De regex inicial a regex concatenada

La función insertar_concatenacion_explicita toma una expresión regular como entrada e inserta explícitamente el símbolo . para indicar concatenación en aquellos casos donde la concatenación es implícita. Este proceso es necesario porque, en las expresiones regulares, la concatenación entre símbolos o subexpresiones generalmente se asume implícitamente, pero a veces es necesario especificarlo de forma explícita para poder realizar ciertas operaciones, como la conversión de expresiones regulares a autómatas.
Un ejemplo es que (a|b)a se convierte en (a|b).a, el cambio puede parecer mínimo pero es de gran ayuda ya que si no se especifica la unión de ‘)’ con ‘a’ mediante ‘.’ pueden ocurrir errores al hacer la transformación

Complejidad:

- **Temporal**: O(n)O(n)O(n), donde nnn es la longitud de la expresión regular. El bucle principal recorre cada símbolo de la expresión una vez.
- **Espacial**: O(n)O(n)O(n), debido a la lista `salida` que almacena la nueva expresión.

De regex concatenada a notación postfija utilizando el algoritmo de Shunting Yard de Dijkstra

El propósito del algoritmo es tomar una expresión regular infija y transformarla en una forma postfija, para que sea más fácil de evaluar o procesar sin la necesidad de resolver la jerarquía de operadores con paréntesis.

Complejidad:

- **Temporal**: O(n)O(n)O(n), donde nnn es la longitud de la expresión regular. Cada carácter de la expresión es procesado una vez.
- **Espacial**: O(n)O(n)O(n), debido a la pila utilizada para almacenar operadores y la lista de salida.

De postfija a nfa

Una vez tenemos una expresión postfija como ab|, la función procesa en orden la expresión, y según lo va analizando asigna estados y transiciones, es decir que al ingresar ‘a’ crea un estado o y un estado 1 y los conecta mediante ‘a’ y lo mismo con b, al llegar a ‘|’ crea un nuevo estado inicial y un nuevo estado de aceptación y conecta el estado inicial a los estados iniciales de las transiciones mediante epsilon. Así generando un nfa, aunque este no sea en su mejor forma o más eficiente debido a que puede generar mas transiciones epsilon de las que se usan comúnmente funciona.

Complejidad:

- **Temporal**: O(n)O(n)O(n), donde nnn es la longitud de la expresión regular. Cada carácter de la expresión se procesa una vez.
- **Espacial**: O(S+E)O(S + E)O(S+E), donde SSS es el número de estados y EEE es el número de transiciones en el NFA. Esto es porque los estados y transiciones se almacenan en el NFA generado.

De nfa a dfa

Esta funcion se apoya de otras 2, e_closure y mover, calcula el conjunto de estados alcanzables mediante la función mover, calcula la cerradura epsilon de los estados alcanzables, si el conjunto destino no ha sido procesado ni está pendiente, se añade a estados_dfa, registra la transición en transiciones_dfa. Y esto se repite hasta que no quede nada en estados_dfa.

Complejidad:

1. **Cálculo de estados DFA**:
   - El número de estados del DFA es O(2S)O(2^S)O(2S), donde SSS es el número de estados del NFA (teóricamente, cada estado del DFA corresponde a un subconjunto de estados del NFA).
   - Cada estado del DFA se genera calculando la cerradura epsilon de un subconjunto de estados (O(E)O(E)O(E)).
2. **Transiciones del DFA**:
   - Hay O(2S×∣A∣)O(2^S \times |A|)O(2S×∣A∣) posibles transiciones en el DFA, donde ∣A∣|A|∣A∣ es el tamaño del alfabeto.
   - Para cada transición, se realiza una operación `mover` seguida de `cerradura_epsilon`.
3. **Temporal**: O(2S×∣A∣×(E+S))O(2^S \times |A| \times (E + S))O(2S×∣A∣×(E+S)), ya que por cada transición del DFA se realizan operaciones de `cerradura_epsilon` (O(E)O(E)O(E)) y `mover` (O(S)O(S)O(S)).
   Espacial: O(2S)O(2^S)O(2S), ya que almacenamos los estados del DFA como subconjuntos de estados del NFA.

O(2S+E)

Análisis de complejidad

Complejidad Temporal Global:

La complejidad global está dominada por la función `convertir_nfa_a_dfa`, ya que implica el mayor número de operaciones debido a la generación de subconjuntos. En el peor caso:

O(2S×∣A∣×(E+S))O(2^S \times |A| \times (E + S))O(2S×∣A∣×(E+S))

Donde:

- S es el número de estados en el NFA.
- ∣A∣ es el tamaño del alfabeto.
- E es el número de transiciones epsilon en el NFA.

Esto hace que el proceso sea exponencial en términos del número de estados del NFA, como es típico en la determinació de autómatas.

Complejidad Espacial Global:

La complejidad espacial también está dominada por `convertir_nfa_a_dfa`, ya que los estados del DFA son subconjuntos de estados del NFA. En el peor caso:

O(2S+E)O(2^S + E)O(2S+E)

Esto incluye el almacenamiento de estados DFA (O(2S)O(2^S)O(2S)) y las transiciones (O(E)O(E)O(E)).

Factores Relevantes:

1. **Tamaño de la Expresión Regular**:
    - Afecta principalmente las funciones de procesamiento previo como `convertir_a_posfijo` y `construir_nfa`.
2. **Cantidad de Estados y Transiciones en el NFA**:
    - Determina la complejidad de `cerradura_epsilon`, `mover`, y `convertir_nfa_a_dfa`.
3. **Tamaño del Alfabeto**:
    - Aumenta el número de transiciones que deben evaluarse durante la construcción del DFA.

Resumen:

- El algoritmo tiene un **costo exponencial** debido al crecimiento potencial de los estados del DFA (O(2S)O(2^S)O(2S)).
- Sin embargo, el código está diseñado para manejar casos prácticos en los que SSS y ∣A∣∣A∣∣A∣ son moderados.