Proyecto No. 2 – Teoría de la computación

Estructura del programa

gramar_txt.py – representación de la gramática

Este módulo define la estructura de datos Grammar, la cual modela una CFG.

V: no terminales.

T: terminales.

S: símbolo inicial.

P: producciones.

Aquí también se hace la lectura de la gramática la cual se guarda en un archivo de texto.

cnf.py – conversión y simplificación a CNF

Implementación de los pasos para convertir una CFG en la forma normal de Chomsky CNF.

1. adición de un nuevo símbolo inicial.

2. eliminación de producciones épsilon.

3. eliminación de producciones unitarias.

4. terminalización y binarización de reglas largas.

También se hace una verificación para saber si la gramática ya está en CNF y así evitar todo el proceso.

cyk.py – implementación del algrotimo de CYK

Se aplica programación dinámica para verificar si una cadena pertenece al lenguaje generado por la gramática en CNF.

Construye una tabla triangular donde cada celda almacena los no terminales que generan un segmento.

Se utilizan dos estructuras de inidices para acelerar el acceso a las reglas de producción.

El algoritmo también guarda blackpoints que permite reconstruir el árbol sintáctico de derivación correspondiente.

Flujo de ejecución:

Se corre el programa principal con la gramática en un archivo txt.

Se valida y convierte la gramática a CNF si es necesario.

Se solicita al usuario una cadena para validar si pertenece a la gramática.

Se ejecuta el algoritmo de CYK, midiendo el tiempo de ejecución y generando, en caso de aceptación, el árbol de derivación correspondiente.

Discusión

Durante el desarrollo se presentaron varios retos tanto conceptuales como técnicos.

El primero fue la conversión correcta a CNF, ya que modificar las producciones mientras se iteraba causaba errores. Se solucionó trabajando con copias temporales y aplicando transformaciones por etapas.

Otro obstáculo fue la eliminación de producciones vacías y unitarias, que requirió comprender los cierres de símbolos anulables y unitarios para conservar el lenguaje original.

También resultó desafiante la reconstrucción del árbol sintáctico en el algoritmo CYK, pues el método base solo determina si una cadena es aceptada. Para resolverlo, se implementaron backpointers que almacenan las derivaciones y permiten reconstruir el árbol correspondiente.

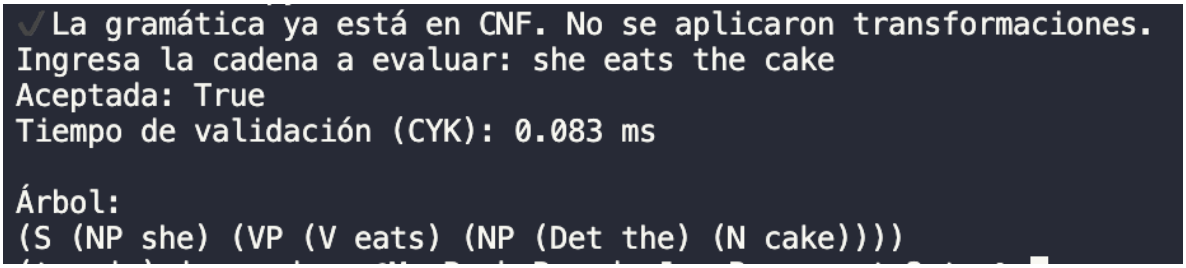
En cuanto a los aprendizajes, se comprobó que modularizar el código facilita el mantenimiento y las pruebas de cada componente. Además, se observó que el algoritmo CYK es eficiente para oraciones cortas y medianas.

Como recomendación final, se destaca la importancia de comprender bien los fundamentos teóricos antes de codificar, ya que esto simplifica la traducción del modelo formal al programa.

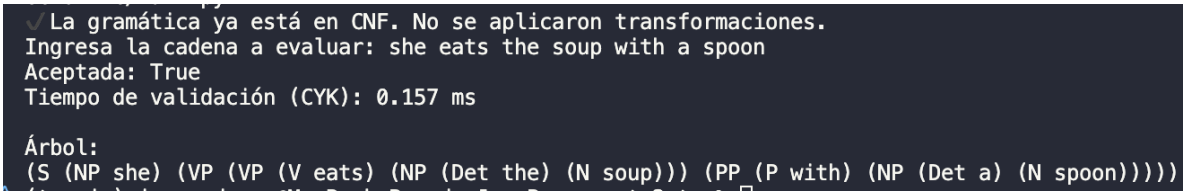
Ejemplos y pruebas

Cadenas aceptadas y tiempo de ejecución:

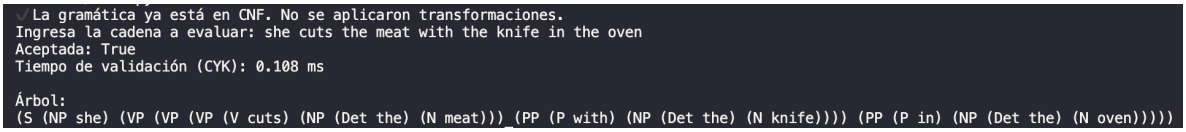
1. she eats the cake



2. he eats the soup with a spoon

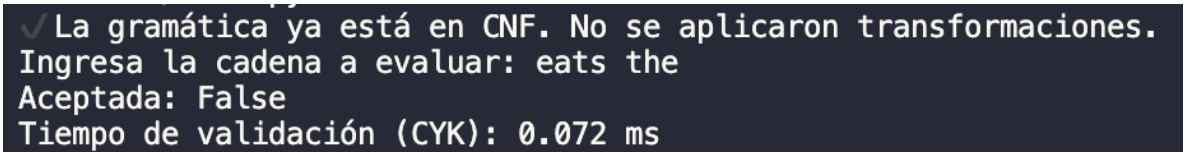


3. she cuts the meat with the knife in the oven

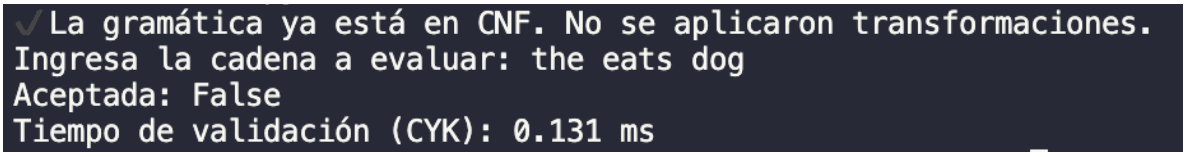


Cadenas no aceptadas

1. eats the



2. the eats dog



Cadenas que pueden provocar errores o comportamientos inesperados

1. Cadena vacía

```
✓ La gramática ya está en CNF. No se aplicaron transformaciones.
Ingresa la cadena a evaluar:
Aceptada: False
Tiempo de validación (CYK): 0.034 ms
```

2. he flys cake

```
✓ La gramática ya está en CNF. No se aplicaron transformaciones.
Ingresa la cadena a evaluar: he flys cake
Aceptada: False
Tiempo de validación (CYK): 0.110 ms
```

3. she eats the

```
✓ La gramática ya está en CNF. No se aplicaron transformaciones.
Ingresa la cadena a evaluar: she eats the
Aceptada: False
Tiempo de validación (CYK): 0.079 ms
```

Link del repositorio:

https://github.com/JoseOrdonezB/proyecto2-tc.git