GIA – Práctica de Programación y Algoritmia Avanzada

José Luis Balcázar, Jordi Delgado
Pedro Jesús Copado Méndez

Mayo 2023

Objetivo

El objetivo de esta actividad es poner en práctica los conocimientos de programación dinámica, para tal caso, se os propone implementar el algoritmo determinista *Cocke–Kasami–Younger* (CKY) o alternativamente conocido como CYK.

La actividad se debe realizar principalmente en grupo de 2 personas, pero se admitirá un reducido número de grupos de 3 personas, que habrán de solicitar dicha configuración explícitamente por email a pedro.jesus.copado@upc.edu antes del **9 \ 05 \ 2023**.

Descripción

El algoritmo CKY determinista es un algoritmo utilizado para analizar la estructura sintáctica de cadenas de texto, y determinar si pueden ser derivadas a partir de las reglas de una gramática. Para más detalles de como funciona el algoritmo CKY recomendamos consultar las notas de la asignatura de PLH o la siguiente bibliografía [1].

Se os pide desarrollar un programa en Python que implemente el algoritmo **determinista** CKY. El algoritmo tomará como entrada una gramática de tipo Context-Free Grammar (CFG) junto con una palabra del alfabeto, y deberá determinar si la palabra pertenece o no al lenguaje inducido por la gramática. El programa debe tener las siguientes características:

- **Entrada:** Lectura de una gramática CFG en Chomsky Normal Form (CNF) y una palabra. Tanto la gramática como la palabra pueden estar en formato libre.
- **Salida:** El programa devolverá un booleano que será True, si la palabra pertenece al lenguaje de la gramática, o False en caso contrario.

- **Programa principal:** Un programa principal desde el cual se pueda probar los algoritmos junto con los juegos de pruebas.
- **Comentarios:** Introducir comentarios en las partes relevantes del código.
- **Test:** Suministrar varios juegos de pruebas completos.

A continuación, se muestran algunas gramáticas en CNF de ejemplo que podéis utilizar para probar el algoritmo, pero su utilización es totalmente opcional:

1. G_1

S	→	a	|	XA	|	AX	| b
A	→	RB						
B	→	AX	|	b	|	a	
X	→	a						
R	→	XB						

2. G_2

S	→	AB	|	CD	|	CB	| SS
A	→	BC	|	a				
B	→	SC	|	b				
C	→	DD	|	b				
D	→	BA						

El código debe ser completamente funcional y ejecutable, es decir, no debe contener errores de programación y dar la salida correcta. Es importante explicar los códigos desarrollados.

Extensiones

Además de lo que os pedimos en el apartado anterior, opcionalmente, se pueden implementar las siguientes extensiones:

1. transformación de cualquier gramática CFG a CNF.
2. ampliar el algoritmo CKY a probabilístico.

Evaluación

Tan solo uno de los miembros del grupo deberá realizar la entrega en **el RACO** antes de la fecha límite **25 \ 05 \ 2023 hasta las 23:59**. No se aceptarán entregas posteriores a la fecha límite, y en ningún caso por email. La entrega será un único archivo .zip con el siguiente formato *nombre_completo1_nombre_completo2.zip*. Ejemplo:

$$Maria\_Garcia\_Juan\_Perez.zip$$

Cada entrega deberá contener almenos los siguientes ficheros:

- Un documento en pdf donde se describa brevemente:
  1. los nombres y apellidos de los integrantes del grupo
  2. enumeración de los ficheros que se entregan.
  3. la explicación de códigos desarrollados.
  4. una explicación del formato elegido para describir gramáticas
  5. descripción de las extensiones realizadas (en caso de haberse realizado).
  6. una valoración del aprendizaje obtenido.
  7. otras consideraciones que creais pertinentes.
- Ficheros con el código fuente.
- Ficheros con los test.

No se evaluarán entregas que no sigan este formato. Es importante que los programas desarrollados estén bien explicados. Notad que solo se evaluará las extensiones, si y solo si, se ha entregado la parte obligatoria. Se tendrá en cuenta para la evaluación, los juegos de pruebas suministrados con los algoritmos, donde se ilustre el correcto funcionamiento del mismo. En caso de detectar alguna copia, ya sea de Internet o de algún otro grupo, la puntuación será 0. En caso de que la copia sea de algún otro grupo, se calificará con un 0 tanto al grupo que copia como al que se ha dejado copiar. **Notad** que únicamente se permite el uso del módulo nltk.grammar siempre que no se utilizen las siguientes funcionalidades:

- remove_unitary_rules(grammar)
- binarize(grammar)
- chomsky_normal_form(new_token_padding,flexible)

References

[1] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.