BBM497: Introduction to Natural Language Processing Lab. **(Due: 02/05/2020)**

Submission Assignment #3

Instructor: Burcu CAN - Necva BÖLÜCÜ *Name:* Okan ALAN, *Student No:* 21526638

1 Introduction

In this report, I discuss my approaches to given tasks and data structures that I used. This report containing how I deal with the given task.

In this assignment, our goal was to learn the Cocke–Younger–Kasami algorithm (alternatively called CYK, or CKY) is a parsing algorithm for context-free grammars, named after its inventors, John Cocke, Daniel Younger, and Tadao Kasami. It employs bottom-up parsing and dynamic programming.

2 Context Free Grammar(CFG) Rules

In this section is related to "rules" function.

In formal language theory, a context-free grammar (CFG) is a formal grammar in which every production rule is of the form $A \rightarrow \alpha$ where A is a single nonterminal symbol, and α is a string of terminals and/or nonterminals (α can be empty). A formal grammar is considered "context free" when its production rules can be applied regardless of the context of a nonterminal.

I am a careless person, when I started to coding for this homework, I tried to handle rules which are non-terminals and terminals together such as "$ROOT \rightarrow$ is it true that S ?".End of the day I handled them but there were a lot of if condition and bad coded places in my code. This was caused to be trash my code. Then I wondered how my friends handled this situation. They said that read the assignment pdf. The next sentence is from assignment pdf; a sample CFG rule set is provided for you in the Chomsky Normal Form(CNF). The format of the CFG rules is as follows:

S NP VP #S → NP VP
Noun book #Noun → book

That part of pdf said to me, I don't need to handle mix type lines. Therefore I shouldn't consider ROOT lines. But I am going to explain how I handled this situation. Let's say, our rule is "ROOT is it true that S ?". "?" and "true" aren't our terminal rules. First of all, I have to put them to CNF. I added new terminal rules such as "IRREGULAR0 true". Then, I converted terminals to nonterminals such as "is it true that S ?" → "ROOT Verb NP IRREGULAR4 Det S IRREGULAR5 " Then I start to merge two by two them because of according to above the first CNF rule. Then it looks like
ROOT VP IRREGULAR4 Det S IRREGULAR5
ROOT IRREGULAR6 Det S IRREGULAR5
ROOT IRREGULAR7 S IRREGULAR5
ROOT IRREGULAR8 IRREGULAR5
ROOT IRREGULAR9

Those rules are added to my rules dictionary. I submitted both codes. Firsts one is handled.py which related to the above explanation, the other one is main.py which accepts only valid rules.

I have 2 different dictionaries to store rules. One of them is called cfg_rules dictionary. Its structure looks like { Nonterminal : ["Nonterminal Nonterminal", "terminal"] } such as { "VP" : ["Verb NP"], "Pronoun" : ["president","sandwich"] }. Other is magic to use dynamic programming. CYK algorithm need to memoize previously calculated things. Its structure is the exact opposite of cfg_rules such as { "this" : "Det" }.

3 Cocke–Younger–Kasami(CYK) Algorithm

In computer science, the Cocke–Younger–Kasami algorithm (alternatively called CYK, or CKY) is a parsing algorithm for context-free grammars, named after its inventors, John Cocke, Daniel Younger and Tadao Kasami. It employs bottom-up parsing and dynamic programming.

i	want	you
$X_{1,1}$	$X_{2,2}$	$X_{3,3}$
$X_{1,2}$	$X_{2,3}$	
$X_{1,3}$		

i	want	you
NP	Verb	NP
$\emptyset$	VP	
S		

$$X_{1,2} = (X_{1,1}, X_{2,2})$$
$$X_{2,3} = (X_{2,2}, X_{3,3})$$
$$X_{1,3} = (X_{1,1}, X_{2,3}) \bigcup (X_{1,3}, X_{3,3})$$

CYK algorithm split sentence to 2 parts to verify that does the language contains a given sentence. Let's say our sentence is "i want you". CYK examines "(i) - (want you)", "(i want) - (you)". If one of them is grammatically correct, CYK said that it is a valid sentence for that language. This sentence is grammatically verified sentence by our rules. "(i)-(want you)" → "(NP)-(VP)" → "(S)". To do that, we have to calculate separately tiny pieces. Also, I print the above triangular matrix to output file.

4 Generate Sentence

When I first wrote this part at an assignment, randomly generated sentences were never valid grammatically. Therefore I created a coin. I throw money, if it comes to head, I generate cfg-rule-based sentence, otherwise, I generate sentences randomly from nonterminals.

4.1 Rule-Based Sentence Generation

I create a queue then I put "S"("ROOT" in handle.py) into the queue. After that, I start to travel in the queue. If a visiting element is terminal, that element goes to the tail of the queue. If the visited element is nonterminal, I put a corresponding terminal or nonterminal from cfg rules at the end of the queue. Then I delete it. These iterations continue until there is no deletion. Let's say I'm visiting "S", it is nonterminal. I put "NP", "VP" to the tail of the queue. The iteration is below.

Iteration Number	Inside of Queue
1	["S"]
2	["NP","VP"]
3	["NP","PP","Verb",""Pronounn"]
4	["Pronoun","Prep","NP","like","me"]
5	["you","with","Det","Noun","like","me"]
6	["you","with","this","sandwich","like","me"]

5 Pseudo-Code

1. Read cfg rules
2. Parse cfg rules
3. Generate sentence via cfg rules
4. Check the generated sentence is valid