Universidad de Cundinamarca

Estructura de Datos

R3-A5-S15 Reforestando

R3-A6-S16 Ruta Óptima

Javier Mateo Barrero Vanegas

Jim Alejandro Quiñones Martínez

Luis Ángel Martínez Cuenca

Harick Yesid Villarraga Rincon

Cristian Esteban Ruiz Parra

Docente: Dilia Ines Molina Cubillos

16 de noviembre de 2025

1. Introducción

El presente informe aborda el estudio de dos pilares fundamentales en la ciencia de la computación y la representación formal: las notaciones matemáticas y las estructuras de datos jerárquicas conocidas como árboles. La capacidad de una máquina para procesar y evaluar expresiones algebraicas de manera eficiente depende de la notación utilizada, mientras que la organización de la información y la optimización de los algoritmos de búsqueda recaen en las estructuras arbóreas. La comprensión de la Notación Polaca (Prefija), la Notación Polaca Inversa (Postfija) y la Notación Infija es esencial para el diseño de compiladores y evaluadores de expresiones, y su relación canónica con los recorridos de los Árboles de Expresión Binarios subraya su interdependencia. Así mismo, el análisis de los árboles, desde los N-arios hasta los auto-balanceados (AVL), es crucial para la gestión eficiente de la memoria y el rendimiento en sistemas de bases de datos y algoritmos avanzados.

2. Objetivos del Estudio

Objetivo General

Analizar y comprender la interconexión fundamental entre las notaciones matemáticas formales (Infija, Prefija, Postfija) y las estructuras de datos jerárquicas (Árboles), identificando cómo esta relación es explotada en la algoritmia y la ciencia de la computación para el procesamiento eficiente y no ambiguo de la información.

Objetivos Específicos

1. Definir y contrastar las características, reglas sintácticas y usos de las notaciones Infija, Prefija y Postfija para la representación de expresiones algebraicas.
2. Establecer los conceptos fundamentales de los árboles como estructuras de datos no lineales, describiendo su terminología clave y sus aplicaciones estratégicas en la informática.
3. Estudiar los tipos de árboles más relevantes (N-arios, Binarios y AVL), enfocándose en las propiedades de ordenamiento y balanceo que optimizan la eficiencia algorítmica.
4. Explicar la relación crítica entre las notaciones (a través de sus recorridos) y el Árbol de Expresión Binario, como base para el diseño de algoritmos de conversión y evaluación de expresiones.

3. Notaciones Matemáticas: Fundamentos y Propiedades

El desarrollo de la matemática y la ciencia de la computación depende intrínsecamente de un sistema de comunicación riguroso y universal. Las notaciones matemáticas constituyen este lenguaje formal, esencial para la expresión concisa de conceptos complejos y la manipulación algorítmica.

3.1. Definición y Rol de la Notación Matemática

Definición: La notación matemática se define como un sistema simbólico formal compuesto por símbolos y reglas sintácticas rigurosas , diseñado para transmitir ideas y conceptos matemáticos con una precisión inigualable. Este sistema opera como un lenguaje estandarizado que permite la manipulación eficiente de argumentos complejos.

Propósito: Su objetivo fundamental es comunicar ideas de manera clara, precisa y sin ambigüedad mediante símbolos con valor propio, no meras abreviaturas. Además, facilita el razonamiento rápido al permitir que los profesionales realicen cálculos abstractos utilizando una iconografía algebraica familiar.

3.2. Características Intrínsecas de las Notaciones Formales

Las propiedades inherentes de la notación matemática son las que le confieren su poder como herramienta científica global, superando las limitaciones del lenguaje natural.

Universalidad y Estandarización

La notación matemática es un lenguaje universal que asegura que las ideas complejas se comuniquen de manera estandarizada. Esta estandarización es vital para la colaboración, garantizando que un símbolo o expresión tenga el mismo significado preciso para un matemático o ingeniero en cualquier parte del mundo.

Concisión y Eficiencia

Una de las ventajas más significativas es su brevedad. La notación es notablemente más corta que la expresión correspondiente en lenguaje natural. Esta concisión acelera la formulación de nuevas declaraciones inequívocas, permitiendo construir rápidamente argumentos o estructuras algorítmicas.

Precisión y Determinismo (Minimización de Ambigüedad)

La notación se basa en reglas sintácticas que establecen un determinismo esencial para el análisis sintáctico (parsing) y la ejecución algorítmica en la computación. En áreas como la lógica y los cálculos, esta estructura formal reduce significativamente la ambigüedad en comparación con el lenguaje natural.

3.3. Aplicaciones Generales en la Tecnología y la Ingeniería

Las notaciones se usan en toda disciplina matemática para facilitar el análisis, la comunicación y el cálculo.

Notación Científica

La notación científica ($m \times 10^n$) es esencial para la representación eficiente de números muy grandes o muy pequeños. Se compone de una "mantisa" (m), con una sola cifra entera distinta de cero ($1 \leq m \leq 10$), y el "orden de magnitud" (10^n).

Se utiliza ampliamente en:

- Especificaciones Técnicas: Para describir capacidades de hardware (microprocesadores) o magnitudes en biotecnología.
- Programación y Análisis de Datos: Permite optimizar cálculos complejos, definir constantes y simplificar la conversión de unidades.

Lenguajes Formales y Compiladores

En la teoría de la computación, las notaciones formales definen la estructura y las reglas de los lenguajes. Por ejemplo, las expresiones regulares (RE) son una notación fundamental utilizada para denotar lenguajes formales y son aplicadas por analizadores léxicos, los cuales dividen el código fuente en *tokens* (como operadores o palabras clave). Este uso de notaciones garantiza el determinismo necesario para que los compiladores traduzcan el código humano a instrucciones de máquina.

4. Tipos de Notaciones para Expresiones Algebraicas

Las notaciones Infija, Postfija y Prefija ofrecen diferentes estrategias para estructurar expresiones aritméticas o lógicas. El uso de notaciones no ambiguas (Prefija y Postfija) es fundamental en compiladores y programación para facilitar la evaluación y el procesamiento de fórmulas.

4.1. Notación Infija (Infix Notation)

Es la notación más común y conocida, donde los operadores se escriben entre sus operandos.

- Formato Tradicional: Operando - Operador - Operando.
- Ejemplo: A + B.
- Características: Es la forma más fácil de leer para los humanos, pero es inherentemente ambigua. Requiere el uso de paréntesis y reglas de precedencia para definir el orden de ejecución.

4.2. Notación Postfija (Notación Polaca Inversa - RPN)

En esta notación, los operadores se colocan después de sus operandos (Operando 1, Operando 2, Operador). Es la sintaxis ideal para la evaluación en sistemas informáticos.

- Formato Funcional: Operando - Operando - Operador.
- Ejemplo: AB + (equivalente a A + B).
- Características: Elimina la ambigüedad de precedencia y la necesidad de paréntesis. La evaluación es lineal, eficiente y se realiza utilizando una pila.
- Relación con Árboles: Se obtiene mediante un recorrido Postorden (Izquierda-Derecha-Raíz) del Árbol de Expresión Binario.

4.3. Notación Prefija (Notación Polaca)

En la notación prefija, el operador se sitúa antes de sus operandos (Operador, Operando 1, Operando 2).

- Formato Canónico: Operador - Operando - Operando.
- Ejemplo: + AB (equivalente a A + B).
- Características: Al igual que la postfija, elimina la ambigüedad de precedencia.
- Relación con Árboles: Se obtiene mediante un recorrido Preorden (Raíz-Izquierda-Derecha) del Árbol de Expresión Binario.

Table 1: Comparación Estructural y de Recorridos

Criterio	Notación Infija	Notación Prefija (Polaca)	Notación Postfija (RPN)
Orden	Operando - Operador - Operando	Operador - Operando - Operando	Operando - Operando - Operador
Ejemplo	(2 + 3) * 4	* + 234	23 + 4 *
Recorrido del Árbol	Inorden	Preorden	Postorden
Paréntesis	Requeridos	No requeridos	No requeridos
Evaluación Computacional	Compleja (depende de precedencia)	Simple (Uso de pila)	Muy eficiente (Uso de pila)

5. Conceptos Fundamentales de Estructuras de Árbol

Los árboles son estructuras de datos no lineales fundamentales que permiten representar y gestionar información jerárquica con una eficiencia logarítmica.

5.1. Definición Formal de Árbol como Estructura de Datos No Lineal

Definición: Un árbol es una estructura de datos jerárquica compuesta por nodos interrelacionados y conectados por aristas. Se define recursivamente: consiste en un nodo raíz junto con cero o más subárboles disjuntos, cada uno de los cuales es un árbol por sí mismo.

Características Principales:

- Nodo Raíz: El nodo inicial, el único que no tiene un padre.
- Nodo Hoja (Terminal): Nodos que no poseen ningún hijo.
- Arista: La conexión que une a un nodo padre con un nodo hijo.
- Profundidad: La longitud del camino único (número de aristas) desde la raíz del árbol hasta un nodo específico.
- Altura del Árbol: Es la altura de su nodo raíz, medida como la longitud del camino más largo desde la raíz hasta una hoja.
- Ausencia de Ciclos: Una característica definitoria es que no existen ciclos en la estructura.

5.2. Usos Estratégicos de los Árboles en la Informática

Los árboles son fundamentales en informática para representar estructuras de datos, expresiones algebraicas, organización de información y toma de decisiones.

Sistemas de Archivos y Sistemas Operativos

Modelan naturalmente la estructura de directorios en los sistemas operativos, donde el directorio raíz actúa como el nodo raíz del árbol. Los sistemas de archivos avanzados, como NTFS, utilizan estructuras basadas en árboles (B-Trees) para optimizar la gestión de archivos.

Indexación y Bases de Datos

Los árboles de búsqueda N-arios (como los B-Trees) son estructuras vitales para la gestión eficiente de grandes volúmenes de datos. Su diseño minimiza las operaciones de Entrada/Salida (I/O) de disco y garantiza funciones de búsqueda, inserción y eliminación eficientes con una complejidad temporal logarítmica (O(log n)).

Compiladores y Árboles de Expresión

Los compiladores utilizan Árboles de Expresión Binarios para representar la estructura lógica de las fórmulas matemáticas. En esta estructura, los nodos internos representan operadores y los nodos hoja representan operandos. Esta representación jerárquica es fundamental para el análisis semántico y la evaluación precisa de las operaciones.

Inteligencia Artificial y Modelado

Los Árboles de Decisión son ampliamente utilizados en el campo de la Inteligencia Artificial (IA) y el aprendizaje automático como modelos de clasificación y regresión. Proporcionan una representación jerárquica clara de la toma de decisiones basada en reglas.

6. Tipos Especializados de Árboles

La clasificación de los árboles depende de las restricciones que se imponen sobre el número de hijos por nodo y el ordenamiento de los datos, lo que influye directamente en su rendimiento y aplicación.

6.1. Árboles N-arios (Multirrama)

Un árbol N-ario, o k-ario, es una estructura en la que cada nodo puede tener un número máximo de N o k hijos. El árbol binario es el caso especial donde N=2.

- Usos Clave: Son útiles para modelar estructuras jerárquicas naturales donde un elemento puede tener múltiples hijos, como los sistemas de archivos (directorios) o los índices de bases de datos (B-Trees).

6.2. Árboles Binarios (N=2)

El árbol binario es el caso más simple y estudiado, donde cada nodo tiene a lo sumo dos hijos, denominados hijo izquierdo y derecho.

- Árbol Binario de Búsqueda (BST): Impone una restricción de orden sobre los datos. Para cualquier nodo, su clave es mayor que todas las claves en su subárbol izquierdo y menor que todas las claves en su subárbol derecho.
- Recorridos: Los recorridos en profundidad son clave para la serialización y la obtención de notaciones :
    - Preorden: Raíz, Izquierda, Derecha (produce notación Prefija).
    - Inorden: Izquierda, Raíz, Derecha (produce notación Infija y datos ordenados en un BST).
    - Postorden: Izquierda, Derecha, Raíz (produce notación Postfija).

6.3. Árboles AVL (Adelson-Velsky y Landis)

Los árboles AVL son árboles binarios de búsqueda auto-balanceados. Fueron el primer tipo de BST balanceado, diseñados para garantizar que las operaciones de búsqueda se mantengan eficientes (O(log n)).

- Factor de Equilibrio (FE): Un AVL mantiene la propiedad de que para cada nodo, la diferencia de altura entre su subárbol derecho y su subárbol izquierdo es como máximo uno.

FE = Altura Derecha - Altura Izquierda

Un nodo es considerado balanceado si su Factor de Equilibrio es -1, 0, o +1.

- Rotaciones: Si el factor de equilibrio se vuelve 2 o -2 (indicando un desequilibrio), el árbol se reestructura mediante rotaciones (Simples o Dobles) para restaurar la propiedad AVL. Los casos principales incluyen:
    - Rotación Simple Izquierda (RSI) / Derecha (RSD): Se aplican cuando el desequilibrio y el signo del hijo son consistentes.
    - Rotación Doble Izquierda (RDI) / Derecha (RDD): Se aplican cuando el desequilibrio y el signo del hijo son opuestos (patrón zigzag).

El rebalanceo asegura que la altura del árbol nunca se degrade, manteniendo el rendimiento óptimo de las operaciones de búsqueda y modificación.

7. Síntesis y Relación Crítica: Árboles de Expresión

La interconexión entre las notaciones formales y las estructuras de árbol es fundamental en la informática, especialmente en el diseño de compiladores.

7.1. Construcción de Árboles a partir de Notación Postfija (RPN)

La notación Postfija es la entrada óptima para la construcción algorítmica de un Árbol de Expresión Binario. El proceso utiliza una pila para gestionar los subárboles :

1. Recorrido: Se lee la expresión Postfija de izquierda a derecha.
2. Operandos: Se crea un nodo (subárbol simple) por cada operando y se apila.
3. Operadores: Cuando se encuentra un operador, se desapilan los dos subárboles superiores. Estos se convierten en los hijos (el primero desapilado es el hijo derecho y el segundo es el hijo izquierdo) del nuevo nodo operador.
4. Apilamiento: El nuevo nodo operador (la raíz del subárbol más grande) se apila de nuevo.

Este proceso permite que la máquina traduzca una secuencia lineal de *tokens* (Postfija) en una estructura de datos jerárquica (el Árbol de Expresión) que encapsula el orden de ejecución sin ambigüedad.

7.2. El Mapeo Canónico de Recorridos y Notaciones

La relación directa entre los recorridos en profundidad y las notaciones formaliza cómo el árbol de expresión representa la fórmula.

Recorrido del Árbol	Notación Resultante	Orden de Lectura
Preorden	Prefija	Raíz - Izquierda - Derecha
Inorden	Infija	Izquierda - Raíz - Derecha
Postorden	Postfija	Izquierda - Derecha - Raíz

Esta correspondencia demuestra que las notaciones no ambiguas (Prefija y Postfija) son simplemente serializaciones lineales que prescriben el orden de evaluación dictado por la jerarquía del árbol.

8. Marco Teórico

8.1. Explicación del código

1. **Clase nodo**

```
class Nodo:
    def __init__(self, dato):
        self.dato = dato
        self.izquierdo = None
        self.derecho = None
```

Esta clase representa cada nodo individual del árbol binario. Un nodo es la unidad básica de almacenamiento en un árbol y contiene tres elementos fundamentales:

- **dato**: almacena el valor numérico del nodo
- **izquierdo**: es un apuntador (referencia) al nodo hijo izquierdo. Inicialmente es None (vacío)
- **derecho**: es un apuntador al nodo hijo derecho. También inicia en None

Esta estructura permite conectar los nodos entre sí formando la estructura del árbol.

2. **Clase arbolbinario - constructor**

```
class ArbolBinario:
    def __init__(self):
        self.raiz = None
```

Esta es la clase principal que gestiona todo el árbol binario. El constructor inicializa el árbol con la raíz en None, lo que indica que el árbol está vacío al crearse. La raíz es el nodo principal desde donde se despliegan todos los demás nodos.

3. **Método insertar**

```
def insertar(self, dato):
    if self.raiz is None:
        self.raiz = Nodo(dato)
        print(f"Dato {dato} insertado como raiz")
    else:
        self._insertar_recursivo(self.raiz, dato)
```

Este método público permite agregar nuevos elementos al árbol. primero verifica si el árbol está vacío (raíz es none), en cuyo caso el nuevo dato se convierte en la raíz. si ya existe una raíz, delega la tarea al método recursivo para encontrar la posición correcta según las reglas del árbol binario de búsqueda.

4. **Método insertar recursivo**

```
def _insertar_recursivo(self, nodo, dato):
    if dato < nodo.dato:
        if nodo.izquierdo is None:
            nodo.izquierdo = Nodo(dato)
            print(f"Dato {dato} insertado")
        else:
            self._insertar_recursivo(nodo.izquierdo, dato)
    elif dato > nodo.dato:
        if nodo.derecho is None:
            nodo.derecho = Nodo(dato)
            print(f"Dato {dato} insertado")
        else:
            self._insertar_recursivo(nodo.derecho, dato)
    else:
        print(f"El dato {dato} ya existe en el arbol")
```

Este método auxiliar implementa la lógica de inserción ordenada del árbol binario de búsqueda:

- Si el dato a insertar es **menor** que el nodo actual, se intenta colocar en el subárbol izquierdo
- Si es **mayor**, se coloca en el subárbol derecho
- Si es **igual**, se rechaza porque no se permiten duplicados

El método se llama a sí mismo recursivamente hasta encontrar una posición vacía (None) donde insertar el nuevo nodo.

5. **Recorrido in-orden**

```
def inorden(self):
    print("\nRecorrido In-orden:")
    if self.raiz is None:
        print("El arbol esta vacio")
    else:
        self._inorden_recursivo(self.raiz)
        print()

def _inorden_recursivo(self, nodo):
    if nodo:
        self._inorden_recursivo(nodo.izquierdo)
        print(nodo.dato, end=" ")
        self._inorden_recursivo(nodo.derecho)
```

El recorrido in-orden sigue el patrón: **izquierda → raíz → derecha**

este recorrido tiene una característica importante: en un árbol binario de búsqueda, imprime los elementos en orden ascendente. primero visita recursivamente todo el subárbol izquierdo, luego imprime el nodo actual, y finalmente recorre el subárbol derecho.

6. **Recorrido post-orden**

```
def postorden(self):
    print("\nRecorrido Post-orden:")
    if self.raiz is None:
        print("El arbol esta vacio")
    else:
        self._postorden_recursivo(self.raiz)
        print()

def _postorden_recursivo(self, nodo):
    if nodo:
        self._postorden_recursivo(nodo.izquierdo)
        self._postorden_recursivo(nodo.derecho)
        print(nodo.dato, end=" ")
```

El recorrido post-orden sigue el patrón: **Izquierda → Derecha → Raíz**

Este recorrido visita primero todos los hijos antes de visitar el padre. Es útil para operaciones donde se necesita procesar los nodos hijos antes que los padres, como por ejemplo eliminar un árbol desde las hojas hacia la raíz.

7. **Recorrido pre-orden**

```
def preorden(self):
    print("\nRecorrido Pre-orden:")
    if self.raiz is None:
        print("El arbol esta vacio")
    else:
        self._preorden_recursivo(self.raiz)
        print()

def _preorden_recursivo(self, nodo):
    if nodo:
        print(nodo.dato, end=" ")
        self._preorden_recursivo(nodo.izquierdo)
        self._preorden_recursivo(nodo.derecho)
```

El recorrido pre-orden sigue el patrón: **Raíz → Izquierda → Derecha**

Este recorrido visita primero el nodo padre antes que sus hijos. Es útil para crear una copia del árbol o para evaluar expresiones en notación prefija.

8. **Función mostrar menú**

```
def mostrar_menu():
    print("\n========================================")
    print("   ARBOL BINARIO ORDENADO")
    print("========================================")
    print("1. Insertar dato")
    print("2. Imprimir en In-orden")
    print("3. Imprimir en Post-orden")
    print("4. Imprimir en Pre-orden")
    print("5. Salir")
    print("========================================")
```

Función auxiliar que despliega el menú de opciones en consola. Presenta de forma clara las 5 operaciones disponibles para el usuario. Utiliza caracteres "=" para darle formato visual al menú.

9. **Función principal (main)**

```
def main():
    arbol = ArbolBinario()

    while True:
        mostrar_menu()
        opcion = input("\nSeleccione una opcion (1-5): ")

        if opcion == "1":
            try:
                dato = int(input("Ingrese el numero a insertar: "))
                arbol.insertar(dato)
            except:
                print("Error: debe ingresar un numero valido")

        elif opcion == "2":
            arbol.inorden()

        elif opcion == "3":
            arbol.postorden()

        elif opcion == "4":
            arbol.preorden()

        elif opcion == "5":
            print("\nGracias por usar el programa!")
            break

        else:
            print("\nOpcion no valida, intente de nuevo")
```

Es el punto de entrada del programa. Controla el flujo principal mediante un ciclo infinito que:

1. Muestra el menú al usuario
2. Captura la opción seleccionada

3. Ejecuta la operación correspondiente mediante una estructura if-elif
4. Incluye manejo básico de errores con try-except para validar la entrada numérica
5. El ciclo termina cuando el usuario selecciona la opción 5 (salir)

Cada opción del menú llama a los métodos correspondientes de la clase ArbolBinario.

10. Punto de ejecución

```
if __name__ == "__main__":
    main()
```

Esta línea verifica si el archivo se está ejecutando directamente (no importado como módulo). Si es así, llama a la función main() para iniciar el programa. Es una buena práctica en Python que permite reutilizar el código en otros programas sin ejecutar automáticamente la función principal.

8.2. Código Completo

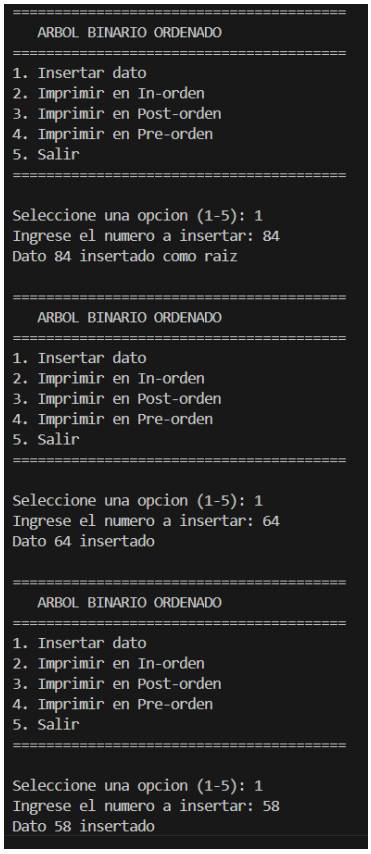
```
# Clase para crear los nodos del arbol
class Nodo:
    def __init__(self, dato):
        self.dato = dato  # valor que guarda el nodo
        self.izquierdo = None  # apuntador al hijo izquierdo
        self.derecho = None  # apuntador al hijo derecho


# Clase principal del arbol binario
class ArbolBinario:
    def __init__(self):
        self.raiz = None  # inicialmente el arbol esta vacio

    # Metodo para insertar un nuevo dato en el arbol
    def insertar(self, dato):
        if self.raiz is None:
            # si el arbol esta vacio, el nuevo nodo es la raiz
            self.raiz = Nodo(dato)
            print(f"Dato {dato} insertado como raiz")
        else:
            # si ya hay raiz, llamamos al metodo recursivo
            self._insertar_recursivo(self.raiz, dato)

    # Metodo auxiliar recursivo para insertar
    def _insertar_recursivo(self, nodo, dato):
        if dato < nodo.dato:
            # si el dato es menor, va a la izquierda
            if nodo.izquierdo is None:
                nodo.izquierdo = Nodo(dato)
```

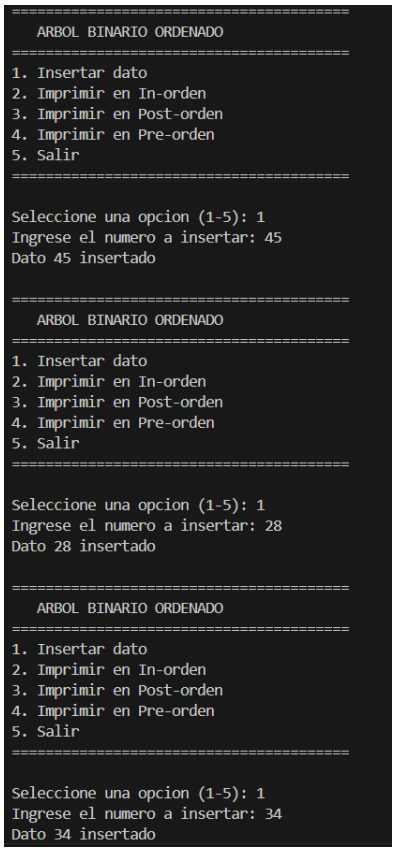
```
                print(f"Dato {dato} insertado")
            else:
                self._insertar_recursivo(nodo.izquierdo, dato)
        elif dato > nodo.dato:
            # si el dato es mayor, va a la derecha
            if nodo.derecho is None:
                nodo.derecho = Nodo(dato)
                print(f"Dato {dato} insertado")
            else:
                self._insertar_recursivo(nodo.derecho, dato)
        else:
            # si el dato ya existe no lo insertamos
            print(f"El dato {dato} ya existe en el arbol")

    # Recorrido inorden: Izquierda -> Raiz -> Derecha
    def inorden(self):
        print("\nRecorrido In-orden:")
        if self.raiz is None:
            print("El arbol esta vacio")
        else:
            self._inorden_recursivo(self.raiz)
            print()  # salto de linea al final

    def _inorden_recursivo(self, nodo):
        if nodo:
            self._inorden_recursivo(nodo.izquierdo)  # primero el
subarbol izquierdo
            print(nodo.dato, end=" ")  # luego imprimimos la raiz
            self._inorden_recursivo(nodo.derecho)  # finalmente el
subarbol derecho

    # Recorrido postorden: Izquierda -> Derecha -> Raiz
    def postorden(self):
        print("\nRecorrido Post-orden:")
        if self.raiz is None:
            print("El arbol esta vacio")
        else:
            self._postorden_recursivo(self.raiz)
            print()

    def _postorden_recursivo(self, nodo):
        if nodo:
            self._postorden_recursivo(nodo.izquierdo)  # primero
izquierda
            self._postorden_recursivo(nodo.derecho)  # luego derecha
```

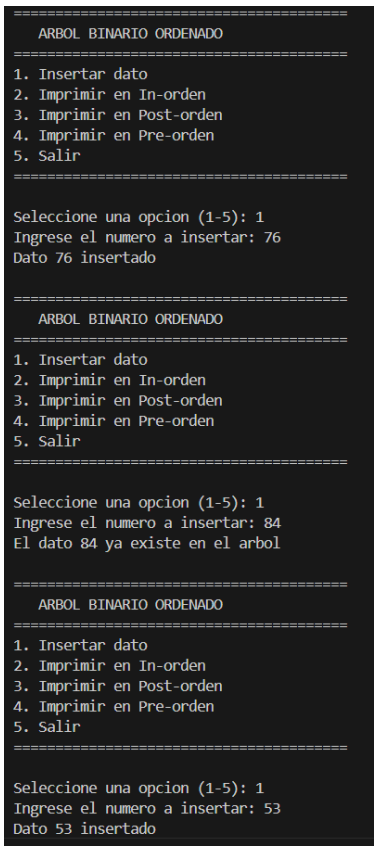
```
            print(nodo.dato, end=" ")  # al final la raiz

    # Recorrido preorden: Raiz -> Izquierda -> Derecha
    def preorden(self):
        print("\nRecorrido Pre-orden:")
        if self.raiz is None:
            print("El arbol esta vacio")
        else:
            self._preorden_recursivo(self.raiz)
            print()

    def _preorden_recursivo(self, nodo):
        if nodo:
            print(nodo.dato, end=" ")  # primero imprimimos la raiz
            self._preorden_recursivo(nodo.izquierdo)  # luego izquierda
            self._preorden_recursivo(nodo.derecho)  # y por ultimo
derecha


# Funcion que muestra el menu principal
def mostrar_menu():
    print("\n========================================")
    print("   ARBOL BINARIO ORDENADO")
    print("========================================")
    print("1. Insertar dato")
    print("2. Imprimir en In-orden")
    print("3. Imprimir en Post-orden")
    print("4. Imprimir en Pre-orden")
    print("5. Salir")
    print("========================================")


# Funcion principal del programa
def main():
    arbol = ArbolBinario()  # creamos un arbol vacio

    # ciclo principal del programa
    while True:
        mostrar_menu()
        opcion = input("\nSeleccione una opcion (1-5): ")

        # opcion 1: insertar dato
        if opcion == "1":
            try:
                dato = int(input("Ingrese el numero a insertar: "))
```

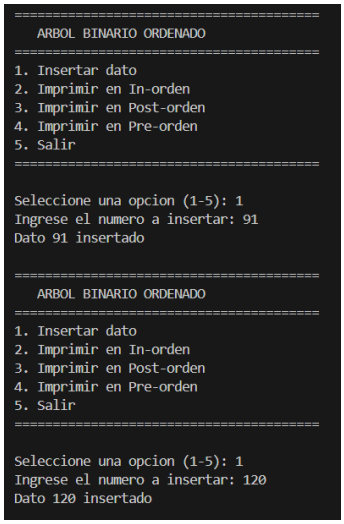
```
                arbol.insertar(dato)
            except:
                print("Error: debe ingresar un numero valido")

        # opcion 2: mostrar inorden
        elif opcion == "2":
            arbol.inorden()

        # opcion 3: mostrar postorden
        elif opcion == "3":
            arbol.postorden()

        # opcion 4: mostrar preorden
        elif opcion == "4":
            arbol.preorden()

        # opcion 5: salir del programa
        elif opcion == "5":
            print("\nGracias por usar el programa!")
            break

        # si ingresa una opcion invalida
        else:
            print("\nOpcion no valida, intente de nuevo")


# ejecutar el programa
if __name__ == "__main__":
    main()
```

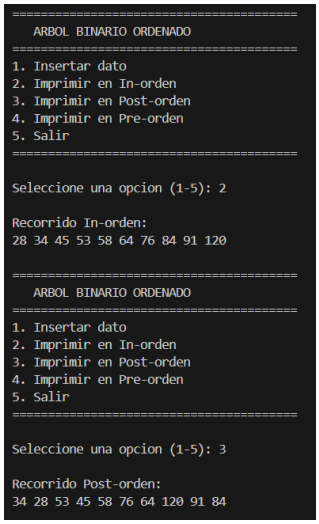
8.3. Funcionamiento

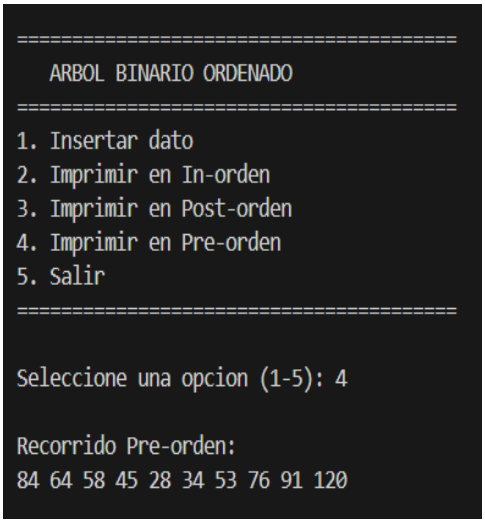












9. Conclusiones

El estudio de las notaciones formales y las estructuras de árbol revela una sinergia indispensable para la ingeniería de software y la algoritmia.

1. Eliminación de Ambigüedad y Eficiencia: Las notaciones Postfija (RPN) y Prefija, al eliminar la dependencia de paréntesis y reglas de precedencia, son las formas de representación más eficientes para el procesamiento interno de las computadoras. La implementación de una pila permite evaluar la notación Postfija de manera lineal, siendo un proceso crucial en la fase de análisis sintáctico de compiladores.

2. La Importancia de la Jerarquía Estructural: Los árboles de expresión (o Árboles de Sintaxis Abstracta) son la representación gráfica que conecta las notaciones lineales con la estructura jerárquica de la expresión. Los recorridos Postorden, Preorden e Inorden en este árbol se mapean directamente a las notaciones Postfija, Prefija e Infija, respectivamente, validando la estructura como un modelo canónico de las operaciones.

3. Garantía de Rendimiento Logarítmico: La utilización de árboles de búsqueda especializados, como los Árboles AVL, es vital en sistemas de datos dinámicos. Estos garantizan un balanceo estricto al mantener el factor de equilibrio de todos sus nodos en $\pm$ 1 o 0. Esta propiedad asegura que las operaciones críticas (búsqueda, inserción, eliminación) mantengan una complejidad temporal óptima de O(log n), evitando la degeneración del árbol a una lista ligada.

4. Modelado de Sistemas a Gran Escala: La generalización de los árboles a estructuras N-arias (como los B-Trees) permite el modelado eficiente de jerarquías complejas y la optimización de accesos a memoria secundaria, siendo la base de la indexación en sistemas de archivos y bases de datos a gran escala**.**

10. Referencias

1. Contenido 2. Notación Matemática | PDF | Sustracción - Scribd, fecha de acceso: noviembre 16, 2025, https://es.scribd.com/document/876881516/2

2. LA INFLUENCIA DEL DEL LENGUAJE MATEMÁTICO EN EL PROCESO DE APRENDIZAJE DE LAS MATEMÁTICAS - Ciencia Latina Revista Científica Multidisciplinar, fecha de acceso: noviembre 16, 2025, https://ciencialatina.org/index.php/cienciala/article/download/11795/17185/

3. ¿Una notación realmente hace que las matemáticas sean menos ambiguas? - Reddit, fecha de acceso: noviembre 16, 2025, https://www.reddit.com/r/learnmath/comments/14fearw/does_a_notation_really_make_mathematics_less/?tl=es-es

4. Aplicaciones de Arboles | PDF | Ingeniería Informática | Datos - Scribd, fecha de acceso: noviembre 16, 2025, https://es.scribd.com/document/370913096/Aplicaciones-de-Arboles

5. Notación polaca inversa - Wikipedia, la enciclopedia libre, fecha de acceso: noviembre 16, 2025, https://es.wikipedia.org/wiki/Notaci%C3%B3n_polaca_inversa

6. Notación científica - Wikipedia, la enciclopedia libre, fecha de acceso: noviembre 16, 2025, https://es.wikipedia.org/wiki/Notaci%C3%B3n_cient%C3%ADfica

7. Indexación de árbol B vs. Indexación de hash vs. Indexación de gráficos: ¿Cuál es la adecuada para tu base de datos? - MyScale, fecha de acceso: noviembre 16, 2025, https://myscale.com/blog/es/b-tree-vs-hash-indexing-right-for-database

8. Notación Científica: ¿qué es? | Lenovo México, fecha de acceso: noviembre 16, 2025, https://www.lenovo.com/mx/es/glosario/notacion-cientifica/

9. Decision tree - KeyTrends, fecha de acceso: noviembre 16, 2025, https://keytrends.ai/es/academy/glosario/inteligencia-artificial/decision-tree

10. Ciencias computacionales Propedeutico: Teor´ıa de Autómatas y Lenguajes Formales Expresiones regulares y lenguajes - INAOE, fecha de acceso: noviembre 16, 2025, https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Automatas/03_Automatas_ExpresionesRegularesLenguajes/CAPTUL1.PDF

11. fecha de acceso: noviembre 16, 2025, https://www.utm.mx/~mgarcia/arboles.pdf

12. Notación infija, prefija y postfija de una expresión matemática - YouTube, fecha de acceso: noviembre 16, 2025, https://www.youtube.com/watch?v=g2JkTm9ZwU4

13. Ejercicios de Notaciones en Arboles-Team | PDF | Informática | Matemáticas - Scribd, fecha de acceso: noviembre 16, 2025, https://es.scribd.com/document/788391564/Ejercicios-de-notaciones-en-arboles-Team

14. Traducción de una expresión en un arbol binario, fecha de acceso: noviembre 16, 2025, https://www.infor.uva.es/~cvaca/asigs/AlgInfPost.htm

15. Entendemos por árbol una estructura de datos formada por varios objetos - UPIICSA, fecha de acceso: noviembre 16, 2025, https://www.sites.upiicsa.ipn.mx/estudiantes/academia_de_informatica/estructura_y_rd/docs/u3/%C3%81rboles,%20explicaci%C3%B3n%20y%20ejemplos.pdf

16. Notación postfija, infija y prefija - YouTube, fecha de acceso: noviembre 16, 2025, https://www.youtube.com/watch?v=pQ2ZmaB1cTk

17. Shunting yard algorithm - Wikipedia, fecha de acceso: noviembre 16, 2025, https://en.wikipedia.org/wiki/Shunting_yard_algorithm

18. -INTRODUCCION ÁRBOLES:, fecha de acceso: noviembre 16, 2025, https://www.infor.uva.es/~belar/Ampliacion/Cursos%20Anteriores/Practicas_C/Curso%202003-04/Practicas%20ALUMNOS%202003-2004/Daniel_y_Eva/arboles%20binarios.doc

19. Árboles (trees). Continuamos con estructuras de datos, y… | by Lupo Montero | Medium, fecha de acceso: noviembre 16, 2025, https://medium.com/@lupomontero/%C3%A1rboles-trees-51783ba4ebe5

20. Árbol (informática) - Wikipedia, la enciclopedia libre, fecha de acceso: noviembre 16, 2025, https://es.wikipedia.org/wiki/%C3%81rbol_(inform%C3%A1tica)

21. Árbol AVL - Wikipedia, la enciclopedia libre, fecha de acceso: noviembre 16, 2025, https://es.wikipedia.org/wiki/%C3%81rbol_AVL

22. 6.2. Ejemplos de árboles — Solución de problemas con algoritmos y estructuras de datos, fecha de acceso: noviembre 16, 2025, https://runestone.academy/ns/books/published/pythoned/Trees/EjemplosDeArboles.html

23. The Shunting Yard Algorithm, fecha de acceso: noviembre 16, 2025, https://mathcenter.oxford.emory.edu/site/cs171/shuntingYardAlgorithm/

24. Árbol B+ Reformulado - Q2BStudio, fecha de acceso: noviembre 16, 2025, https://www.q2bstudio.com/nuestro-blog/17842/arbol-b-reformulado

25. Algoritmos PREORDEN, INORDEN y POSTORDEN en 3 minutos - YouTube, fecha de acceso: noviembre 16, 2025, https://www.youtube.com/watch?v=Jo2euX89Oz8

26. Árboles de decisión: creando modelos inteligentes capaces de explicar su respuesta, fecha de acceso: noviembre 16, 2025, https://tecscience.tec.mx/es/divulgacion-ciencia/arboles-de-decision-creando-modelos-inteligentes-capaces-de-explicar-su-respuesta/

27. Algoritmos y Estructura de Datos - Parte 7 de 7: Árboles y Gráficas - ResearchGate, fecha de acceso: noviembre 16, 2025, https://www.researchgate.net/profile/Reiner-Creutzburg/publication/277814853_Algoritmos_y_Estructura_de_Datos_Parte_7_Arboles_y_Graficas/links/558fdefd08ae1e1f9badf8e6/Algoritmos-y-Estructura-de-Datos-Parte-7-Arboles-y-Graficas.pdf?origin=scientificContributions

28. Tema 11: "Árbol balanceado AVL" - Sitio Web Docente del Prof ..., fecha de acceso: noviembre 16, 2025, https://docencia.eafranco.com/materiales/estructurasdedatos/11/Tema11.pdf

29. Estructura de datos - Árboles - Oscar Blancarte - Software Architecture, fecha de acceso: noviembre 16, 2025, https://www.oscarblancarteblog.com/2014/08/22/estructura-de-datos-arboles/

30. 4 ÁRBOLES. ÁRBOLES BINARIOS., fecha de acceso: noviembre 16, 2025, https://www6.uniovi.es/usr/cesar/Uned/EDA/Apuntes/TAD_apUM_04.pdf

31. Clase8-Arboles - ESTRUCTURAS DE DATOS, fecha de acceso: noviembre 16, 2025, https://www.uv.mx/personal/ermeneses/files/2021/08/Clase8-Arboles.pdf

32. LAS MATEMÁTICAS: UN LENGUAJE PARA DESCRIBIR LA NATURALEZA - Dialnet, fecha de acceso: noviembre 16, 2025, https://dialnet.unirioja.es/descarga/articulo/9149576.pdf

33. Qué son y cómo usar los Árboles - Luis Llamas, fecha de acceso: noviembre 16, 2025, https://www.luisllamas.es/que-es-un-arbol/

34. Dibujar árbol a partir de notación polaca inversa - YouTube, fecha de acceso: noviembre 16, 2025, https://www.youtube.com/watch?v=nulTOwli65c

35. Arboles de expresión a partir de expresión posfija - YouTube, fecha de acceso: noviembre 16, 2025, https://www.youtube.com/watch?v=wZ3vwpmWXl8

36. 6.4. Notación prefija, infija y posfija - Google Docs, fecha de acceso: noviembre 16, 2025, https://docs.google.com/document/d/1qnw6_DRv2ntsmrSPiOSDBaOCSndfELthgF8xK5UQ4-I/edit