Complejidad del Algoritmo CYK

Juan Gallardo; Juan Joya; Jefferson Gutiérrez

September 2024

1 Introducción

El algoritmo CYK (Cocke-Younger-Kasami) es un método utilizado para determinar si una cadena puede ser generada por una gramática libre de contexto en forma normal de Chomsky. En este informe, se detalla la complejidad del algoritmo a través de sus diferentes etapas, con el objetivo de demostrar que la complejidad total es $O(n^3)$.

2 GIC a FNC

La conversión de una gramática independiente del contexto (GIC) a la forma normal de Chomsky (FNC) es esencial para el análisis sintáctico, ya que simplifica la gramática mediante una estructura uniforme y elimina producciones vacías e indirectas. Esto mejora el análisis sintáctico al aumentar el determinismo y reducir los casos a considerar durante el mismo. Además, la FNC conserva el lenguaje generado por la GLC original, lo que facilita su manipulación y comprensión [1].

3 Descripción del Algoritmo

El algoritmo CYK opera sobre una cadena de entrada w de longitud n y una gramática definida. En este caso, fue evaluado en Python y en C++[2][3]. A continuación, se describen los pasos clave del algoritmo y su complejidad.

3.1 Inicialización de la Tabla

La primera etapa del algoritmo implica la inicialización de una tabla T de dimensiones $n \times n$, donde cada celda se inicializa como un conjunto vacío. Esto requiere $O(n^2)$ operaciones, ya que cada celda necesita ser creada e inicializada.

Complejidad: $O(n^2)$

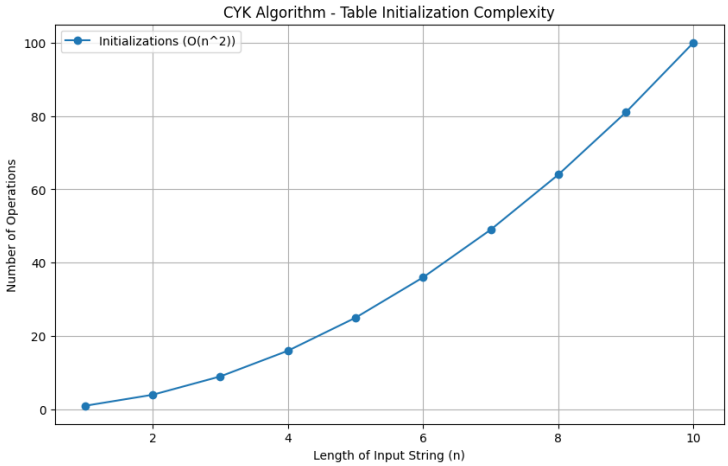

Figure 1: Inicialización de la tabla en el algoritmo CYK.

3.2 Llenado de la Diagonal

En esta etapa, se compara cada terminal de la cadena con los terminales de la gramática. Dado que hay n palabras en la cadena, esta operación se realiza n veces.

Complejidad: $O(n)$

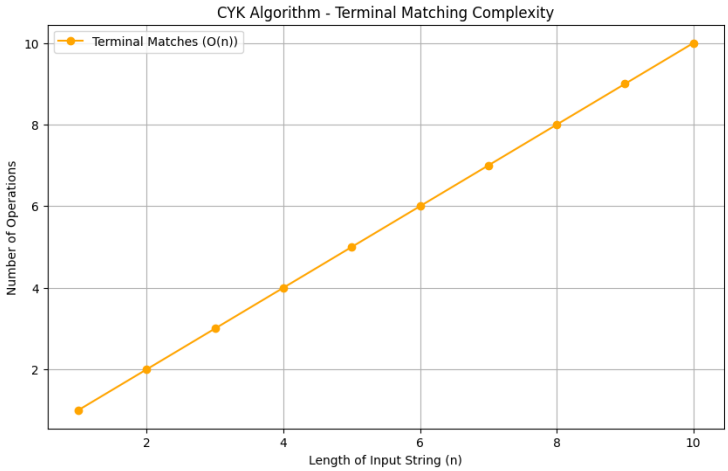

Figure 2: Llenado de Diagonal en el algoritmo CYK.

3.3 Llenado Completo de la Tabla

Este es el paso más crítico y costoso del algoritmo. Para llenar cada celda $T[i][j]$, se necesitan las siguientes operaciones:

1. Se debe iterar sobre todas las posibles divisiones k de la subcadena $w[i...j]$.
2. Para cada celda $T[i][j]$, hay que verificar todas las reglas no terminales que se pueden aplicar.

Dado que para cada celda $T[i][j]$ se realizan hasta n particiones, el número de celdas es $O(n)^2$ y cada celda requiere $O(n)$ comparaciones para las reglas. Por lo tanto, la complejidad total de este algoritmo es:

$$\text{Complejidad: } O(n^3)$$

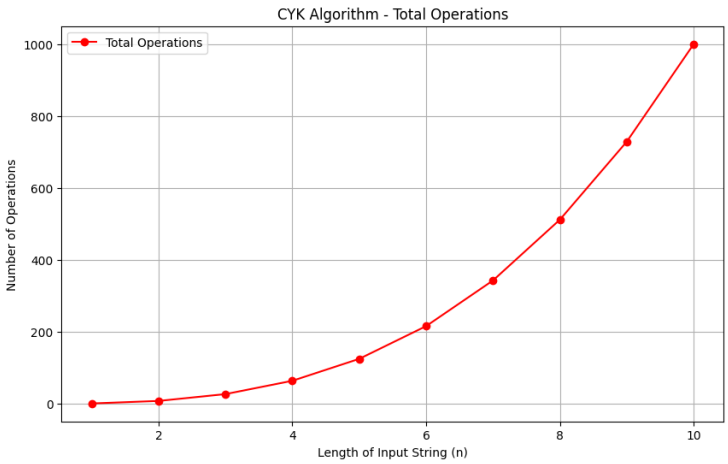

Figure 3: Complejidad final en el algoritmo CYK.

3.4 Complejidad Total

Al analizar la complejidad del algoritmo CYK, observamos que su comportamiento se puede modelar mediante una función cúbica:

$$f(x) = x^3$$

Podemos representar en una gráfica el comportamiento de ambas.

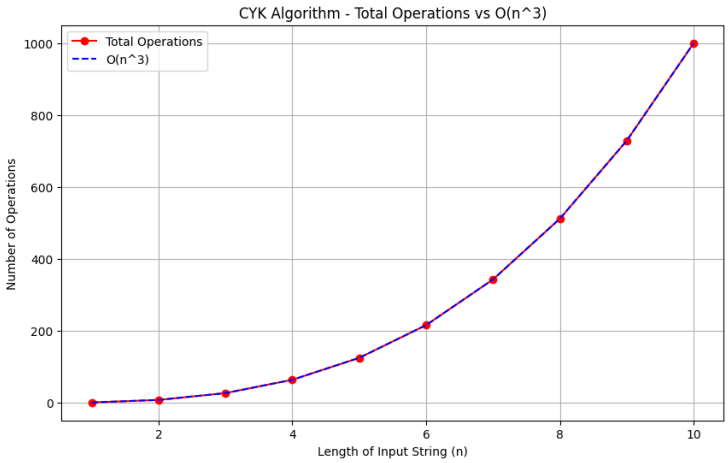

Figure 4: Complejidad final en el algoritmo CYK vs función cúbica.

Esto demuestra que el algoritmo tiene la misma complejidad que la función cúbica, siendo este $O(n^3)$

4 Conclusiones

El análisis de la complejidad del algoritmo CYK indica que, aunque las etapas iniciales de la inicialización de la tabla y el llenado de la diagonal pueden parecer independientes, estan asociadas con el computado de la tabla completa. El llenado del completo de la tabla es el factor dominante que eleva la complejidad total a $O(n^3)$. Este comportamiento cúbico se debe a la necesidad de evaluar múltiples particiones y aplicar las reglas gramaticales para cada combinación de subcadenas.

A medida que aumenta el tamaño de la cadena de entrada n, la cantidad de operaciones crece exponencialmente, lo que hace que el algoritmo sea poco práctico para entradas de gran longitud en situaciones en las que se requiere alta eficiencia. Sin embargo, la robustez del algoritmo en la verificación de cadenas generadas por gramáticas libres de contexto sigue siendo un aspecto valioso en aplicaciones donde el tamaño de la cadena no es extremadamente grande.

En resumen, el crecimiento cúbico de la complejidad impone una limitación considerable, pero la eficacia del CYK para problemas que requieren proce-

samiento gramatical lo mantiene como una herramienta fundamental en la teoría de lenguajes formales y análisis sintáctico.

5 Referencias

1. Why we convert CFG (context free grammer) into CNF (Chomsky normal form)? (s. f.). Quora. https://www.quora.com/Why-we-convert-CFG-context-free-grammer-into-CNF-Chomsky-normal-form[1]

2. GeeksforGeeks. (2023, 13 febrero). Cocke–Younger–Kasami (CYK) Algorithm. GeeksforGeeks. https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/[2]

3. Ahmadshafique. (s. f.). GitHub - ahmadshafique/CYK-Parser: This is a C++ implementation of CYK Algorithm. GitHub. https://github.com/ahmadshafique/CYK-Parser[3]