FIB

Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Implementació de l'algoritme *Cocke-Kasami-Younger*

PROGRAMACIÓ I ALGORÍTMICA AVANÇADA

Autors:

Zhihao Chen; Zhiqian Zhou

31 de maig de 2024

Índex

1 Introducció

1.1 Objectiu de la Pràctica

L'objectiu de la pràctica és posar en pràctica els coneixements adquirits en les assignatures de Programació i Algorítmica Avançada (PAA) i de Processament del Llenguatge Humà (PLH). Concretament, es tracta d'implementar l'algoritme determinista *Cocke-Kasami-Younger* (CKY) utilitzant la programació dinàmica amb el llenguatge de programació *Python*.

1.2 Restriccions

Durant la pràctica, no es farà servir cap mòdul addicional de *Python* per ajudar directament o simplificar a la resolució del problema (com ara NLTK, ja que aquest inclou l'algoritme implementat).

S'ha fet servir el mòdul "copy", concretament, la seva funció "deepcopy" per evitar problemes de *aliasing* en realitzar algunes assignacions de variables.

1.3 Extensions Opcionals Implementades

Algunes extensions opcionals que s'han implementat durant la pràctica són:

- Transformació de qualsevol gramàtica lliure de context (CFG) a la forma normal de *Chomsky* (CNF).
- Algorisme CKY Probabilístic (PCKY).

1.4 Fitxers Inclosos en l'Entrega

En l'entrega de la pràctica s'inclouen els següents fitxers:

- Document descriptiu de la pràctica (*PAA_Pràctica.pdf*).
- Fitxer amb els codis de les implementacions dels algoritmes (*CKY_FNC_PCKY.py*).
- Fitxer amb el programa principal (*main.py*).
-
- Carpeta amb els fitxer amb els jocs de prova (*JOCS_PROVA*). El nom dels fitxers indica de quina extensió són.

Aquest conjunt de fitxers proporciona una descripció completa de la pràctica, així com el codi necessari per executar les implementacions dels algoritmes CKY i PCKY, juntament amb els jocs de prova per verificar el correcte funcionament dels mateixos.

2 Format de gramàtiques

És essencial la descripció de les gramàtiques lliures de context per a la pràctica. L'estructura de dades que s'ha decidit utilitzar per emmagatzemar la informació de les gramàtiques és el diccionari.

Les raons principals són la seva rapidesa en les consultes, amb un cost temporal constant, i l'alta interpretabilitat obtinguda gràcies al format de clau-valor.

Una gramàtica, que bàsicament és un generador de llenguatges, es pot expressar popularment de la manera següent:

$$A \to bA \tag{1}$$

De manera que A és un símbol no terminal i b un símbol terminal.

Els diccionaris tenen com a claus la part esquerra de la regla i com a valor una llista, els elements de la qual són els símbols de la part dreta de la regla. Aquests símbols són representats en forma de tuples, ja que, si la part dreta de la regla conté més d'un símbol, caldria identificar-los conjuntament.

Per exemple, la regla $A \to bA|a$ estaria codificada com $dict('S' : [('b','A'), ('a',)])$.

2.0.1 Gramàtiques en FNC

La gramàtica que s'introdueix com a entrada per l'algoritme de CKY ha de ser en *Chomsky Normal Form* FNC, per tant, se sap que les regles tindran aquestes formes:

$$A \to BC \tag{2}$$

$$A \to a \tag{3}$$

on les lletres majúscules són símbols no terminals i, les minúscules, terminals.

En la forma normal de *Chomsky*, la part dreta de la regla pot contenir o bé un símbol terminal o bé dos símbols no terminals. Per tal d'identificar cada generació, s'utilitzen tuples per emmagatzemar-los. El fet de tenir com a valors una llista es deu al fet que la part esquerra es pot repetir en diverses regles; és a dir, d'un símbol no terminal es poden construir diverses regles. Com que el valor és una llista, això també proporciona l'agilitat d'incorporar noves regles a partir d'un símbol no terminal existent gràcies a la seva propietat de mutabilitat.

Per exemple, per les regles anteriors, la codificació seria $dict('A' : [('B','C'), ('a',)])$. Cada tupla emmagatzema la generació de la clau del diccionari.

En cas que la regla sigui probabilística, només cal afegir-li la probabilitat a cada tupla i tancar-la en una altra tupla, com per exemple: $dict('A' : [(('B','C'), 0.6), (('a',), 0.4)])$.

3 Tractament i codificació de la paraula buida

La presència de la paraula buida en la gramàtica podria causar certs problemes; tanmateix, no es pot ignorar-la. En la forma normal de *Chomsky*, la paraula buida només pot ser generada pel símbol inicial; per tant, no es pot tractar-la simplement com un símbol terminal, cosa que si que fa el computador.

A l'hora de realitzar la transformació, una condició ideal és no haver de preocupar-se per les paraules buides si el llenguatge no genera la paraula buida. Per assolir aquesta condició, cal treure la paraula buida tan aviat com es pugui, generant una gramàtica G', que és idèntica a G, però que no genera la paraula buida. És a dir:

$$G' = G - \{< paraulabuida >\} \tag{4}$$

Durant la generació de G' també cal saber si la paraula buida pertany a G; d'aquesta manera, ja no serà necessari l'ús de l'algorisme de CKY per classificar la paraula buida.

Per altra banda, utilitzar una cadena buida ("") per representar la paraula buida pot causar problemes d'ambigüitat perquè pot confondre's amb l'absència de valor o de entrada. En el processament, les operacions amb cadenes buides poden ser diferents a les operacions amb cadenes no buides. Emprar alternatives com símbols especials ("ϵ") pot ajudar a evitar aquests problemes i assegurar una implementació més robusta i clara.

Pel que fa a les gramàtiques probabilístiques, s'assumeix que no contindran paraules buides per simplicitat, ja que manipular les transformacions de les probabilitats és massa complex.

4 Explicació dels codis

4.1 Inversió de diccionaris

En un algorisme CKY, és interessant obtenir la clau del diccionari (símbol no terminal generador) a partir d'un dels seus valors (símbols generats). Per això, es podria implementar una funció per invertir el diccionari.

La funció s'anomena "invertir_dicc", que rep un diccionari on els valors són llistes i retorna el diccionari invertit, on cada element de la llista original dels valors serà una clau per accedir a la seva clau en el diccionari antic.

Per assolir s'utilitzaran dos bucles *for*: en la primera s'accedeix a cada *item* del diccionari (*clau, valor*) i en la segona s'accedeix a cada element de la llista (*valor_i*), que és un iterable. Dintre del segon bucle es comprova si *valor_i* ja existia com una clau del nou diccionari. En el cas negatiu crea un conjunt amb l'element *clau* que té com a clau el *valor_i*, en el cas afirmatiu simplement afegeix la *clau* al conjunt corresponent al*valor_i*.

Cal esmentar que s'han utilitzat conjunts com a valors del diccionari invertit. Aquest fet té dos avantatges principals: evita elements repetits i problemes d'*aliasing*. Aquestes dues propietats seran convenients per a la implementació posterior de l'algorisme de CKY.

4.2 Algorisme de CKY

L'algoritme *Cocke-Kasami-Younger* (CKY) és capaç de determinar si una cadena pot ser generada per una gramàtica en forma normal de *Chomsky* (CNF). L'entrada de la funció CKY consisteix en la gramàtica en CNF codificada com un diccionari, tal com ja s'ha dit, i una llista dels components de la paraula que es vol analitzar

Per a una millor comprensió del codi, s'explica per parts:

- Per millorar l'eficiència, s'inverteix el diccionari que representa la gramàtica amb la funció *invertir_dicc*.
- Se sap que la gramàtica de l'entrada està en CNF i aquest només pot generar la paraula buida a partir de $S \rightarrow epsilon$, on S és el no terminal inicial. Per tant, si la paraula de l'entrada és la paraula buida, es retorna la comprovació de si epsilon (codificat com l'string buit) és clau de la gramàtica invertida.
- *word* és la llista de components de la paraula que s'analitza. Se li introdueix al davant un *None* per facilitar la lectura dels indexos.
- *table* és la taula amb tantes files i columnes com components de paraula hi hagi que emmagatzema els resultats dels subproblemes, característic de la programació dinàmica. Es podria considerar com taula imaginària en forma d'esglaó, ja que aquest està codificat com un diccionari, garantint la facilitat i interpretabilitat de les consultes, i les "caselles" que no es generen

no existeixen. Cada casella de la taula té dos indexos: inici "i" i fi "j" com a posicions dels components de la paraula. Aleshores, la casella emmagatzemarà el símbol o símbols no terminals amb els quals, apuntant a caselles provinents, s'arriba a abarcar tots els components del rang entre inici i fi (ambdos inclosos). Les caselles de la primera fila abarquen només el component senyalat per l'índex, ja que "i"="j". En la segona fila "j"="i"+1, per tant cada casella abarca dos components consecutius. I així fins a arribar a la última fila. Per tant, si s'arriba a crear la casella que abarca tota la paraula, és a dir, des del primer component fins a l'últim, aleshores la paraula pot ser generada per la gramàtica.

- El primer bucle inicialitza la primera fila de la taula. Tal com ja s'ha dit, la primera fila té els dos indexos iguals que vol dir que abarca només el component de la paraula de la posició dels indexos. Per tant, dins de la casella hi hauran els símbols no terminals corresponent a l'estructura $A \to a$ on "A" és el símbol no terminal i "a" representa el component de la paraula tractada.

- "x" itera sobre les files començant des de la segona.

- "i" representa el primer índex de les caselles. La taula té forma d'esglaó, per tant "i" també podria representar la columna on es troba tenint en compte que el nombre de columnes de cada fila va decreixent.

- "j" és "i"+"x" i és el segon índex de les caselles. Vol dir que a cada fila es va augmentant els components que abarca.

- "k" itera sobre els índexos intermedis entre "i" i "j". Per poder obtenir el símbol no terminal que abarca entre "i" i "j", cal comprobar si hi ha un regle $A \to BC$ on "B" cobreixi entre "i" i "k" i "C" entre "k+1" i "j". Si fos així, "A" seria el símbol no terminal que abarca entre "i" i "j".

- Cal comprovar si la taula conté (i,k) i (k+1,j). Si aquests índexos no estan creats, vol dir que no s'havia trobat cap símbol que abarqués aquell rang. Per tant, no es podrà obtenir informació entre "i" i "j" amb (i,k) i (k+1,j), però seria possible quan es canvii el valor de "k".

- Una casella pot contenir múltiples símbols no terminals guardats en un conjunt (per evitar repeticions). Pot ser que entre "i" i "j" hi hagués diferents valors de "k" on es trobi regles que compleixen amb (i,k) i (k+1,j). Per tant, cal iterar en tots els símbols de la casella (i,k) i de la casella (k+1,j).

- Si la combinació dels símbols de (i,k) i (k+1,j) surt com a clau en la gramàtica invertida, aleshores vol dir que es pot generar aquests símbols a partir d'un no terminal de la gramàtica.

- Si l'índex (i,j) existeix a la taula, a la casella corresponent se li afegirà el valor de la combinació dels símbols de (i,k) i (k+1,j) en la gramàtica invertida.

- En cas contrari, no se li afegirà sinò que es crearà.

- Finalment, es comprova si la casella de la última fila, que és la que abarca tota la paraula, ha estat creat i conté el símbol no terminal inicial. En cas afirmatiu, retorna True. En cas contrari vol dir que cap combinació ha arribat a cobrir tota la paraula partint del símbol no terminal inicial i retorna False.

Al moment d'afegir el símbol no terminal que genera símbols de les posicions (i,k) i (k+1,j) a la casella (i,j), és possible que aquest ja estigués contingut gràcies a altres produccions. Com només interessa saber si la paraula pot ser generada per la gramàtica o no, la repetició de símbols en una mateixa casella no té cap sentit pràctic. A més, la creació de la casella (i, j) pren directament el valor del diccionari que representa la gramàtica invertida. Si aquest valor fos una llista, s'hauria de fer còpies per evitar *aliasing*, cosa que complicaria el codi.

Per aquests motius, s'ha optat per utilitzar conjunts en lloc de llistes per representar els valors del diccionari invertit. Això simplifica la gestió dels símbols que generen les subcadenes i millora l'eficiència de l'algorisme CKY.

4.3 Transformació a la FNC

Les entrades per a l'algorisme de CKY han de ser en forma normal de *Chomsky*. En la implementació anterior de CKY s'havia assumit aquesta condició. En aquesta extensió de la pràctica, s'ha decidit crear un procés de processament de gramàtiques perquè puguin transformar-se a la seva forma FNC.

Aquest procés es basa en les teories estudiades a l'assignatura de PLH i en informació recopilada de la *Wikipedia*, que proporciona els passos necessaris per a la conversió. Segons aquestes fonts, la transformació consta de cinc passos clau.

- **START:** Introduir un nou símbol inicial que genera el símbol inicial anterior
- **DEL:** Eliminar regles que generen paraules buides
- **TERM:** Eliminar regles híbrides: substituir terminals per nous terminals en les regles que generen un terminal amb els altres símbols
- **UNIT:** Eliminar regles unitàries
- **BIN:** Eliminar regles no binàries

És crucial realitzar els passos en un ordre específic, ja que alguns poden interferir amb els resultats dels altres. Segons la informació proporcionada per la Wikipedia, s'ha establert una relació entre els passos de la transformació a la forma normal de Chomsky (FNC). Aquest ordre assegura una conversió efectiva de la gramàtica:

Mutual preservation of transformation results [hide]

Transformation *X* always preserves (✓) resp. may destroy (✗) the result of *Y*:

X \ Y	START	TERM	BIN	DEL	UNIT
START		✓	✓	✗	✗
TERM	✓		✗	✓	✓
BIN	✓	✓		✓	✓
DEL	✓	✓	✓		✗
UNIT	✓	✓	✓	(✓)*	

*UNIT preserves the result of **DEL** if **START** had been called before.

Figura 1: Preservació mútua dels resultats de la transformació

Segons les directrius de l'assignatura de PLH, es decideix ometre el pas de *START* en la transformació de la gramàtica a la forma normal de *Chomsky* (FNC), ja que no es va tractar durant el curs. No obstant això, es determina que el pas de *DEL*, que tracta el tractament de les paraules buides, és rellevant per a la conversió i, per tant, es mantindrà a la transformació.

Conversión a FNC

1. Reglas híbridas: sustituir terminales por nuevos no terminales

   P.e.: $\text{INF\_VP} \rightarrow \text{to VP } (p_1) \implies \text{INF\_VP} \rightarrow \text{TO VP } (p_1)$, $\text{TO} \rightarrow \text{to } (1.0)$

2. Reglas unitarias:

   P.e.: $\text{NP} \rightarrow \text{N } (p_1)$, $\text{N} \rightarrow \text{dog } (p_2) \implies \text{NP} \rightarrow \text{dog } (p_1 * p_2)$

3. Reglas no binarias:

   P.e.: $\text{S} \rightarrow \text{VP NP PP } (p_1) \implies \text{S} \rightarrow \text{VP X } (p_1)$, $\text{X} \rightarrow \text{NP PP } (1.0)$

Figura 2: Conversió a FNC explicada a PLH

Per tant, els passos de conversió que es duran a terme són *DEL*, *TERM*, *UNIT* i *BIN*. Aquest ordre de transformació s'ha seleccionat per garantir que els resultats es preservin correctament i que la gramàtica es converteixi de manera adequada a la FNC. Amb aquesta seqüència de passos, es podrà assegurar una transformació eficient i precisa de la gramàtica.

La implementació de la transformació a la forma normal de *Chomsky* (FNC) es basa en una classe que proporciona un mètode disponible per als usuaris per realitzar la conversió de la gramàtica. A

més, aquesta classe conté un conjunt de mètodes auxiliars que no estan disponibles per als usuaris, però que són necessaris per dur a terme la transformació de manera correcta.

Aquests mètodes auxiliars són responsables de realitzar les diverses etapes de la transformació, com ara eliminar les regles que generen paraules buides, substituir terminals en regles híbrides, eliminar regles unitàries i eliminar regles no binàries. Encara que els usuaris no tinguin accés directe a aquests mètodes, són essencials per a la implementació adequada de la transformació:

- La inicialització de la classe requereix una gramàtica representada en forma de diccionari, com es va esmentar anteriorment, i un símbol que representa el símbol inicial de la gramàtica.
- El mètode "CFG2CNF" crida a les altres mètodes auxiliars i retorna la gramàtica en forma normal de *Chomsky*, juntament amb un valor booleà que indica si el llenguatge generat conté la paraula buida o no. També ofereix una opció de "print_grammar" per escriure la gramàtica transformada de manera més interpretable al terminal, utilitzant fletxes i barres. És important destacar que tots els mètodes auxiliars són destructius, per tant, al cridar el mètode "CFG2CNF", es realitzarà una còpia densa (*deepcopy*) de la gramàtica i els canvis es processaran sobre aquesta còpia. Per tant, s'ha optat el mòdul "copy" del *Python*.
- El mètode "__represent_grammar" és el que s'encarrega de imprimir la gramàtica a la sortida. És tan senzill com llegir els *items* del diccionari i imprimir-los de forma

$$clau \rightarrow valor_0|...|valor_i|...|valor_n \tag{5}$$

- El mètode "__rem_eps" correspon a la transformació *DEL* i s'encarrega d'eliminar totes les regles que generen la paraula buida (ϵ) i de fer els ajustos necessaris per mantenir el mateix llenguatge sense aquestes regles.

  En primer lloc, elimina totes les regles que generen únicament la paraula buida. Com que no es poden esborrar elements d'una llista dins d'un bucle *for*, substitueix la part generada de la regla per "(None, None)", que serveix com a identificador per eliminar-les posteriorment. Si després d'aquesta eliminació el no terminal generador no produeix cap altra cosa, esborrarà directament el no terminal de les regles. Això assegura que es comencen eliminant les regles que només produeixen la paraula buida, les quals no afecten altres parts de la gramàtica.

  Després, es tracten les regles que inclouen algun no terminal que genera la paraula buida. En aquests casos, cal incorporar una nova regla que ignori el no terminal que genera ϵ, és a dir, es genera la mateixa regla sense aquest no terminal. Si una regla conté múltiples no terminals que generen la paraula buida, s'han de considerar totes les combinacions possibles de suposicions d'ignoració. Això es fa trobant totes les combinacions de posicions dels no terminals a ignorar. Després d'eliminar les posicions corresponents en la regla, s'afegeix com una nova regla. Si la regla inclou un no terminal que ja ha estat eliminat anteriorment, també s'ignorarà perquè ja no existeix en les regles.

  Aquest procés assegura que es fan tots els canvis necessaris per mantenir el mateix llenguatge, eliminant alhora totes les paraules buides restants. Així es garanteix que, tot i eliminar les regles

que generen ϵ, la gramàtica continua produint les mateixes cadenes, mantenint la coherència del llenguatge.

- El mètode "__rem_hib" correspon al pas de *TERM* de la transformació, és a dir, elimina les regles híbrides de la gramàtica. Com que la mida dels diccionaris no es pot canviar durant el bucle, tots els canvis que es realitzin s'emmagatzemen a un conjunt local anomenat "later_change". El "later_change" pot emmagatzemar tuples o bé de mida 2, o bé de mida 4. Quan la mida és 2, es tracta d'una nova regla a incorporar, on el primer element de la tupla correspon al nou símbol no terminal creat i el segon és el símbol terminal generat. Si la mida és 4, es tracta de substituir els símbols terminals de la part dreta de la regla. La tupla conté el símbol no terminal amb la regla que cal tractar, la posició de la regla dins de la llista de regles associades al símbol, la posició del no terminal que cal substituir dins de la tupla que representa la regla i un nou símbol no terminal de substitució.

  Els elements del "later_change" han de provenir d'una regla de mida major a 1, la qual cosa significa que genera una juxtaposició de símbols, i han de contenir almenys un símbol no terminal. Això assegura que només es considerin les regles que contenen combinacions de símbols i que es requereixi almenys un símbol no terminal perquè es pugui realitzar una substitució.

- El "__rem_unit_prod" correspon al pas de *UNIT* de la transformació i té com a objectiu eliminar les produccions unitàries de la gramàtica. Per a això, utilitza una llista anomenada "unit_productions" que actua com una pila i emmagatzema les produccions unitàries que es troben a la gramàtica. Per trobar les regles unitàries, simplement recull totes les regles que generen només un símbol no terminal. Quan es troba una regla unitària, aquesta es canvia per una tupla "(None, None)", que actua com a identificador perquè el mètode pugui eliminar-la del diccionari més endavant (ja que no es pot eliminar directament dins d'un bucle *for*). Una vegada s'han trobat totes les regles unitàries, es procedeix a afegir les regles corresponents al no terminal que es va eliminar al símbol no terminal que la genera. Durant aquest procés d'afegiment, és possible que es creïn noves regles unitàries, les quals es tornen a afegir a la llista "unit_productions". El bucle acaba quan la llista "unit_productions" està buida.

  Així doncs, el mètode "__rem_unit_prod" simplement elimina les regles unitàries i incorpora les regles que genera el no terminal resultant al no terminal generador de la regla unitària.

- El "__rem_no_bin" correspon al pas de *BIN* de la transformació, i té com a objectiu eliminar les regles no binàries de la gramàtica, específicament aquelles que generen més de dos símbols. El funcionament del mètode és senzill: crea un diccionari auxiliar per substituir posteriorment al diccionari de la gramàtica, i durant un bucle *while*, revisa si hi ha tuples de mida major a 2. Si troba tuples de mida major a 2, les descomposa iterativament fins que no n'hi hagi. Mentre es realitza aquest procés, el mètode afegeix les regles binàries descomposades o originals, així com les unitàries (que només generen no terminals), al nou diccionari auxiliar.

- El mètode "__get_new_non_terminal" simplement retorna un nou símbol no terminal codificat com un nombre natural. Aquest mètode s'utilitza quan cal crear noves regles amb nous símbols

no terminals. Això implica assumir que els símbols no terminals originals no són nombres naturals, ja que aquests s'utilitzen com a base per a la generació dels nous símbols no terminals.

- El "__TrobarProduccionsEpsilon" identifica els símbols no terminals que poden generar la paraula buida i els guarda en el conjunt "nullable". Primer, detecta els no terminals que directament produeixen ϵ. Després, construeix iterativament el conjunt "nullable", afegint-hi els no terminals que poden generar ϵ indirectament a través de símbols ja presents en "nullable". Aquest procés continua fins que no es poden afegir més símbols, assegurant que tots els no terminals capaços de generar ϵ estan identificats.

  Aquesta informació és essencial per ajustar correctament la gramàtica durant la transformació a la FNC, mantenint la consistència del llenguatge generat.

- El "__TrobarNomesEpsilon" identifica els símbols no terminals que únicament generen la paraula buida i els guarda en el conjunt "only_eps". Funciona de manera similar al mètode "__TrobarProduccionsEpsilon", però amb una diferència clau: ha d'iterar sobre totes les regles associades a un no terminal per assegurar-se que efectivament només generen ϵ.

  Aquest mètode és especialment útil durant la transformació de la gramàtica, ja que permet tractar les paraules buides de manera precisa i eficient, garantint que els no terminals que només produeixen ϵ siguin correctament identificats i manejats.

- El "__powerset"retorna el conjunt potència de l'entrada que rep. A més de l'entrada, que és el conjunt original, també rep un paràmetre current que ha de ser inicialitzat a zero per al correcte funcionament del mètode, i el tamany del conjunt original. Aquest mètode utilitza l'estratègia de *backtracking* per generar de manera eficient totes les combinacions possibles del conjunt, formant així el conjunt potència.

4.4 Algorisme de PCKY

La incorporació d'elements probabilístics a l'algorisme de CKY representa una ampliació significativa, permetent que l'algorisme no només determini la validesa sintàctica d'una frase, sinó que també proporcioni una estimació de la probabilitat de generació de cada arbre sintàctic possible. En aquesta pràctica, per simplificar, l'algorisme de PCKY implementat només retorna un booleà que indica si la gramàtica pot generar la frase d'entrada i la probabilitat més alta associada amb la generació d'aquesta frase.

Aquesta extensió de l'algorisme requereix un canvi en el format de les gramàtiques. El format anterior, basat en un diccionari de llistes de tuples, ja no és adequat. En lloc d'això, cal utilitzar un diccionari de llistes de tuples de dos elements. En aquest nou format, el primer element de cada tupla representa els símbols generats per la regla, mentre que el segon element representa la probabilitat associada amb la regla.

Per simplificar encara més, es fa l'assumpció que les entrades ja es troben en forma normal de *Chomsky*. Això permet que l'algorisme es centri exclusivament en l'anàlisi sintàctic (*parsing*) i en el

càlcul de les probabilitats de generació, sense la necessitat de preocupar-se per altres transformacions gramaticals.

El mètode "invertir_dicc" manté les claus com els símbols generats, però els valors ara són un conjunt de tuples de dos elements: el símbol generador i la probabilitat associada. Això permet incorporar la informació de probabilitats a la representació de la gramàtica invertida.

Quant als canvis al mètode "algorithm", s'han realitzat ajustos específics per acomodar aquesta informació de probabilitats. Ara, en el moment de canviar el contingut de la casella, es realitza una multiplicació entre les dues probabilitats dels símbols generats i la probabilitat del símbol generador, prenent en compte les regles que generen els símbols corresponents i que compleixen les condicions per ser escollits com a caselles objectiu (amb posicions (i, k) i (k+1, j), respectivament).

Posteriorment, després de calcular els elements de la casella, es crida al mètode "preproc_dicc" per filtrar aquelles tuples que no tenen la probabilitat més alta per als mateixos símbols generats.

Finalment, els elements que retorna el mètode "algorithm" ara consisteixen en un valor booleà que indica si les paraules poden ser generades per la gramàtica o no, i quina probabilitat se li associa. Per tant, si la gramàtica no pot generar la paraula, retorna "(False, 0)".

5 Extensions realitzades i testos

El programa principal llegeix les entrades, i segons l'extensió a la qual es refereix, anirà creant objectes de classes diferent i cridant mètodes corresponents.

Els testos tenen com a objectiu de comporvar el funcionament correcte de l'algorisme. Els exemples mostrats als jocs de prova són majoritàriament del *RACSO*, la qual cosa també proporciona un *parser* internament. Per tant, sempre es pot verificar els resultats dels exemples disponibles.

Els testos i els paràmetres del programa principal s'hauran d'executar canviant manualment el fitxer objectiu d'entrada en el fitxer *main.py*. Lastimament, no va donar temps per fer els testos per a l'extensió 2 de manera formal. No obstant això, pel que fa a l'extensió 2 és proporcionar directament les entrades que puguin ser reconegudes pel algoritme, és a dir, en forma de diccionaris. Les sortides de les quals són també introduïdes manualment.

5.1 Problema principal

En el problema principal es demana la implementació de l'algoritme CKY utilitzat per analitzar l'estructura sintàctica de cadenes de text, i determinar si poden derivar-se a partir de les regles d'una gramàtica.

L'algoritme pren com a entrada una gramàtica de tipus Context-Free Grammar (CFG) en CNF, el símbol que pren com a inicial i la paraula. Aleshores, retorna un booleà determinant si la paraula pertany o no al llenguatge de la gramàtica.

5.1.1 Test

Per testejar el programa, s'han volgut analitzar diversos aspectes:

- Si l'algoritme pot analitzar bé el símbol inicial. La paraula ha de ser formada a partir del símbol inicial, per tant, si la paaraula pot derivar-se d'un altre símbol no terminal però no de l'inicial es considera que la paraula no és del llenguatge de la gramàtica.
- Si es compleixen el respecte a l'odre de generació.
- Amb gramàtiques que se segueixen uns patrons clars, es proven diversos casos per comprovar l'efectivitat (com pot ser la composició de a's i b's).
- Anàlisi de la detecció de paraula buida.

5.2 Extensió 1

En l'extensió 1, es fa la transformació de qualsevol gramàtica CFG a CNF. Per tal d'aconseguir-ho, s'ha creat la classe FNC.

5.2.1 Test

Per testejar el programa, s'han volgut analitzar diversos aspectes:

- Altra vegada, es comprova si l'algoritme pot analitzar bé el símbol inicial.
- Es comprova que el pas a FNC estigui correcte.
- Es comprova que, per casos extrems, es pugui generar la gramàtica en FNC i d'aquí arribar a obtenir la sortida adequada.
- Si es compleixen el respecte a l'odre de generació.
- Amb gramàtiques que se segueixen uns patrons clars, es proven diversos casos per comprovar l'efectivitat (com pot ser el palíndrom).
- Anàlisi de la detecció de paraula buida. En l'extensió 1, el CKY no és qui determina si la paraula buida està al llenguatge, sinò que se sap quan s'elimina l'epsilon de la gramàtica.

6 Conclusió

La pràctica ha estat una immersió fascinant en la comprensió i implementació d'algorismes d'anàlisi sintàctica mitjançant la programació dinàmica, amb un enfocament particular en l'algorisme de *Cocke-Kasami-Younger* (CKY) i la seva extensió a l'algorisme probabilístic (PCKY). Hem explorat com aquest algorisme pot determinar la validesa sintàctica d'una frase i proporcionar una probabilitat associada amb cada arbre sintàctic possible.

En primer lloc, la transformació de les gramàtiques en forma normal de *Chomsky* ha estat essencial per a la correcta aplicació de l'algorisme CKY. Aquesta transformació ha simplificat significativament el procés d'anàlisi sintàctic, ja que permet treballar amb regles estandarditzades i facilita la implementació de l'algorisme.

L'ús de l'estructura de dades basada en diccionaris ha estat clau en la representació de les gramàtiques i en la implementació de l'algorisme CKY. Aquesta estructura de dades ofereix una manera eficient de guardar les regles de la gramàtica i facilita l'accés i la manipulació d'aquestes regles durant l'anàlisi sintàctic.

La incorporació d'elements probabilístics ha afegit una dimensió addicional a l'algorisme CKY, permetent calcular la probabilitat de generació de les frases analitzades. Mitjançant el càlcul de les probabilitats associades a les regles de la gramàtica, s'ha pogut determinar la probabilitat més alta de generació per a cada frase, oferint una mesura de confiança en la validesa sintàctica de les mateixes.

A més, l'aplicació de tècniques de programació dinàmica ha contribuït a l'eficiència de l'algorisme CKY, permetent evitar el recalcul de subproblemes ja resolts i reduint així el temps d'execució de l'algorisme. Aquest enfocament ha estat crucial per a l'aplicació pràctica de l'algorisme CKY en l'anàlisi de frases llargues i complexos.