Construcción de Árboles de Expresión desde Notación Postfija

Introducción

Un árbol de expresión es una estructura de datos que representa expresiones matemáticas donde:

- Los nodos internos son operadores (+, -, *, /, ^)
- Las hojas son operandos (valores numéricos)

Esta estructura facilita la evaluación de expresiones matemáticas y permite representar claramente la precedencia de operadores.

Algoritmo de Construcción

La construcción de un árbol de expresión a partir de una notación postfija (también conocida como notación polaca inversa) se realiza utilizando una pila, siguiendo estos pasos:

1. Inicializar una pila vacía
2. Para cada token de la expresión postfija:
    - Si es un operando (número), crear un nodo hoja y apilarlo
    - Si es un operador:
        - Desapilar dos nodos (el primero será el hijo derecho, el segundo el izquierdo)
        - Crear un nuevo nodo con el operador y los hijos correspondientes
        - Apilar el nuevo nodo
3. Al finalizar, la pila debería contener un único nodo que es la raíz del árbol

Ejemplo Paso a Paso

Veamos cómo construir el árbol para la expresión `(3 + 4) * 5`, que en notación postfija es `3 4 + 5 *`.

Paso 1: Leer "3" (operando)

```
[Acción] Crear nodo con valor 3 y apilar
[Estado de la pila] → [3]
```

Paso 2: Leer "4" (operando)

```
[Acción] Crear nodo con valor 4 y apilar
[Estado de la pila] → [3, 4]
```

Paso 3: Leer "+" (operador)

```
[Acción]
- Desapilar 4 (será hijo derecho)
- Desapilar 3 (será hijo izquierdo)
- Crear nodo con operador "+" y apilar

[Estado de la pila] → [+]
                       / \
                      3   4
```

Paso 4: Leer "5" (operando)

```
[Acción] Crear nodo con valor 5 y apilar
[Estado de la pila] → [+, 5]
                       / \
                      3   4
```

Paso 5: Leer "*" (operador)

```
[Acción]
- Desapilar 5 (será hijo derecho)
- Desapilar el nodo "+" (será hijo izquierdo)
- Crear nodo con operador "*" y apilar

[Estado de la pila] → [*]
                      / \
                     +   5
                    / \
                   3   4
```

Estructura del Árbol Final

El árbol resultante representa correctamente la expresión `(3 + 4) * 5`:

```
    *
   / \
  +   5
 / \
3   4
```

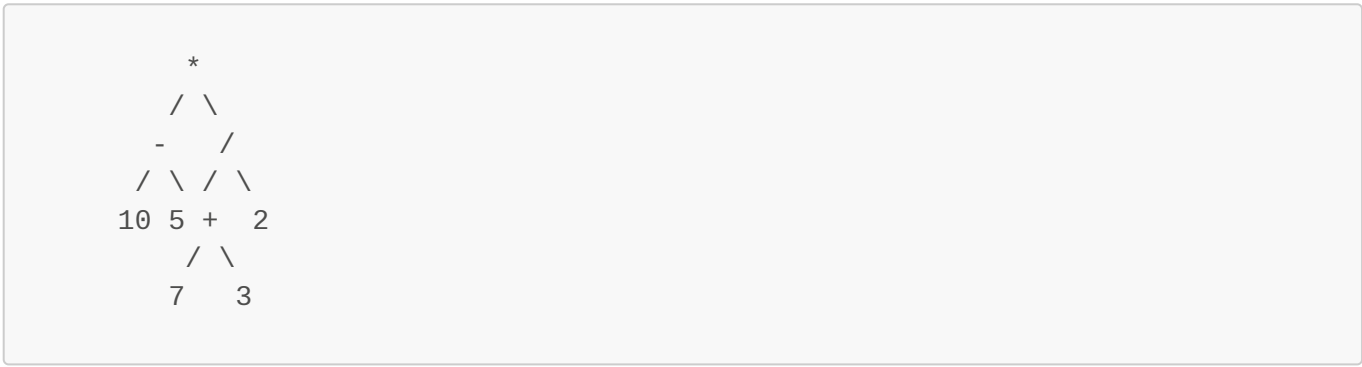
Ejemplo Más Complejo

Veamos el proceso de construcción para la expresión `(10 - 5) * ((7 + 3) / 2)`, cuya notación postfija es `10 5 - 7 3 + 2 / *`:

Proceso Completo

1. Leer "10" → Apilar nodo(10)
2. Leer "5" → Apilar nodo(5)
3. Leer "-" → Crear nodo(-) con hijos nodo(10) y nodo(5), apilar
4. Leer "7" → Apilar nodo(7)
5. Leer "3" → Apilar nodo(3)
6. Leer "+" → Crear nodo(+) con hijos nodo(7) y nodo(3), apilar
7. Leer "2" → Apilar nodo(2)
8. Leer "/" → Crear nodo(/) con hijos nodo(+) y nodo(2), apilar
9. Leer "*" → *Crear nodo(*)* con hijos nodo(-) y nodo(/), apilar

Árbol Final



Evaluación del Árbol

Una vez construido el árbol, podemos evaluarlo recursivamente:

1. Si el nodo es un operando, devolver su valor
2. Si el nodo es un operador:
   - Evaluar recursivamente el subárbol izquierdo
   - Evaluar recursivamente el subárbol derecho
   - Aplicar el operador a los resultados obtenidos

Recorridos del Árbol

Los diferentes recorridos nos permiten obtener distintas representaciones de la expresión:

1. **Inorden (izquierda, raíz, derecha)**:
   - Produce una representación similar a la notación infija: `((3 + 4) * 5)`
   - Útil para visualizar la expresión de forma tradicional

2. **Preorden (raíz, izquierda, derecha)**:
   - Muestra primero los operadores: `* + 3 4 5`
   - Útil para evaluar expresiones en lenguajes como LISP

3. **Posorden (izquierda, derecha, raíz)**:
   - Coincide con la notación postfija original: `3 4 + 5 *`

- Útil para verificar que el árbol se construyó correctamente