Título del trabajo: Arboles binarios y sus busquedas en Python
Alumnos: Lucas Ezequiel Amato – lucas.amato@tupad.utn.edu.ar
Materia: Programación I
Profesor/a: AUS Bruselario, Sebastián
Fecha de Entrega: 9/6/2025

Introducción:

En la programación se necesitan distintas maneras de organizar la información y luego poder buscar esa información de manera eficiente, guardar todo en listas no es un solución eficaz

En este trabajo se investigara los Árboles binarios en python, su implementación y porque son eficaces para organizar la información, se propone demostrar con un caso práctico el uso eficiente de estos arboles binarios

Marco Teórico:

¿Que es un árbol?

Un **árbol** es una **estructura de datos jerárquica** que se utiliza para organizar información de manera que cada elemento (llamado **nodo**) puede tener **cero o más nodos hijos**, pero **solo un nodo padre** (excepto la raíz).

Componentes de un árbol:

- **Raíz (root)**: Es el nodo principal, el primero del árbol. No tiene padre.
- **Nodos hijos (children)**: Son nodos que dependen o están conectados a un nodo superior (padre).
- **Nodos hoja (leaf nodes)**: Son nodos que no tienen hijos.
- **Subárbol**: Cualquier nodo con sus respectivos descendientes forma un subárbol.
- **Altura**: Número máximo de niveles desde la raíz hasta una hoja.

¿Qué es un árbol binario?

Un **árbol binario** es un tipo especial de árbol donde **cada nodo tiene como máximo dos hijos** (de ahí binario), que se llaman comúnmente: Hijo Izquierdo e Hijo Derecho

Maneras de recorrer un arbol binario:

Tipos de recorridos de un árbol binario:
In-Order (Izquierda - Raíz - Derecha)
Pre-Order (Raíz - Izquierda - Derecha)
Post-Order (Izquierda - Derecha - Raíz)

Recorrido Inorder: Los nodos del subárbol izquierdo se visitan primero, seguidos por el nodo raíz y el subárbol derecho. Este recorrido visita todos los nodos en orden de secuencia de claves no decreciente.

Recorrido Preorder: El nodo raíz se visita primero, seguido por los subárboles izquierdo y derecho.

Recorrido Postorder: Los nodos del subárbol izquierdo se visitan primero, seguidos por el subárbol derecho, y finalmente, la raíz.

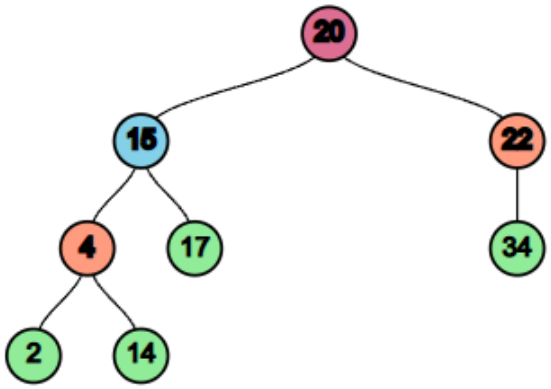
No hay una diferencia notable de velocidad para recorrer entre los 3, solo tienen diferencias en caso de uso para distintas operaciones, para dar un ejemplo Post-Order es mas adecuado para borrar nodos ya que la Raiz se visita a lo ultimo, evitando errores.

Uso Practico de un arbol binario

- Busqueda de informacion en un set de data muy grande
- Sistemas de archivos
- Compiladores

Caso práctico:

Se implementa un arbol binario y sus 3 tipos de búsquedas
Visualización del arbol binario representado en el codigo



```python
#Se crea el arbol binario
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key


#Esta funcion permite ingresar un nodo nuevo al arbol
def insert(root, key):
    if root is None:
        return Node(key)
    if root.val == key:
            return root
    if root.val < key:
            root.right = insert(root.right, key)
    else:
            root.left = insert(root.left, key)
    return root


# Funcion de recorrido "in order" ( Left -> Root -> Right)
def inorder(node):
    if node:
        inorder(node.left)
        print(node.val, end=" ")
        inorder(node.right)

# Funcion de recorrido "Pre Order" ( Root -> Left -> Right )
def preorder(node):
    if node is None:
        return
    print(node.val, end=' ')
    preorder(node.left)
    preorder(node.right)

# Funcion de recorrido "Post Order" (Left -> Right -> Root)

def postorder(node):
    if node is None:
        return
    postorder(node.left)
```

```
    postorder(node.right)
    print(node.val, end=' ')



r = Node(20) #Crea el arbol (y su primer nodo)
r = insert(r, 15)
r = insert(r, 4)
r = insert(r, 22)
r = insert(r, 34)
r = insert(r, 17)
r = insert(r, 2)
r = insert(r, 14)

# Imprimimos la busqueda "in order"
print("Recorrido in order:")
inorder(r)
print("\nRecorrido preorder:")
preorder(r)
print("\nRecorrido postorder:")
postorder(r)
```

Output de consola:

Recorrido in order:
2 4 14 15 17 20 22 34
Recorrido preorder:
20 15 4 2 14 17 22 34
Recorrido postorder:
2 14 4 17 15 34 22 20

Metodologia Utilizada:

Se investigo las diferencias entre los arboles y los arboles binarios, las diferentes formas de implementarlos, las distintas maneras de recorrerlos y sus diferencias en completar ese recorrido.

Resultados Obtenidos:

Se pudo construir el arbol binario en codigo y en base a ese arbol con unos valores manuales asignados comprobar el uso de los 3 tipos de recorrido

Se investigo la diferencia de velocidad entre los 3 , siendo esta la misma **O(n)**

Complejidad de Tiempo: O(n)

Cada nodo se visita exactamente una vez.

El trabajo realizado en cada nodo es constante O(1).

El tiempo total es proporcional al número de nodos.

Se puede ver en el output el orden de cada tipo de recorrido acorde

Conclusiones:

Aprender de estos arboles es muy util ya que nos permite liberarnos de las listas simples para poder hacer una busqueda y organizacion de volumenes de datos mas grandes, se pueden hacer aplicaciones mas performantes y ademas nos enseña en el caso de este trabajo a usar clases de otra manera.

Bibliografia:

https://www.w3schools.com/python/python_dsa_trees.asp
https://www.geeksforgeeks.org/binary-search-tree-in-python/
https://treeconverter.com/ - Visualizacion arbol
https://www.geeksforgeeks.org/introduction-to-binary-tree/