Nombre: Miguel Adrian Tubac Agustin

Carné: 202101927 **Curso:** MC2

Proyecto Final

Investigación

Algoritmos de búsqueda en arboles por ancho:

La búsqueda en árboles por anchura (BFS, Breadth-First Search) es un algoritmo de búsqueda utilizado en estructuras de datos de árboles o grafos. A continuación, te proporciono información sobre este algoritmo:

La búsqueda en árboles por anchura se centra en explorar todos los nodos de un nivel antes de pasar al siguiente nivel en un árbol o grafo. Comienza desde el nodo raíz y se expande gradualmente hacia los nodos vecinos antes de avanzar al siguiente nivel.

Proceso:

1. Se comienza desde el nodo raíz y se coloca en una cola.
2. Se extrae un nodo de la cola, se procesa y se agregan todos sus nodos hijos a la cola.
3. Se repite el proceso hasta que la cola esté vacía.

Características:

- Asegura que los nodos se procesen en orden de su nivel.
- Utiliza una estructura de datos cola para gestionar los nodos a visitar.
- Es completo y encuentra la solución más corta en términos de número de aristas para problemas de búsqueda.

Aplicaciones:

- Resolución de problemas de búsqueda en inteligencia artificial.
- Encontrar la distancia más corta entre dos nodos en un grafo no ponderado.
- Construcción de árboles de expansión mínima.

Ventajas:

- Simple y fácil de entender.
- Garantiza la búsqueda de la solución más corta en términos de número de aristas.

Desventajas:

- Puede requerir una cantidad significativa de memoria para almacenar nodos en la cola.
- No siempre es eficiente en términos de espacio, especialmente para grafos grandes.

Complejidad temporal:

- El tiempo de ejecución es proporcional al número total de nodos y aristas en el grafo o árbol.
- La búsqueda en árboles por anchura es un algoritmo fundamental con aplicaciones en diversos campos, desde ciencias de la computación hasta inteligencia artificial y redes. Su simplicidad y eficacia en la búsqueda de soluciones cortas lo convierten en una elección común en muchos contextos.

Algoritmos de búsqueda en arboles por profundidad:

La búsqueda en árboles por profundidad (DFS, Depth-First Search) es un algoritmo de búsqueda utilizado en estructuras de datos de árboles o grafos. A continuación, te proporciono información sobre este algoritmo:

La búsqueda en árboles por profundidad se centra en explorar lo más profundamente posible a lo largo de una rama antes de retroceder. Comienza desde el nodo raíz y se adentra en el árbol o grafo hasta llegar a la hoja antes de retroceder y explorar otra rama.

Proceso:

1. Se comienza desde el nodo raíz.
2. Se sigue explorando hacia abajo por una rama tanto como sea posible antes de retroceder.
3. Se repite el proceso hasta haber explorado todas las ramas.

Características:

- Puede implementarse de manera recursiva o mediante el uso de una pila para gestionar los nodos a visitar.
- No garantiza la búsqueda de la solución más corta en términos de número de aristas.
- Puede usarse para encontrar ciclos en grafos dirigidos.

Aplicaciones:

- Resolución de problemas de búsqueda en inteligencia artificial.
- Encontrar ciclos en grafos dirigidos.
- Topología de ordenamiento.

Ventajas:

- Puede ser más eficiente en términos de espacio en comparación con la búsqueda en anchura, ya que no requiere almacenar tantos nodos en memoria.
- Se presta bien a la implementación recursiva.

Desventajas:

- No garantiza la búsqueda de la solución más corta en términos de número de aristas.
- Puede caer en bucles infinitos si no se implementa adecuadamente en grafos con ciclos.

Complejidad temporal:

- El tiempo de ejecución es proporcional al número total de nodos y aristas en el grafo o árbol.
- La búsqueda en árboles por profundidad es versátil y se utiliza en diversos contextos. La elección entre búsqueda en anchura y búsqueda en profundidad depende de la naturaleza del problema y de los objetivos específicos de la aplicación.

Algoritmo para arboles de notación polaca:

La notación polaca, también conocida como notación prefija o notación de prefijo, es una forma de escribir expresiones matemáticas en la que el operador precede a sus operandos. Existen dos variantes principales de notación polaca: la notación polaca prefija y la notación polaca posfija (también conocida como notación de sufijo o notación inversa polaca).

En este caso, te proporcionaré información sobre un algoritmo para evaluar expresiones en árboles de notación polaca posfija, que es una variante comúnmente utilizada. La notación polaca posfija es fácil de evaluar utilizando una estructura de datos de pila.

Algoritmo para Evaluar Árboles de Notación Polaca Posfija:

- Inicia una pila vacía.
- Recorre la expresión desde el principio hasta el final.

Para cada elemento:

- Si es un operando (número), colócalo en la pila.
- Si es un operador, saca dos operandos de la pila, aplica el operador y coloca el resultado de nuevo en la pila.
- Al final del recorrido, el resultado final estará en la cima de la pila.

Ejemplo:

Considera la expresión en notación polaca posfija: 3 4 + 2 *.

Paso 1: Inicia una pila vacía.

Paso 2: Recorre la expresión.

Paso 3:

Encontramos 3, lo colocamos en la pila.

Encontramos 4, lo colocamos en la pila.

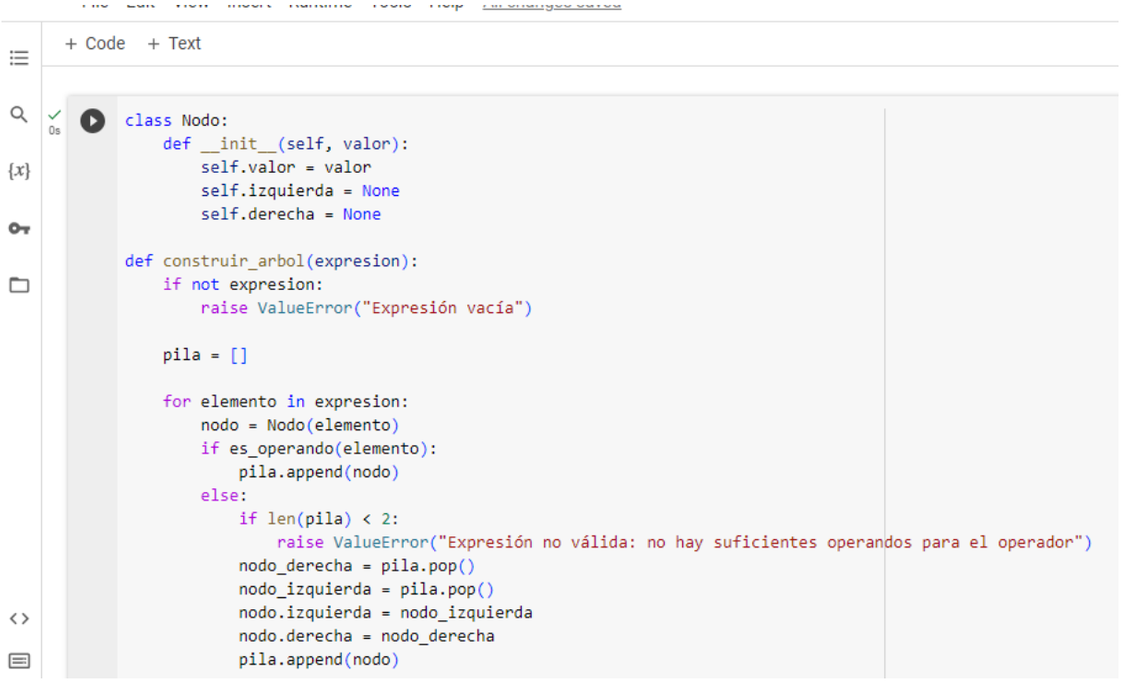
Encontramos +, sacamos 3 y 4 de la pila, sumamos y colocamos 7 en la pila.

Encontramos 2, lo colocamos en la pila.

Encontramos *, sacamos 7 y 2 de la pila, multiplicamos y colocamos 14 en la pila.

Paso 4: El resultado final (14) está en la cima de la pila.

1. Evaluación de la expresión: '3', '4', '+', '2', '*'



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

def construir_arbol(expresion):
    if not expresion:
        raise ValueError("Expresión vacía")

    pila = []

    for elemento in expresion:
        nodo = Nodo(elemento)
        if es_operando(elemento):
            pila.append(nodo)
        else:
            if len(pila) < 2:
                raise ValueError("Expresión no válida: no hay suficientes operandos para el operador")
            nodo_derecha = pila.pop()
            nodo_izquierda = pila.pop()
            nodo.izquierda = nodo_izquierda
            nodo.derecha = nodo_derecha
            pila.append(nodo)
```

```
    if len(pila) != 1:
        raise ValueError("Expresión no válida: quedan nodos en la pila al final")

    return pila[0]

def es_operando(elemento):
    return elemento.isdigit()

def recorrer_inorden(nodo):
    if nodo:
        recorrer_inorden(nodo.izquierda)
        print(nodo.valor, end=' ')
        recorrer_inorden(nodo.derecha)

# Ejemplo de uso
expresion_polaca = ['3', '4', '+', '2', '*']
arbol = construir_arbol(expresion_polaca)
recorrer_inorden(arbol)
```

```
3 + 4 * 2
```

2. Evaluación de la expresión: '4', '7', '*', '3', '+'

```
# Ejemplo de uso
expresion_polaca = ['4', '7', '*', '3', '+']
arbol = construir_arbol(expresion_polaca)
recorrer_inorden(arbol)
```

```
4 * 7 + 3
```

3. Evaluación de la expresión: '6', '9', '/', '3', '*'

```
# Ejemplo de uso
expresion_polaca = ['6', '9', '/', '3', '*' ]
arbol = construir_arbol(expresion_polaca)
recorrer_inorden(arbol)
```

```
6 / 9 * 3
```

Conclusiones

1. Versatilidad del Algoritmo:
   - La implementación del algoritmo permite construir un árbol a partir de una expresión en notación polaca posfija.
   - Puede manejar tanto operandos como operadores y construir la estructura del árbol de manera eficiente.

2. Manejo de Errores:
   - La implementación incluye verificaciones para casos de error, como una expresión vacía o la falta de operandos para un operador.
   - Estas verificaciones ayudan a garantizar la integridad de la expresión y la correcta construcción del árbol.

3. Utilización de Nodos:
   - La estructura de nodo utilizada permite representar tanto operandos como operadores.
   - Cada nodo tiene referencias a sus nodos hijos, lo que facilita la navegación y manipulación del árbol.

4. Evaluación Posterior:
   - El algoritmo se enfoca en la construcción del árbol y no en la evaluación del resultado.
   - La evaluación del resultado se puede realizar posteriormente utilizando un recorrido inorden o postorden del árbol.

5. Recorridos del Árbol:
   - Se proporcionan funciones de recorrido inorden y postorden para visualizar la expresión original o la notación polaca posfija equivalente.
   - Estos recorridos permiten entender la estructura del árbol y la relación con la expresión original.

6. Adaptabilidad y Personalización:
   - La función es_operando se puede personalizar según las reglas específicas para identificar operandos en tu aplicación.
   - Esto brinda flexibilidad para adaptar el algoritmo a diferentes contextos y tipos de expresiones.