Facultat d'Informàtica de Barcelona

GIA UPC
Programació i Algorísmia Avançada

Implementació de l'algorisme CKY

Alumnes :
Nadal Par, Marta
Pumares Benaiges, Irene

Tutors :
Balcázar, José Luis
Delgado, Jordi

May 29, 2024

Contents

1 Descripció de la tasca

Es demana desenvolupar un programa en Python que implementi l'algorisme determinista CKY per determinar si una cadena de text pot ser derivada a partir de les regles d'una gramàtica. L'algorisme prendrà com a entrada una gramàtica de tipus Context-Free Grammar (CFG) juntament amb una paraula de l'alfabet, i haurà de determinar si la paraula pertany o no al llenguatge induït per la gramàtica. El programa ha de tenir les característiques següents:

- Entrada: Lectura d'una gramàtica CFG en Chomsky Normal Form (CNF) i una paraula. Tant la gramàtica com la paraula poden estar en format lliure.
- Sortida: El programa tornarà un booleà que serà True, si la paraula pertany al llenguatge de la gramàtica, o False en cas contrari.
- Pragrama principal: Un programa principal des del qual es pugui provar els algorismes juntament amb els jocs de proves.
- Comentaris: Introduir comentaris a les parts rellevants del codi.
- Test: Subministrar diversos jocs de proves complets.

2 Implementació

En aquesta pràctica s'ha desenvolupat l'algorisme CKY. A més de l'algorisme CKY bàsic, s'ha optat per implementar també la seva versió probabilística i la transformació de gramàtiques CFG a forma normal de Chomsky (CNF).

Per a fer-ho, s'ha dividit el codi en diversos fitxers, cadascun amb una funció específica per tal de mantenir una estructura organitzada.

L'estructura de la implementació consta de sis fitxers, que són els següents:

- `cky.py`: classe CKY
- `cky_probabilistic.py`: classe ProbabilisticCKY
- `converter.py`: classe CNFConverter
- `grammar_generator.py`: classe GenerateGrammar
- `word_generator.py`: classe GenerateWord
- `main.py`: conté el fluxe principal del programa, des del qual es criden tots els mètodes necessaris per a l'execució d'aquest

2.1 Format de la gramàtica

Per a aquesta implementació, s'ha optat per formatejar la gramàtica de la següent manera: es tracta d'una llista de tuples, on cada tupla és una regla. Si la gramàtica és probabilística, cada regla és una tupla que conté una altra tupla (head, body) i la probabilitat. Si la gramàtica no és probabilística, cada regla és simplement una tupla que conté el head i el body.

$[((\prime ST\prime, [\prime A\prime, \prime B\prime]), 1.0),$	$ST \to AB \quad (0.5)$
$((\prime A\prime, [\prime C\prime, \prime D\prime]), 0.7),$	$A \to CD \mid a \quad (0.7 \mid 0.3)$
$((\prime A\prime, [\prime a\prime]), 0.3),$	
$((\prime B\prime, [\prime E\prime, \prime F\prime]), 0.6),$	$B \to EF \mid a \quad (0.6 \mid 0.4)$
$((\prime B\prime, [\prime b\prime]), 0.4),$	
$((\prime C\prime, [\prime c\prime]), 1.0),$	$C \to a \quad (1.0)$
$((\prime D\prime, [\prime d\prime]), 1.0),$	$D \to b \quad (1.0)$
$((\prime E\prime, [\prime e\prime]), 1.0),$	$E \to c \quad (1.0)$
$((\prime F\prime, [\prime f\prime]), 1.0)]$	$F \to d \quad (1.0)$

En aquest exemple, la gramàtica de l'esquerra segueix el format utilitzat per al programa, mentre que la de la dreta és una representació convencional.

Si la gramàtica utilitzada ja està en Forma Normal de Chomsky (FNC), la primera regla, que correspon al símbol inicial, ha d'especificar 'ST' com a símbol no terminal (head) de la regla. En cas contrari, no s'ha d'incloure 'ST' a la gramàtica, ja que el programa podria tenir problemes per a la transformació a FNC.

2.2 Algorisme CKY

El fitxer `cky.py` conté la implementació de l'algorisme CKY (Cocke-Kasami-Younger) per a verificar si una paraula pot ser generada per una gramàtica en Forma Normal de Chomsky (CNF).

Definició de la classe

El constructor `__init__` inicialitza la instància amb la gramàtica proporcionada en CNF. La gramàtica es passa com una llista de tuples, on cada tupla representa una regla de la forma (No Terminal, [Cos de la regla]).

Mètode parse

El mètode `parse` implementa l'algorisme CKY. Pren una paraula com a entrada i retorna un valor booleà indicant si la paraula pertany al llenguatge de la gramàtica o no.

Primer, s'inicialitza la taula CKY, que es completa en dues fases: primer, s'omplen les cel·les de la diagonal amb els símbols no terminals que produeixen els símbols terminals de la paraula. Després, es completa la resta de la taula seguint les regles de la gramàtica i afegint els símbols quan es troben coincidències. Finalment, si en la última posició es troba el símbol inicial de la gramàtica, retorna `True`, en cas contrari, retorna `False`.

2.3 Algorisme CKY probabilístic

El fitxer `cky_probabilistic.py` conté la implementació de la versió probabilística de l'algorisme CKY. Aquesta versió de l'algorisme no només verifica si una paraula pot ser generada per una gramàtica en CNF, sinó que també calcula la probabilitat que la paraula sigui generada per la gramàtica.

Definició de la classe

El constructor `__init__` inicialitza la instància de ProbabilisticCKY amb una gramàtica probabilística en CNF, de la forma ((No Terminal, [Cos de la regla]), probabilitat). A més, crida al mètode `compute_probabilities` per crear un diccionari `probabilities` que emmagatzema les probabilitats associades a cada regla per facilitar-ne l'accés durant l'execució de l'algorisme.

Mètode parse

El mètode `parse` implementa l'algorisme CKY probabilístic. Pren una paraula com a entrada i retorna la probabilitat que la paraula sigui generada per la gramàtica, o False en cas que no pugui ser generada. Això es realitza de la mateixa manera que en l'algorisme CKY, però per tal de calcular les probabilitats, es multiplica la probabilitat de la regla aplicada per les probabilitats de les subcademes corresponents en la taula.

2.4 Conversió d'una gramàtica CFG a CNF

El fitxer `converter.py` conté la implementació d'una classe anomenada CNFConverter que transforma una gramàtica lliure de context (CFG) a la seva forma normal de Chomsky (CNF). Aquesta classe només permet treballar amb gramàtiques no probabilístiques.

Definició de la classe
El constructor `__init__` inicialitza la classe amb la gramàtica proporcionada, un booleà que indica si és probabilística o no i un booleà que indica si el símbol inicial de la gramàtica pot generar la paraula buida. Crea una còpia de la gramàtica CFG proporcionada, que s'utilitzarà per convertir-la a la forma normal de Chomsky (CNF).

Mètode is_cnf

El mètode `is_cnf` comprova si la gramàtica està en forma normal de Chomsky. Revisa cada regla de la gramàtica i verifica que estigui en alguna de les següents formes:

- $A \to BC$, on A, B, C són símbols no terminals.
- $A \to a$, on A és un símbol no terminal i a és un símbol terminal.
- $ST \to \epsilon$, només a la regla inicial.

Mètode converter

Aquest mètode s'encarrega de convertir la gramàtica a CNF. Per fer-ho, s'han de seguir uns passos en ordre, i aquesta funció s'encarrega de cridar als mètodes que representen cada pas en l'ordre pertinent.

Mètode create_start_symbol

Té com a objectiu modificar la gramàtica per garantir que tingui una regla inicial única, representada per 'ST' (Start Symbol). D'aquesta manera ens assegurem que la gramàtica estigui en format correcte per l'anàlisi posterior.

Mètode remove_epsilon_productions

Aquest mètode elimina les produccions epsilon (produccions que generen la cadena buida). Ho fa de manera iterativa, identificant i eliminant aquestes produccions i ajustant la gramàtica en conseqüència. Si la gramàtica original pot generar la cadena buida, en el mètode 'converter' es fa l'addició d'una regla que genera la cadena buida a partir del símbol inicial ('ST') per mantenir la capacitat de generar la cadena buida, tot preservant la forma CNF. Això assegura que la gramàtica resultant sigui equivalent a la original.

Mètode remove_unit_productions

Després d'eliminar les cadenes buides, aquest mètode s'encarrega d'eliminar les regles que generen un únic símbol no terminal. Si existeixen regles de la forma $A \to B$ aquestes s'han d'eliminar. Això implica que, si tenim la regla $B \to b$, aquesta es transformarà en la

regla $A \to b$. Per tant, totes les regles que continguin B en el seu cos, seràn substituides per b.

Mètode introduce_aux_symbols

El tercer pas consisteix en evitar tenir regles que produeixen símbols mixtes (terminals i no terminals) o que produeixen més d'un símbol terminal. Per fer-hi front, s'afegeixen símbols no terminals addicionals. Per exemple, si una regla és de la forma $A \to B, a$; s'ha d'introduir un nou símbol no terminal N i reescriure la regla anterior com $A \to B, N$. A més, s'afegeix la regla $N \to a$.

Mètode replace_long_productions

Com s'ha comentat, no poden haver regles que generin més de dos símbols no terminals, per tant, per a construir el cinquè i últim pas de la conversió, es defineix aquest mètode que s'encarrega de descomposar aquestes regles. Per fer-ho, si existeix la regla $A \to B, C, D$, s'ha de descomposar en $A \to B, X$ i s'ha d'afegir la regla $X \to C, D$.

2.5 Generació d'una gramàtica aleatòria

El fitxer `grammar_generator.py` conté la implementació d'una classe que genera una gramàtica aleatòria que pot estar en forma normal de Chomsky o no, i pot ser probabilística o no (si és probilística ja estarà en CNF). La generació de gramàtiques aleatòries és útil per provar que la implementació funciona correctament amb qualsevol gramàtica.

Definició de la classe

El constructor `__init__` inicialitza un diccionari de gramàtica buit, que contindrà les regles de la gramàtica generada.

Primer de tot, es defineixen diversos mètodes per generar diferents tipus de símbols, després es defineixen dos mètodes per gener les regles, ja siguin en forma CNF o no i finalment un mètode que s'encarrega de cridar als anteriors per a generar la gramàtica aleatòria.

Mètode generate_epsilon

Aquest mètode retorna una cadena buida, que representa la cadena epsilon. Serà utilitzat per generar regles per gramàtica que no estàn en CNF.

Mètode generate_nonterminal

Aquest mètode genera i retorna un símbol no terminal aleatori, de la forma "N" seguit d'un número aleatori entre 1 i 10.

Mètode generate_terminal

Aquest mètode genera i retorna un símbol terminal aleatori, que és una lletra minúscula entre la 'a' i la 'z'.

Mètode generate_cnf_rules

Aquest mètode es fa servir per generar regles en forma normal de Chomsky (CNF). Com s'ha comentat, perquè una gramàtica estigui en CNF, ha de tenir totes les regles de la froma $NoTerminal \rightarrow Terminal$ o $NoTerminal \rightarrow NoTerminal, NoTerminal$.

Primer, es genera el 'head' (no terminal) cridant al mètode `generate_nonterminal`. Després es genera el 'body', que es decideix aleatòriament (amb un 50% de probabilitat) si serà un símbol terminal o dos símbols no terminals, cridant als mètodes que els generen. En el cas de generar dos símbols no terminals, s'assegura que no siguin iguals al 'head' ni a qualsevol generador del head o generador de generadors, d'aquesta manera s'evita la creació de bucles.

Mètode generate_non_cnf_rules

Aquest mètode genera regles que no estan en CNF. Les regles poden contenir símbols terminals, no teminals o cadena buida. Per evitar regles massa llargues, la mida del 'body' es limita a un màxim 5. A més, aquest mètode s'assegura que no hi hagi més d'una producció epsilon en la mateixa regla, que el 'head' de la regla no aparegui en el 'body' de la regla i que no hi hagi regles que només generen la cadena buida. Fent aquestes comprovacions, s'assegura que no es generin cicles i s'eviten problemes al transformar a CNF.

Mètode find_generators

Donat un símbol no terminal, aquest mètode, troba tots els no terminals que el generen, això es fa de manera recursiva, comporvant totes les regles que generen el símbol que genera el no terminal donat i així successivament. Això permet evitar la creació de cicles en les regles.

Mètode generate_random_grammar

Finalment, es genera una gramàtica aleatòria amb un nombre de regles de partida que varia entre 5 i 8, encara que després pot ser que se n'afegeixin més. Pot generar una gramàtica CNF o no, i pot ser probabilística o no, segons els paràmetres que se li passen.

El mètode garanteix que les gramàtiques generades siguin coherents. S'assegura que la primera regla generada no sigui un terminal, que no es puguin generar regles amb el mateix símbol no terminal al 'head' i al 'body' i que tots els símbols no terminals necessaris estiguin presents en el 'head' d'alguna regla. A més, en el cas de generar una gramàtica probabilística, es garanteix que la suma de les probabilitats de les regles que tenen el mateix 'head' sigui 1.

2.6 Generació d'una paraula aleatòria

El fitxer `word_generator.py` conté la implementació d'una classe que genera paraules a partir d'una gramàtica, sigui aquesta gramàtica probabilística o no. A més, pot generar paraules vàlides (acceptades per la gramàtica) o invàlides (no acceptades per la gramàtica). Per generar paraules que siguin útils per fer els experiments i comprovar la implementació, no es poden generar paraules buides.

Definició de la classe

El constructor inicialitza la instància amb la gramàtica proporcionada i un booleà que indica si la gramàtica és probabilística o no.

Mètode generate_word

Aquest mètode genera una paraula a partir de la gramàtica donada. Depenent del valor del paràmetre `valid`, es decideix si es genera una paraula vàlida o invàlida.

Si `valid` és `True`, es crida al mètode `generate_valid_word` per generar una paraula vàlida, passant-li el símbol inicial (ST) de la gramàtica com a paràmetre. Si `valid` és `False`, es crida al mètode `generate_invalid_word`, també amb el símbol inicial (ST), per generar una paraula invàlida.

Mètode generate_valid_word

Aquest és el mètode que genera paraules vàlides dins del llenguatge definit per la gramàtica. A més, maneja tant gramàtiques probabilístiques com no probabilístiques, adaptant el procés de selecció de regles.

Donat el símbol (`start_symbol`), es selecciona totes aquelles regles qe tinguin el símbol com a 'head', i aleshores es tria aleatòriament entre una d'aquestes regles. Si la gramàtica és probabilística, aquesta selecció es fa tenint en compte les probabilitats associades a cada regla. Per cada símbol del cos de la regla triada , si és terminal, s'afegeix a la paraula, sinó, es busca la següent regla recursivament, cridant-se a si mateix.

Mètode generate_invalid_word

Aquest mètode, per tal de generar una paraula no acceptada per la gramàtica, genera una paraula vàlida cridant al mètode anterior i es modifica un caràcter aleatori de la paraula creada.

2.7 main

El fitxer `main.py`, serveix per generar, convertir i analitzar gramàtiques. Permet experimentar amb gramàtiques en CNF o no, gramàtiques probabilístiques o no, i verificar la pertinença de paraules al llenguatge de la gramàtica. Això es realitza utilitzant les classes definides anteriorment.

Per provar les nostres implementacions de diferents maneres, el programa ofereix tres opcions per generar i utilitzar les gramàtiques, permetent a l'usuari escollir l'execució que ell prefereixi d'entre les següents:

Opció 1: Executar Gramàtica i Paraula per Defecte:

Si l'usuari escull l'opció 1, el programa executa una gramàtica i una paraula predefinides, mostra la gramàtica original, la converteix a CNF si és necessari, genera una paraula, i verifica si la paraula pertany a la gramàtica utilitzant l'algoritme CKY. Si es vol modificar la gramàtica per defecte, s'ha d'editar el codi d'aquest fitxer.

Opció 2: Generació Aleatòria:

Si l'usuari escull l'opció 2, el programa permet a l'usuari especificar si la gramàtica ha de ser probabilística, en cas d'escollir una gramàtica probabilística, aquesta ja estarà en CNF per defecte, en altre cas, es permet escollir si es vol una gramàtica en CNF o no, i, finalment, si la paraula generada ha de pertànyer a la gramàtica. A continuació, el programa genera una gramàtica i una paraula aleatòriament segons aquests paràmetres i verifica si la paraula pertany a la gramàtica.

Opció 3: Joc de Proves:

Si l'usuari escull l'opció 3, el programa executa un conjunt d'experiments predefinits (cridant a la funció `executar_experiment`). Són 6 experiments que combinen tots els paràmetres possibles: si la gramàtica està en CNF o no, si és probabilística o no i si la paraula pertany a la gramàtica o no (excloent el cas de probabilística i no en CNF). Els resultats d'aquests experiments es guarden en un fitxer de text anomenat `resultats_joc_de_proves.txt`.

3 Resultats dels joc de prova

Per aquesta secció, es farà ús de la tercera opció que ofereix el programa. Com bé s'ha explicat anteriorment, s'ha executat un experiment amb cada una de les combinacions de paràmetres disponibles i s'ha fixat una llavor per assegurar la reproductibilitat dels resultats (seed = 9876543).

A continuació, s'analitzen els resultats extrets del fitxer de text generat. Per cada experiment, es proporcionen els diferents paràmetres utilitzat (CNF, probabilística, paraula vàlida), la gramàtica, la paraula i el resultat obtingut. A més, s'ha realitzat l'algorisme a mà per comprovar si la implemetació funciona correctament, tant pel cas probabilístic com per no probabilístic. En el cas de l'experiment número 5, que implica la conversió de la gramàtica de CFG a CNF, hem verificat que la paraula pugui ser generada tant amb la gramàtica original com amb la gramàtica en CNF.

Experiment 1

Paràmetres:
- Gramàtica en CNF ✓
- Gramàtica probabilística ×
- Paraula vàlida ✓

Gramàtica original:

$$
\begin{array}{lcl}
ST \to N2\,N5 & N5 \to N10\,N1 & N9 \to N4, N1 \\
ST \to N9\,N6 & N5 \to N4\,N10 & N9 \to b \\
ST \to N1\,N2 & N5 \to N7\,N8 & N1 \to a \\
ST \to l & N6 \to n & N1 \to u \\
ST \to f & N6 \to q & N1 \to m \\
N2 \to N9\,N8 & N6 \to v & N4 \to N7, N6 \\
N2 \to N5\,N1 & N9 \to c & N8 \to r \\
N2 \to w & N9 \to u & N10 \to c \\
N5 \to b & N9 \to l & N7 \to l
\end{array}
$$

Paraula: ulr

Resultat : `True`

Table 1: Algoritme CKY a mà de l'experiment 1

u	l	r
N9, N1	×	**ST**
	ST, N9, N7	N2, N5
		N8

Experiment 2

Paràmetres:
- Gramàtica en CNF ✓
- Gramàtica probabilística ×
- Paraula vàlida ×

Gramàtica original:

$$
\begin{array}{lcllcl}
ST & \to & \varepsilon & N2 & \to & N10\,N6 \\
ST & \to & y & N1 & \to & N5\,N6 \\
ST & \to & N8\,N5 & N1 & \to & N3\,N2 \\
ST & \to & N4\,N10 & N3 & \to & N4\,N7 \\
N5 & \to & k & N6 & \to & N3\,N4 \\
N5 & \to & N2\,N3 & N10 & \to & x \\
N5 & \to & N10\,N10 & N4 & \to & c \\
N8 & \to & N1\,N2 & N7 & \to & s \\
N2 & \to & g & & &
\end{array}
$$

Paraula: *csxcscnxx*

Resultat : `False`

Table 2: Algoritme CKY a mà de l'experiment 2

c	s	x	c	s	c	n	x	x
N4	N3	×	×	×	N2	×	×	×
	N7	×	×	×	×	×	×	×
		N10	×	×	N2	×	×	×
			N4	N3	N6	×	×	×
				N7	×	×	×	×
					N4	×	×	×
						×	×	×
							N10	N5
								N10

Experiment 3

Paràmetres:
- Gramàtica en CNF ✓
- Gramàtica probabilística ✓
- Paraula vàlida ✓

Gramàtica original:

ST	$\rightarrow$	$N2\,N4$ (0.2)	$N4$	$\rightarrow$	o (0.33)	$N9$	$\rightarrow$	$N6\,N7$ (0.5)
ST	$\rightarrow$	o (0.27)	$N4$	$\rightarrow$	a (0.1)	$N5$	$\rightarrow$	$N6\,N1$ (0.27)
ST	$\rightarrow$	l (0.06)	$N4$	$\rightarrow$	e (0.19)	$N5$	$\rightarrow$	e (0.4)
ST	$\rightarrow$	$N7\,N4$ (0.26)	$N4$	$\rightarrow$	g (0.14)	$N5$	$\rightarrow$	$N9\,N7$ (0.28)
ST	$\rightarrow$	$N9\,N10$ (0.2)	$N10$	$\rightarrow$	a (0.06)	$N5$	$\rightarrow$	y (0.04)
$N2$	$\rightarrow$	w (0.2)	$N10$	$\rightarrow$	$N9\,N5$ (0.64)	$N3$	$\rightarrow$	$N5\,N8$ (1.0)
$N2$	$\rightarrow$	$N10\,N5$ (0.47)	$N10$	$\rightarrow$	$N6\,N5$ (0.25)	$N7$	$\rightarrow$	l (1.0)
$N2$	$\rightarrow$	z (0.07)	$N10$	$\rightarrow$	i (0.05)	$N6$	$\rightarrow$	e (1.0)
$N2$	$\rightarrow$	$N3\,N8$ (0.26)	$N9$	$\rightarrow$	f (0.32)	$N8$	$\rightarrow$	w (1.0)
$N4$	$\rightarrow$	$N2\,N5$ (0.23)	$N9$	$\rightarrow$	e (0.18)	$N1$	$\rightarrow$	w (1.0)

Paraula: *eley*

Resultat : `0.001`

Table 3: Algoritme CKY probabilístic a mà de l'experiment 3

e	l	e	y
N4: 0.19 N9: 0.18 N5: 0.4 N6: 1.0	N5: 0.0504 N9: 0.5	N10: 0.128	**ST: 0.001** N2: 0.0024
	ST: 0.06 N7: 1.0	ST: 0.0494	×
		N4: 0.19 N9: 0.18 N5: 0.4 N6: 1.0	N10: 0.01
			N5: 0.04

Experiment 4

Paràmetres:
- Gramàtica en CNF ✓
- Gramàtica probabilística ✓
- Paraula vàlida ×

Gramàtica original:

$$
\begin{array}{lcllcl}
ST & \to & \varepsilon\ (0.2) & N6 & \to & j\ (1.0) \\
ST & \to & l\ (0.2) & N7 & \to & N3\,N3\ (1.0) \\
ST & \to & N8\,N4\ (0.37) & N8 & \to & N1\,N6\ (1.0) \\
ST & \to & N3\,N6\ (0.13) & N3 & \to & h\ (1.0) \\
ST & \to & i\ (1.0) & N1 & \to & t\ (1.0) \\
N4 & \to & N7\,N8\ (1.0) & & &
\end{array}
$$

Paraula: $tjhhtv$

Resultat : `False`

Table 4: Algoritme CKY probabilístic a mà de l'experiment 4

t	j	h	h	t	v
N1: 1.0	N8: 1.0	×	×	×	×
	N6: 1.0	×	×	×	×
		N3: 1.0	N7: 1.0	×	×
			N3: 1.0	×	×
				N1: 1.0	×
					×

Experiment 5

Paràmetres:
- Gramàtica en CNF ×
- Gramàtica probabilística ×
- Paraula vàlida ✓

Gramàtica original:

$$
\begin{array}{lcl}
N3 & \to & N6\,N9 \\
N3 & \to & N9\,y\,a\,l \\
N3 & \to & r \\
N3 & \to & N4\,N5\,h \\
N3 & \to & z\,e \\
N9 & \to & N5\,N8\,N4 \\
N9 & \to & N1\,N10\,u\,j\,a \\
N9 & \to & l
\end{array}
\qquad
\begin{array}{lcl}
N5 & \to & v \\
N5 & \to & g \\
N5 & \to & i\,y\,a \\
N4 & \to & N8\,y\,g \\
N1 & \to & N4\,N4\,N8\,d\,w \\
N8 & \to & e \\
N6 & \to & \varepsilon \\
N10 & \to & \varepsilon
\end{array}
$$

Gramàtica transformada:

$$
\begin{array}{lcl}
ST & \to & \varepsilon \\
ST & \to & N5\,X \\
ST & \to & r \\
ST & \to & N4\,X1 \\
ST & \to & Z\,E \\
ST & \to & N5\,X5 \\
ST & \to & N1\,X6 \\
ST & \to & l \\
N5 & \to & v \\
N5 & \to & g
\end{array}
\qquad
\begin{array}{lcl}
N5 & \to & I\,X2 \\
N4 & \to & N8\,X3 \\
N1 & \to & N4\,X4 \\
N8 & \to & e \\
Y & \to & y \\
A & \to & a \\
L & \to & l \\
H & \to & h \\
Z & \to & z
\end{array}
\qquad
\begin{array}{lcl}
E & \to & e \\
I & \to & i \\
G & \to & g \\
D & \to & d \\
W & \to & w \\
U & \to & u \\
J & \to & j \\
X & \to & Y\,X7 \\
X1 & \to & N5\,H
\end{array}
\qquad
\begin{array}{lcl}
X2 & \to & Y\,A \\
X3 & \to & Y\,G \\
X4 & \to & N4\,X8 \\
X5 & \to & N8\,N4 \\
X6 & \to & U\,X9 \\
X7 & \to & A\,L \\
X8 & \to & N8\,X10 \\
X9 & \to & J\,A \\
X10 & \to & D\,W
\end{array}
$$

Paraula: *iyayal*

Resultat : `True`

Table 5: Algoritme CKY a mà de l'experiment 5

i	y	a	y	a	l
I	×	N5	×	×	**ST**
	Y	X2	×	×	×
		A	×	×	×
			Y	X2	X
				A	X7
					ST, L

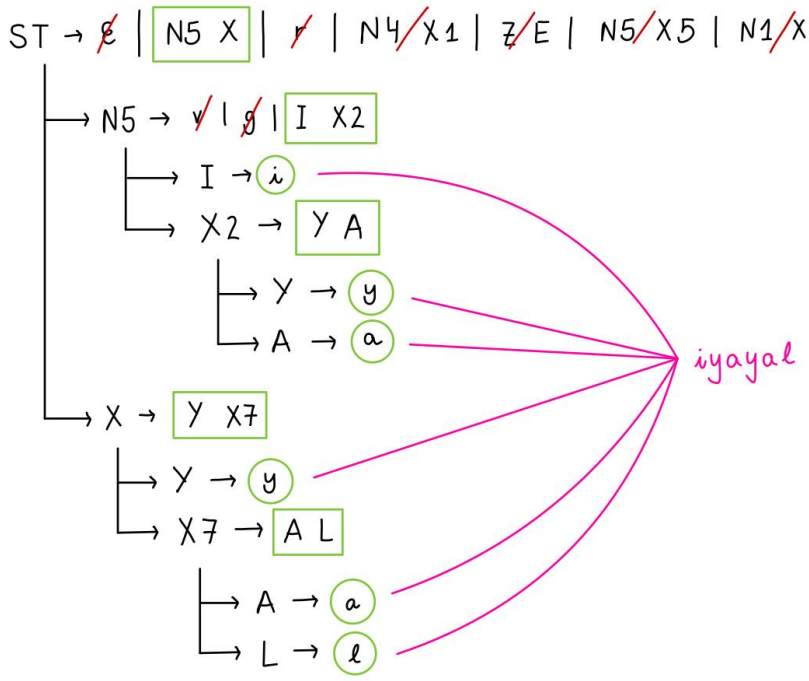

Figure 1: Demostració de la formació de la paraula amb la gramàtica original

Experiment 6

Paràmetres:
- Gramàtica en CNF ×
- Gramàtica probabilística ×
- Paraula vàlida ×

Gramàtica original:

$$
\begin{array}{rcl}
N9 & \to & \varepsilon \\
N9 & \to & N8\,l\,d\,b\,e \\
N9 & \to & k \\
N9 & \to & N5\,t\,l \\
N8 & \to & N10\,N10\,N7\,s \\
N8 & \to & N3\,N7\,t \\
N8 & \to & s\,x\,o \\
N7 & \to & l \\
N7 & \to & N4\,N4\,u
\end{array}
\qquad
\begin{array}{rcl}
N7 & \to & N1\,o\,d\,x \\
N7 & \to & N6 \\
N7 & \to & N1\,k\,i\,a\,e \\
N10 & \to & N5\,N6 \\
N10 & \to & N5\,d \\
N10 & \to & i \\
N5 & \to & k \\
N5 & \to & p\,f\,b \\
N5 & \to & j
\end{array}
\qquad
\begin{array}{rcl}
N5 & \to & l \\
N5 & \to & y \\
N4 & \to & N1 \\
N4 & \to & \varepsilon \\
N4 & \to & N10\,p \\
N6 & \to & y\,s\,l \\
N3 & \to & \varepsilon \\
N1 & \to & f
\end{array}
$$

Gramàtica transformada:

$$
\begin{array}{rcl}
ST & \to & \varepsilon \\
ST & \to & N8\,X1 \\
ST & \to & k \\
ST & \to & N5\,X2 \\
N8 & \to & N10\,X3 \\
N8 & \to & N7\,T \\
N8 & \to & S\,X4 \\
N7 & \to & l \\
N7 & \to & u \\
N7 & \to & F\,X5 \\
N7 & \to & F\,X6 \\
N10 & \to & N5\,Y \\
N10 & \to & N5\,D
\end{array}
\qquad
\begin{array}{rcl}
N10 & \to & i \\
N5 & \to & k \\
N5 & \to & P, X7 \\
N5 & \to & j \\
N5 & \to & l \\
N5 & \to & y \\
N4 & \to & N10\,P \\
N4 & \to & f \\
N7 & \to & Y\,X8 \\
L & \to & l \\
D & \to & d \\
B & \to & b \\
E & \to & e
\end{array}
\qquad
\begin{array}{rcl}
T & \to & t \\
S & \to & s \\
X & \to & x \\
O & \to & o \\
F & \to & f \\
K & \to & k \\
I & \to & i \\
A & \to & a \\
Y & \to & y \\
P & \to & p \\
X1 & \to & L\,X9 \\
X2 & \to & T\,L
\end{array}
\qquad
\begin{array}{rcl}
X3 & \to & N10\,X10 \\
X4 & \to & X\,O \\
X5 & \to & O\,X11 \\
X6 & \to & K\,X12 \\
X7 & \to & F\,B \\
X8 & \to & S\,L \\
X9 & \to & D\,X13 \\
X10 & \to & N7\,S \\
X11 & \to & D\,X \\
X12 & \to & I\,X14 \\
X13 & \to & B\,E \\
X14 & \to & A\,E
\end{array}
$$

Paraula: *ysctldbe*

Resultat : `False`

Table 6: Algoritme CKY a mà de l'experiment 6

y	s	c	t	l	d	b	e
N5, Y	×	×	×	×	×	×	×
	S	×	×	×	×	×	×
		×	×	×	×	×	×
			T	X2	×	×	×
				N7, N5, L	N10	×	X1
					D	×	X9
						B	X13
							E

4 Conclusions

En conclusió, durant aquesta pràctica s'ha realitzat l'apartat obligatòria d'implementar l'algorisme CKY. A més, s'han abordat amb èxit els apartats opcionals, que consisteixen en la implementació del CKY probabilístic i la conversió de gramàtiques tipus CFG a Forma Normal de Chomsky (CNF). A més a més, s'ha desenvolupat un codi addicional per generar gramàtiques i paraules de manera alàtoria.

Realitzar aquestes implementacions ens ha proporcionat una oportunitat per aplicar els coneixements adquirits en l'assignatura anterior de Programació Avançada (PA2), especialment en l'ús de classes per a crear programes ben estructurats. A més, hem aprofundit en la comprensió dels conceptes teòrics de l'assignatura de Processament del Llenguatge Humà (PLH), ja que els argorismes CKY son molt utilitzats en aquest àmbit i han sigut una part important del temari teòric de l'assignatura de PLH.

Destaquem especialment la utilització del defaultdict del mòdul collections de Python, una eina amb la qual no teníem experiència prèvia. La seva capacitat per gestionar valors per defecte en diccionaris ha resultat ser molt útil, millorant la robustesa i eficiència del nostre codi.

En resum, aquesta pràctica ha estat molt útil per consolidar els nostres coneixements en programació i teoria de llenguatges formals, també ens ha proporcionat noves eines que ens seran útils d'ara en endavant.

5 Bibliografía

1. Wikipedia contributors. (s/f). *Forma normal de Chomsky*. Wikipedia, The Free Encyclopedia. Recuperado de `https://ca.wikipedia.org/w/index.php?title=Forma_normal_de_Chomsky&oldid=32888197`

2. (S/f-b). Recuperado el 27 de mayo de 2024, de `http://chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cs.upc.edu/~turmo/plh/lectures/08-parsing-constituents.pdf`

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

4. Chomsky's Normal Form (CNF). (s/f). *Www.javatpoint.com*. Recuperado el 27 de mayo de 2024, de `https://www.javatpoint.com/automata-chomskys-normal-form`

5. Parde, N. [@NatalieParde_NLP]. (2020, diciembre 26). *CKY Algorithm*. Youtube. `https://www.youtube.com/watch?v=17re2zDBuOM`