UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Integrador

Programación: Datos Avanzados

Alumnos

Gerbino Facundo, Pozo Felipe

-

Matemática

Tecnicatura Universitaria en Programación

Universidad Tecnológica Nacional

-

Docente Titular

Julieta Trapé, Nicolás Quirós

-

Docente Tutor

Miguel Barrera Oltra, Carla Bustos

-

Fecha de Entrega

09/06/25

.

01 - Introducción

El trabajo aborda la implementación de árboles binarios en Python, enfocándose en su construcción dinámica a partir de los datos ingresados por el usuario. La elección del tema responde a la necesidad de comprender las estructuras de datos jerárquicas, fundamentales en múltiples áreas de la programación: desde bases de datos y sistemas de archivos hasta en el desarrollo de algoritmos de inteligencia artificial. Se buscó demostrar cómo el árbol binario permite almacenar, ordenar y recorrer datos de forma eficiente, facilitando operaciones de búsqueda, clasificación y representación estructural. El objetivo principal fue diseñar un programa interactivo capaz de generar un árbol, recorrerlo en distintos órdenes y clasificar sus nodos, favoreciendo una comprensión práctica del modelo teórico subyacente.

02 - Marco Teorico

Un árbol binario es una estructura jerárquica compuesta por nodos conectados mediante referencias. Cada nodo contiene un valor y puede tener hasta dos hijos, uno a la izquierda y el otro a la derecha. El nodo superior se denomina raíz, los nodos sin hijos se llaman hojas y los intermedios ramas. Los árboles binarios de búsqueda (abb) mantienen una propiedad ordenada: todos los valores menores al nodo se ubican a su izquierda, y los mayores a su derecha. Los principales recorridos de un árbol binario son:

- **Preorden**: primero se visita la raíz, luego el árbol izquierdo y después el derecho. Este recorrido es útil para replicar la estructura del árbol.
- **Inorden**: primero se recorre el subárbol izquierdo, luego la raíz y finalmente el derecho. En abb, produce una lista ordenada de valores.
- **Postorden**: primero se recorren ambos subárboles y luego se visita la raíz. Se usa, por ejemplo, para liberar estructuras de memoria.

En Python, esta estructura se puede implementar de forma simple mediante clases anidadas y métodos recursivos, aprovechando la flexibilidad del lenguaje. La inserción compara el nuevo valor con el nodo actual y lo ubica a la izquierda o derecha según corresponda. Los recorridos generan listas concatenadas que representan el orden de visita de los nodos.

03 - Caso Práctico

El problema planteado consistio en construir un árbol binario a partir de entradas proporcionadas por el usuario en tiempo real, permitiendo luego recorrerlo en preorden, inorden y postorden. Además, se requiere que el árbol aceptara tanto números como letras, adaptando su comportamiento al tipo de dato ingresado.

Se desarrolló una clase **árbol** que permite insertar nuevos nodos y recorrer la estructura. El programa inicia solicitando un valor para la raíz y luego permite al usuario ingresar sucesivos valores hasta finalizar la construcción. Al terminar, se imprimen los tres recorridos principales. Cada método de recorrido está implementado de forma recursiva y devuelve una lista con el orden correspondiente.

El diseño se basó en la simplicidad y legibilidad del código, con énfasis en el entendimiento de la lógica estructural más que en optimizaciones de rendimiento. Se eligió no usar estructuras auxiliares, como colas o pilas, para priorizar la recursividad de la estructura.

```
## clase que genera el arbol
class Arbol:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = str(data)

    # Insertar un nodo en el árbol
    def insertar(self, data):
        data = str(data)
        if data < self.data:
            if self.left is None:
                self.left = Arbol(data)
            else:
                self.left.insertar(data)
        elif data > self.data:
```

```
            if self.right is None:
                self.right = Arbol(data)
            else:
                self.right.insertar(data)

    # Recorrido preorden: raíz → izquierda → derecha
    def preorden(self):
        result = [self.data]
        if self.left:
            result += self.left.preorden()
        if self.right:
            result += self.right.preorden()
        return result

    # Recorrido inorden: izquierda → raíz → derecha
    def inorden(self):
        result = []
        if self.left:
            result += self.left.inorden()
        result.append(self.data)
        if self.right:
            result += self.right.inorden()
        return result

    # Recorrido postorden: izquierda → derecha → raíz
    def postorden(self):
        result = []
        if self.left:
            result += self.left.postorden()
        if self.right:
            result += self.right.postorden()
        result.append(self.data)
        return result
```

```
# Crear raíz
raiz_usuario = input("Elige el nodo raíz: ")
root = Arbol(raiz_usuario)

# Bucle que le pregunta al usuario que secuencia quiere ingresar
while True:
    entrada = input("¿Quieres terminar el árbol? (yes) Si no,
ingresa el valor: ")
    if entrada.lower() == "yes":
        break
    else:
        root.insertar(entrada)

# Muestra los 3 resultados
print("\nRecorridos del árbol:")
print("Preorden :", " ".join(root.preorden()))
print("Inorden  :", " ".join(root.inorden()))
print("Postorden:", " ".join(root.postorden()))
```

04 - Metodología Utilizada

El desarrollo del trabajo siguió una serie de etapas bien definidas:

1. investigación conceptual sobre árboles binarios y sus recorridos. Se consultó documentación oficial de Python y recursos didácticos recomendados.
2. Diseño de la clase **arbol** definiendo sus atributos (valor, hijo izquierdo y derecho) y sus métodos (insertar, preorden, inorden, postorden)
3. Implementación del programa principal, donde se crea el árbol a partir de una raíz proporcionada por el usuario y se solicita la inserción de nodos adicionales en un bucle hasta que se indica la finalización.
4. Testeo interactivo del código con distintos tipos de datos, evaluando que los recorridos reflejen correctamente la estructura construida.

5. Revisión del código para asegurar claridad y coherencia, con formatos de indentación correctos.

El entorno utilizado fue VSCode, sin el uso de librerías externas.

05 - Resultados Obtenidos

El árbol binario fue construido de forma satisfactoria, insertando cada nuevo dato según su valor relativo a los nodos ya presentes. El programa demostró estabilidad al recibir tanto cadenas alfabéticas como valores numéricos, gracias a la conversión explícita a texto dentro del constructor.

Los recorridos generaron salidas coherentes con la estructura del árbol, por ejemplo, en una ejecución con los valores "m", "c", "z", "a", "f", "x", el inorden devolvió "a c f m x z", reflejando el orden alfabético, mientras que el preorden y el postorden mostraron recorrido según la raíz o las hojas, según el caso.

Se verificó que la estructura se comporta correctamente ante inserciones múltiples, sin sobrescritura ni errores de recursividad. El programa mostró capacidad de generalización para distintas entradas, sin necesidad de adaptar la lógica interna.

06 - Conclusiones

El trabajo permitió aplicar conocimientos sobre estructuras no lineales, recursividad y manejo dinámico de datos en Python. Se observó como los árboles binarios ofrecen una forma eficiente de representar jerarquías y facilitar búsquedas y ordenamientos parciales.

La implementación interactiva demostró que es posible construir estructuras complejas a partir de entradas simples, sin necesidad de bibliotecas avanzadas ni algoritmos complejos.

Como futura mejora, podría añadirse visualización gráfica del árbol, métodos para eliminar nodos o validación de duplicados. La principal dificultad fue asegurar el manejo correcto del input en tiempo real sin romper la lógica recursiva del árbol, especialmente al permitir tipos de datos variados. La solución fue normalizar la entrada a cadenas y mantener comparaciones consistentes.

07 - Bibliografía

- Python software foundation (2024). *Python 3 documentation*.
- W3schools (2024). *Python data structures - Trees*
- Cormen, Leiserson, Rivest, Stein (2009). I*ntroduction to algorithms*