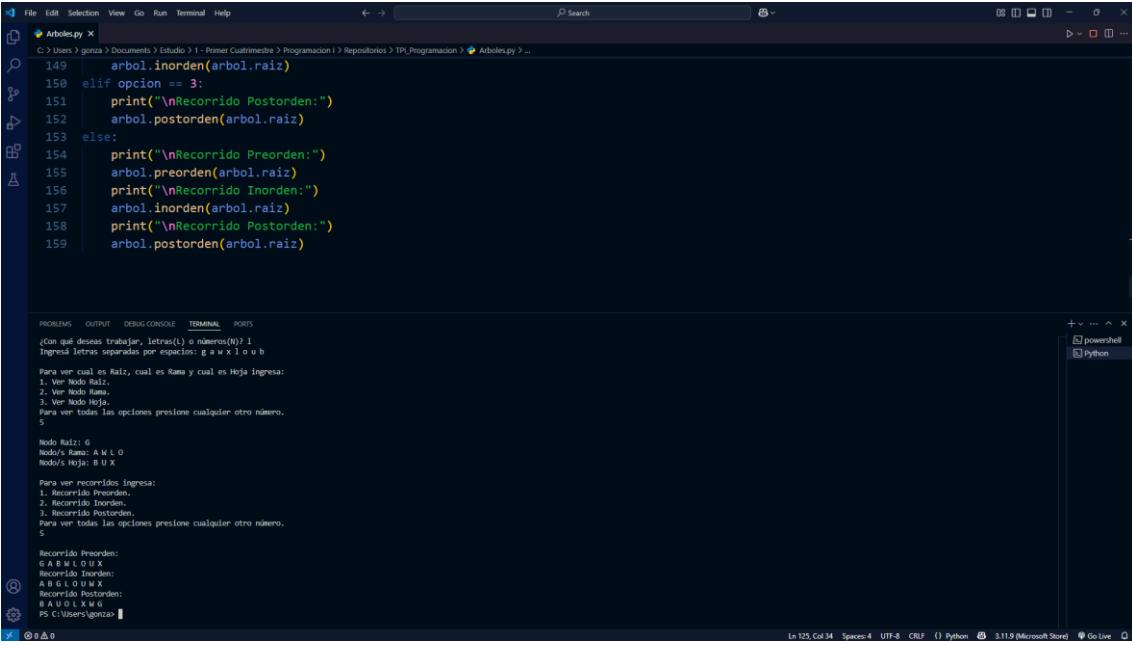
- Python Software Foundation: <https://docs.python.org/3/>
- Implementacion de árboles como listas anidadas:
<https://www.youtube.com/watch?v=-D4SxeHQGIg>
- Árboles (Parte 1, 2, 3 y 4):
https://www.youtube.com/watch?v=cVm51pO35qA&list=PLy5wpwhsM-2IIY-ge_fALJ4K_XAhLZ2I-&index=6
- Árboles:
https://docs.google.com/document/d/10k16oL15EeyOaq92aoi4qwK3t_22X29-FSV2iV-8N1U/edit?tab=t.0

8. Anexos

Video explicativo: <https://youtu.be/NQzK3QD88RM>

Enlace del Repositorio en GitHub: https://github.com/Goonza88/TPI_Programacion.git

Captura de VS CODE con el programa funcionando.



The screenshot shows a Microsoft Visual Studio Code interface. The left sidebar has a file icon and the path C:\Users\gonza\Documents\Estudio>1>Primer Cuatrimestre>Programacion I>Repositorios>TPI_Programacion>Arboles.py. The main editor area contains the following code:

```
149     arbol.inorden(arbol.raiz)
150     elif opcion == 3:
151         print("\nRecorrido Postorden:")
152         arbol.postorden(arbol.raiz)
153     else:
154         print("\nRecorrido Preorden:")
155         arbol.preorden(arbol.raiz)
156         print("\nRecorrido Inorden:")
157         arbol.inorden(arbol.raiz)
158         print("\nRecorrido Postorden:")
159         arbol.postorden(arbol.raiz)
```

Below the code, the terminal window displays the following interaction:

```
{Con qué deseas trabajar, Letras(l) o números(N)? 1
Ingresá letras separadas por espacios: g a w x l o u b
Para ver cual es Raiz, cual es Rama y cual es hoja ingresa:
1. Ver Nodo Raiz.
2. Ver Nodo Rama.
3. Ver Nodo Hoja.
Para ver todas las opciones presione cualquier otro número.
5
Nodo Raiz: G
Nodo/s Rama: A M L O
Nodo/s Hoja: U B W X
Para ver recorridos ingresa:
1. Recorrido Preorden.
2. Recorrido Inorden.
3. Recorrido Postorden.
Para ver todas las opciones presione cualquier otro número.
5
Recorrido Preorden:
G A B M L O U X
Recorrido Inorden:
A B G L O U M X
Recorrido Postorden:
B A U O L K M G
PS C:\Users\gonza>
```

The status bar at the bottom right shows "Line 125, Col 34" and "Python 3.11.9 (Microsoft Store)".