TECNOLÓGICO NACIONAL DE MÉXICO

Baez Sauceda Jesus Arnoldo

Prof. Jose Mario Rios Felix

Inteligencia Artificial

Tarea 1

18 de Septiembre de 2025

```python
class Nodo:
    def __init__(self, nombre, valor=None):
        self.nombre = nombre
        self.valor = valor
        self.izquierdo = None
        self.derecho = None

    def __str__(self):
        return f"Nodo({self.nombre})"


class Arbol:
    def __init__(self):
        self.raiz = None

    def vacio(self):
        """Retorna True si el árbol está vacío, False en caso
contrario"""
        return self.raiz is None

    def buscarNodo(self, nombre):
        """Busca un nodo por nombre usando búsqueda en
profundidad (DFS)"""
        return self._buscarNodo_recursivo(self.raiz, nombre)

    def _buscarNodo_recursivo(self, nodo_actual, nombre):
        """Método recursivo auxiliar para buscar nodos"""
        if nodo_actual is None:
            return None

        if nodo_actual.nombre == nombre:
            return nodo_actual

        # Buscar en el subárbol izquierdo
        resultado_izquierdo =
self._buscarNodo_recursivo(nodo_actual.izquierdo, nombre)
        if resultado_izquierdo:
            return resultado_izquierdo

        # Buscar en el subárbol derecho
        resultado_derecho =
self._buscarNodo_recursivo(nodo_actual.derecho, nombre)
        return resultado_derecho

    def insertar(self, nombre, valor=None):
        """Inserta un nuevo nodo en el árbol manteniendo la
propiedad de BST"""
```

```
        nuevo_nodo = Nodo(nombre, valor)

        if self.vacio():
            self.raiz = nuevo_nodo
            return True

        return self._insertar_recursivo(self.raiz,
nuevo_nodo)

    def _insertar_recursivo(self, nodo_actual, nuevo_nodo):
        """Método recursivo auxiliar para insertar nodos"""

        if nuevo_nodo.nombre < nodo_actual.nombre:
            if nodo_actual.izquierdo is None:
                nodo_actual.izquierdo = nuevo_nodo
                return True
            else:
                return
self._insertar_recursivo(nodo_actual.izquierdo, nuevo_nodo)
        elif nuevo_nodo.nombre > nodo_actual.nombre:
            if nodo_actual.derecho is None:
                nodo_actual.derecho = nuevo_nodo
                return True
            else:
                return
self._insertar_recursivo(nodo_actual.derecho, nuevo_nodo)
        else:

            return False

    def imprimirArbol(self, tipo="inorden"):
        """Imprime el árbol en diferentes órdenes"""
        print(f"\nRecorrido {tipo}:")

        if tipo == "inorden":
            self._inorden(self.raiz)
        elif tipo == "preorden":
            self._preorden(self.raiz)
        elif tipo == "postorden":
            self._postorden(self.raiz)
        elif tipo == "niveles":
            self._por_niveles()
        else:
            print("Tipo de recorrido no válido. Use: inorden,
preorden, postorden, niveles")

    def _inorden(self, nodo):
```

```
        """Recorrido inorden: izquierdo - raíz - derecho"""
        if nodo:
            self._inorden(nodo.izquierdo)
            print(nodo.nombre, end=" ")
            self._inorden(nodo.derecho)

    def _preorden(self, nodo):
        """Recorrido preorden: raíz - izquierdo - derecho"""
        if nodo:
            print(nodo.nombre, end=" ")
            self._preorden(nodo.izquierdo)
            self._preorden(nodo.derecho)

    def _postorden(self, nodo):
        """Recorrido postorden: izquierdo - derecho - raíz"""
        if nodo:
            self._postorden(nodo.izquierdo)
            self._postorden(nodo.derecho)
            print(nodo.nombre, end=" ")

    def _por_niveles(self):
        """Recorrido por niveles (BFS)"""
        if self.vacio():
            return

        cola = [self.raiz]
        while cola:
            nodo_actual = cola.pop(0)
            print(nodo_actual.nombre, end=" ")

            if nodo_actual.izquierdo:
                cola.append(nodo_actual.izquierdo)
            if nodo_actual.derecho:
                cola.append(nodo_actual.derecho)

    def altura(self):
        """Calcula la altura del árbol"""
        return self._altura_recursiva(self.raiz)

    def _altura_recursiva(self, nodo):
        """Método recursivo para calcular altura"""
        if nodo is None:
            return 0
        return 1 +
max(self._altura_recursiva(nodo.izquierdo),
                    self._altura_recursiva(nodo.derecho))
```

```
    def contarNodos(self):
        """Cuenta la cantidad total de nodos"""
        return self._contarNodos_recursivo(self.raiz)

    def _contarNodos_recursivo(self, nodo):
        """Método recursivo para contar nodos"""
        if nodo is None:
            return 0
        return 1 +
self._contarNodos_recursivo(nodo.izquierdo) +
self._contarNodos_recursivo(nodo.derecho)


if __name__ == "__main__":
    arbol = Arbol()

    print(f"¿El árbol está vacío? {arbol.vacio()}")

    nodos = ["F", "B", "G", "A", "D", "I", "C", "E", "H"]
    for nombre in nodos:
        arbol.insertar(nombre)

    print(f"¿El árbol está vacío? {arbol.vacio()}")
    print(f"Altura del árbol: {arbol.altura()}")
    print(f"Total de nodos: {arbol.contarNodos()}")

    busquedas = ["F", "A", "X", "H"]
    for nombre in busquedas:
        resultado = arbol.buscarNodo(nombre)
        if resultado:
            print(f"Nodo '{nombre}' encontrado: {resultado}")
        else:
            print(f"Nodo '{nombre}' no encontrado")

    arbol.imprimirArbol("inorden")  # A B C D E F G H I
    arbol.imprimirArbol("preorden")  # F B A D C E G I H
    arbol.imprimirArbol("postorden")  # A C E D B H I G F
    arbol.imprimirArbol("niveles")  # F B G A D I C E H

    print(f"\nIntentando insertar nodo 'B' (duplicado):
{arbol.insertar('B')}")

    print(f"Insertando nodo 'J': {arbol.insertar('J')}")
    arbol.imprimirArbol("inorden")
```