Visualización y Evaluación de Expresiones Matemáticas con Árboles Binarios en Python.

Alumnos

- Benitez, Rodrigo Iván – rodrigoivanb18@gmail.com
- Caballero, Myriam Inés – myriamicaballero@gmail.com

Materia

- Programación I

Profesor/a

- Rigoni, Cinthia

Tutor

- Lara, Brian

Fecha de entrega

- 09 de junio de 2025

Índice

Introducción

En el marco de la asignatura **Programación I**, se propuso desarrollar un **trabajo integrador** que permitiera aplicar conceptos fundamentales de **estructuras de datos** utilizando el lenguaje **Python**. Tras revisar la **bibliografía provista** y explorar ejemplos prácticos, decidimos implementar una **calculadora de expresiones matemáticas** basada en **árboles binarios.**

La elección de este tema responde a su valor **didáctico** y **funcional**. Los **árboles binarios** no solo son una estructura esencial en informática, sino que también ofrecen una forma clara y jerárquica de representar **expresiones matemáticas**. Esta representación resulta especialmente útil para abordar un problema común en el ámbito

educativo: la comprensión de la **jerarquía de operaciones**. Muchos estudiantes presentan dificultades para aplicar correctamente las reglas de **prioridad entre operadores**, y los docentes enfrentan el desafío de enseñar estos conceptos de manera clara y efectiva.

A través de este proyecto, buscamos demostrar cómo una **estructura de árbol** puede facilitar la **descomposición**, **evaluación** y **visualización** de expresiones matemáticas paso a paso. El objetivo principal es integrar **teoría y práctica** mediante el desarrollo de una herramienta que no solo resuelva expresiones, sino que también sirva como **recurso pedagógico** para visualizar el proceso de cálculo de forma estructurada y comprensible.

Marco Teórico

Las **estructuras de datos** son componentes fundamentales en la programación, ya que permiten organizar y manipular la información de manera eficiente. Entre ellas, el **árbol binario** ocupa un lugar destacado por su capacidad para representar relaciones jerárquicas y operaciones matemáticas complejas.

Un **árbol binario** es una estructura compuesta por nodos, donde cada nodo puede tener como máximo dos hijos: uno izquierdo y uno derecho. Esta organización permite representar expresiones matemáticas de forma estructurada, facilitando tanto su análisis como su evaluación. En este contexto, los **nodos internos** del árbol representan **operadores** (como +, −, *, /), mientras que los **nodos hoja** contienen los **operandos** (números).

Para construir este árbol a partir de una expresión matemática, es necesario convertir la expresión desde su forma habitual (**notación infija**) a una forma más adecuada para el procesamiento computacional: la **notación postfija** o **notación polaca inversa**. En esta notación, los operadores se colocan después de sus operandos, eliminando la necesidad de paréntesis y respetando la jerarquía de operaciones. Por ejemplo, la expresión infija 3 + 4 * 2 se convierte en 3 4 2 * +.

La conversión se realiza mediante el **algoritmo de Shunting Yard**, desarrollado por Edsger Dijkstra. Este algoritmo utiliza una pila para reorganizar los elementos de la expresión, asegurando que los operadores se coloquen en el orden correcto según su **precedencia** y **asociatividad**.

Una vez obtenida la notación postfija, se construye el árbol binario de expresión. Cada vez que se encuentra un número, se crea un nodo hoja. Cuando se encuentra un operador, se crean nodos internos que enlazan con los operandos correspondientes. Este árbol se evalúa de forma **recursiva**, recorriendo primero los subárboles izquierdo y derecho, y luego aplicando la operación del nodo actual.

En Python, este proceso se implementa utilizando clases para representar los nodos del árbol y funciones para convertir la expresión, construir el árbol y evaluarlo. A continuación, se muestra un fragmento de código que define la estructura básica del nodo:

```
class Nodo:

    def __init__(self, valor):
```

```
        self.valor = valor
        self.izquierda = None
        self.derecha = None
```

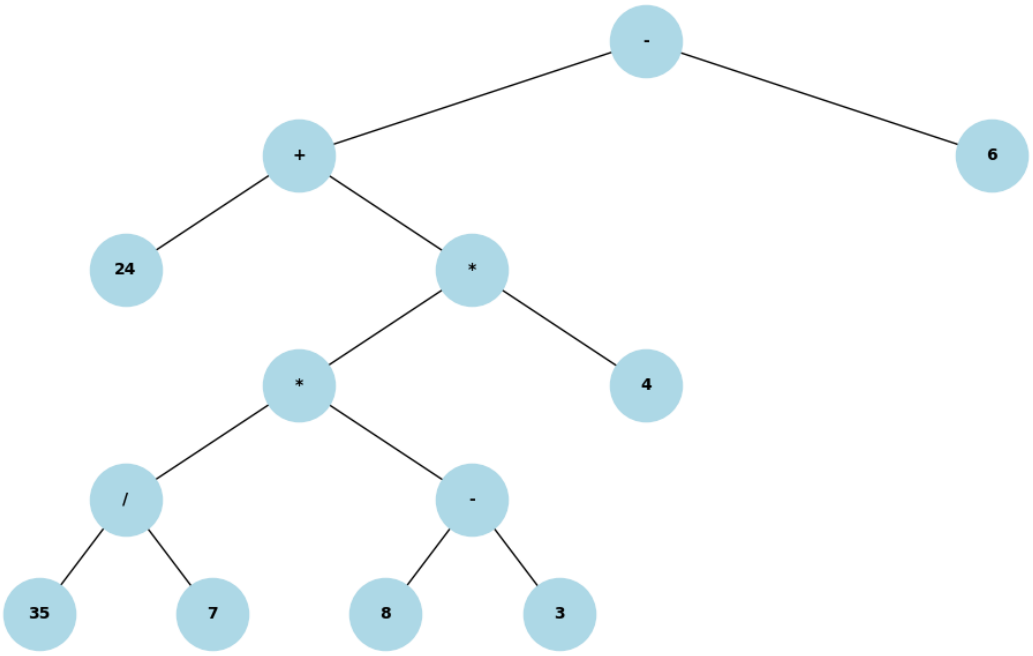
Este enfoque no solo permite resolver expresiones de forma precisa, sino que también ofrece una herramienta **visual y didáctica** para comprender cómo se aplican las reglas de prioridad entre operaciones. La visualización del árbol facilita la interpretación del orden en que se ejecutan las operaciones, lo cual es especialmente útil en contextos educativos.

La visualización del árbol facilita la interpretación del orden en que se ejecutan las operaciones, lo cual es especialmente útil en contextos educativos.
A continuación, se presenta un ejemplo visual del **árbol binario** correspondiente a la expresión:

24 + ((35 / 7) * (8 - 3)) * 4 - 6

Este gráfico permite observar la **jerarquía de operaciones** y cómo se estructura la expresión para su evaluación paso a paso:

1 - GRÁFICO DEL ÁRBOL BINARIO



Nota: En este árbol, los operadores de mayor precedencia (como / y -) aparecen más abajo porque se evalúan primero. La raíz del árbol representa la operación final. Esta estructura refleja el orden real de evaluación de la expresión, no la prioridad visual de los operadores.

Metodología Utilizada

Para el desarrollo de este proyecto se adoptó una **metodología práctica y progresiva**, basada en la integración de teoría, experimentación y validación.

En primer lugar, se realizó una **investigación teórica** sobre estructuras de datos, en particular sobre **árboles binarios** y su aplicación en la evaluación de expresiones matemáticas. Se consultaron fuentes como la documentación oficial de Python, libros especializados y materiales provistos por la cátedra. Esta etapa permitió comprender los fundamentos necesarios para abordar el problema desde una perspectiva estructurada.

A continuación, se exploró y adaptó el código base del proyecto “Implementación y recorrido de árboles binarios en Python”, con el objetivo de aplicarlo a una situación concreta: una **calculadora de expresiones matemáticas**. Se definieron los pasos clave del proceso:

1. **Tokenización** de la expresión infija ingresada por el usuario.
2. **Conversión a notación postfija** mediante el algoritmo de Shunting Yard.
3. **Construcción del árbol binario** a partir de la expresión postfija.
4. **Evaluación recursiva** del árbol para obtener el resultado.
5. **Visualización jerárquica** del árbol para facilitar la comprensión del orden de operaciones.

Durante el desarrollo, se realizaron múltiples **pruebas con expresiones variadas**, incluyendo números negativos, paréntesis anidados y combinaciones de operadores. Esto permitió validar la robustez del algoritmo y ajustar detalles en la lógica de conversión y evaluación.

En cuanto a las **herramientas utilizadas**, se trabajó con el lenguaje **Python**, empleando un entorno de desarrollo como **Visual Studio Code**. Para la visualización del árbol se utilizaron las bibliotecas **NetworkX** y **Matplotlib**, que permitieron representar gráficamente la estructura jerárquica de las expresiones.

El trabajo fue realizado de forma **colaborativa**, con una distribución equilibrada de tareas entre los integrantes del equipo. Se utilizó un repositorio en **GitHub** para el control de versiones y la documentación del código, lo que facilitó la organización y el seguimiento del avance del Proyecto.

Resultados Obtenidos

El resultado principal fue la implementación exitosa de una calculadora que permite ingresar expresiones matemáticas con operaciones básicas, paréntesis y números negativos, y que las resuelve utilizando un árbol binario de expresión.

La aplicación desarrollada permite:

- **Convertir expresiones en notación infija a notación postfija**, respetando la jerarquía de operadores.
- **Construir un árbol binario** que refleja la estructura lógica de la expresión.
- **Evaluar la expresión paso a paso**, utilizando un recorrido recursivo del árbol.

- **Visualizar el árbol de forma jerárquica**, facilitando la comprensión del orden de operaciones.

Durante las pruebas, se utilizaron expresiones variadas para verificar el correcto funcionamiento del sistema. Un ejemplo destacado fue:

```
24 + ((35 / 7) * (8 - 3)) * 4 - 6
```

El programa generó la notación postfija correspondiente, construyó el árbol binario completo, evaluó cada operación intermedia y devolvió el resultado final correcto: **118.0.**

Además, se generó un **gráfico visual del árbol**, que permitió observar claramente cómo se organizan las operaciones y cómo se resuelve la expresión desde las hojas hasta la raíz. Esta representación gráfica refuerza el valor didáctico del proyecto, ya que permite a estudiantes y docentes visualizar el proceso de evaluación de manera estructurada.

El código fue probado con distintos tipos de expresiones, incluyendo:

- Operaciones con paréntesis anidados.
- Números negativos y decimales.
- Combinaciones de operadores con diferentes niveles de precedencia.

En todos los casos, el sistema respondió de forma correcta, demostrando su **robustez**, **precisión** y **valor pedagógico.**

Conclusiones

Este proyecto permitió aplicar de manera concreta los conceptos teóricos sobre **estructuras de datos**, en particular los **árboles binarios**, demostrando que una estructura abstracta puede tener usos prácticos muy significativos. La implementación de una **calculadora basada en árboles** no solo resolvió correctamente expresiones matemáticas con diferentes niveles de complejidad, sino que también ofreció una forma **visual, clara y didáctica** de entender la jerarquía de operaciones.

Desde el punto de vista pedagógico, esta herramienta representa un recurso valioso para la enseñanza de la matemática, ya que permite **visualizar el "orden" detrás de una expresión algebraica**. Para estudiantes con dificultades en este tema, observar el árbol puede marcar una diferencia clave en la comprensión.

Además, el desarrollo del proyecto implicó el desafío de **integrar programación, estructuras de datos y lógica matemática**, lo que fortaleció habilidades técnicas y reafirmó la importancia de la práctica como camino para consolidar el aprendizaje.

Entre las posibles **mejoras futuras**, se podrían considerar:

- Ampliar el soporte para **funciones matemáticas** (como raíz cuadrada, logaritmos, etc.).
- Incorporar una **interfaz gráfica** para facilitar el uso por parte de estudiantes.
- Agregar una opción para **mostrar paso a paso la evaluación** de la expresión con animaciones o explicaciones interactivas.

En resumen, el trabajo no solo cumplió con los objetivos propuestos, sino que también abrió nuevas posibilidades para seguir explorando la relación entre programación y educación.

Anexos

```
import re

class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izquierda = None
        self.derecha = None

def es_operador(c):
    return str(c) in "+-*/^"

def precedencia(op):
    if op in "+-":
        return 1
    if op in "*/":
        return 2
    if op == "^":
        return 3
    return 0

def infija_a_postfija(expresion):
    salida = []
    pila = []
    tokens = re.findall(r'\d+\.\d+|\d+|[+\-*/^()]', expresion)

    for token in tokens:
        if token.replace('.', '', 1).isdigit() or (token.startswith('-') and
token[1:].replace('.', '', 1).isdigit()):
            salida.append(token)
        elif token == '(':
            pila.append(token)
        elif token == ')':
            while pila and pila[-1] != '(':
                salida.append(pila.pop())
            pila.pop()  # Sacar el '('
        else:
            while pila and es_operador(pila[-1]) and precedencia(token) <= precedencia(pila[-1]):
                salida.append(pila.pop())
            pila.append(token)
```

```
    while pila:
        salida.append(pila.pop())

    return salida

def construir_arbol(postfijo):
    pila = []
    for token in postfijo:
        if not es_operador(token):
            nodo = Nodo(float(token))
            pila.append(nodo)
        else:
            nodo = Nodo(token)
            nodo.derecha = pila.pop()
            nodo.izquierda = pila.pop()
            pila.append(nodo)
    return pila[0]

def evaluar_arbol(nodo):
    if not es_operador(nodo.valor):
        return nodo.valor

    izq = evaluar_arbol(nodo.izquierda)
    der = evaluar_arbol(nodo.derecha)

    if nodo.valor == '+':
        return izq + der
    elif nodo.valor == '-':
        return izq - der
    elif nodo.valor == '*':
        return izq * der
    elif nodo.valor == '/':
        return izq / der
    elif nodo.valor == '^':
        return izq ** der

def mostrar_arbol(nodo, nivel=0, prefijo="Raíz: "):
    if nodo is not None:
        print(" " * (nivel * 4) + prefijo + str(nodo.valor))
        if nodo.izquierda or nodo.derecha:
```

```
            if nodo.izquierda:
                mostrar_arbol(nodo.izquierda, nivel + 1, "L--- ")
            if nodo.derecha:
                mostrar_arbol(nodo.derecha, nivel + 1, "R--- ")


def main():
    expresion = input("Ingresa una expresión matemática: ")
    postfijo = infija_a_postfija(expresion)
    print("Expresión en notación postfija:", ' '.join(postfijo))

    arbol = construir_arbol(postfijo)
    print("\nÁrbol de expresión:")
    mostrar_arbol(arbol)

    resultado = evaluar_arbol(arbol)
    print("\nResultado:", resultado)

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