**CS-5340/6340, Programming Assignment #2**
**Due: Friday, October 2, 2020 by 11:59pm**

Your task is to build a syntactic parser from scratch using the **CKY parsing algorithm**. Your parser should accept two input files: (1) a probabilistic context-free grammar (PCFG) file, and (2) a sentences file. Your program should be named **"cky"** and should accept the files as command-line arguments in the order above. We should be able to run your program like so:

*python3 cky.py pcfg.txt sentences.txt*

If you use Java, the arguments should similarly be accepted on the command-line in the same order.

---

## The PCFG File

The PCFG file will contain probabilistic context-free grammar rules in CNF form. You can assume that each rule will consist of exactly two non-terminal symbols on its right-hand side, or exactly one terminal symbol on its right-hand side. The left-hand side of each rule will be separated from the right-hand side by the symbols "->". Each line will contain a grammar rule followed by its probability.

Here is a sample grammar file:

```
S -> NP VP .80
S -> VP NP .20
NP -> I .37
NP -> fish .128
NP -> trust .172
NP -> shrinks .33
VP -> fish .4
VP -> shrinks .3025
VP -> trust .2975
```

---

## The Sentences File

The sentences file will contain sentences that you should give to your parser as input. The words will be separated by whitespace. Each sentence will be on a separate line.

A sample sentences file is shown below:

```
I fish
trust shrinks
```

Your program should treat the words as **case sensitive** (e.g., the words "dog", "Dog", and "DOG" are *different* words).

## CKY Parsing Algorithm

Your program should implement the CKY parsing algorithm to identify all possible parses for a sentence. You should not use any probabilities for this, so you can ignore the probabilities in the PCFG file.

## Output Specifications

For each input sentence, your program should print three things to standard output:

1. **PARSING SENTENCE: <sentence>**
2. **NUMBER OF PARSES FOUND: <number>**
   The number of parses corresponds to the number of $S$ constituents found in cell[1,N] of the table, where N is the length of the input sentence. If no legal parses can be found for the sentence, then print a zero.
3. **TABLE:** followed by the contents of each cell in the table on a separate line.

   For an NxN matrix, the CKY algorithm only uses the cells for the [i][i] diagonal and above that, so please only print the cells used by CKY in your output. Print the table entries for the rows from top to bottom. Within a row, print the entries from left to right (i.e., corresponding to reading the sentence from left to right).

   For each cell[r][c], print the list of constituents (non-terminals) that have been successfully produced by the parser for words r through c. Important: please print the non-terminals in alphabetical order! If NO constituents have been produced, then print a hyphen (-).

For example, given a sentence of length 3, your output should be formatted like this:

```
PARSING SENTENCE: <sentence>
NUMBER OF PARSES FOUND: <num>
TABLE:
cell[1,1]: <entries>
cell[1,2]: <entries>
cell[1,3]: <entries>
cell[2,2]: <entries>
cell[2,3]: <entries>
cell[3,3]: <entries>
```

Here is what the output should look like for the grammar and sentences shown earlier:

```
PARSING SENTENCE: I fish
NUMBER OF PARSES FOUND: 1
TABLE:
cell[1,1]: NP
cell[1,2]: S
cell[2,2]: NP VP

PARSING SENTENCE: trust shrinks
NUMBER OF PARSES FOUND: 2
TABLE:
cell[1,1]: NP VP
cell[1,2]: S S
cell[2,2]: NP VP
```

## CS-6340 Students

In addition to the non-probabilistic CKY parsing algorithm, CS-6340 students should also implement the probabilistic CKY parsing algorithm. For probabilistic CKY, your parser should use the probabilities in the PCFG to determine the most probable parse for a sentence.

Please submit both algorithms in the same code base so that we can invoke the probabilistic CKY algorithm by adding a "-prob" argument on the command line, after the PCFG and sentences file names. For example, we should be able to run your probabilistic CKY parser using this command:

*python3 cky.py pcfg.txt sentences.txt -prob*

(Or similarly if you use Java.)

## Output Specifications

Your probabilistic CKY parser should print its output in the same format as for the non-probabilistic parser, except that it should also print the probability of each constituent in parentheses. Print the probabilities with exactly 4 digits after the decimal point, e.g. NP(0.2000), and use rounding. For example, the value 0.366666 should be printed as 0.3667, and 0.366432 should be printed as .3664.

This is what the output should look like for the grammar and sentences shown earlier using probabilistic CKY:

```
PARSING SENTENCE: I fish
NUMBER OF PARSES FOUND: 1
TABLE:
cell[1,1]: NP(0.3700)
cell[1,2]: S(0.1184)
cell[2,2]: NP(0.1280) VP(0.4000)

PARSING SENTENCE: trust shrinks
NUMBER OF PARSES FOUND: 1
TABLE:
cell[1,1]: NP(0.1720) VP(0.2975)
cell[1,2]: S(0.0416)
cell[2,2]: NP(0.3300) VP(0.3025)
```

## SUBMISSION INSTRUCTIONS

On CANVAS, please submit an archived (.tar) and/or zipped (.gz or .zip) file containing:

1. The source code for your CKY parser(s). Be sure to include all files that we will need to compile and run your programs!
2. A README file that includes the following information:
   - how to compile and run your code
   - which CADE machine you tested your program on (this info may be useful to us if we have trouble running your program)
   - any known bugs, problems, or limitations of your program
3. Run your CKY program on the **CADE** machines using the input files in the Program #2 folder on CANVAS. Please submit the output files that are produced using the same name as the sentences input file with the extension ".cky". For example:

   *python3 cky.py pcfg.txt sentences.txt > sentences.txt.cky*

   **CS-6340 Students:** You should also produce output files for your probabilistic CKY algorithm and name your output file with the extension ".probcky". For example:

   *python3 cky.py pcfg.txt sentences.txt > sentences.txt.probcky*

**Important:** Your program must be written in Python or Java and it **MUST** compile and run on the linux-based CADE machines! We will not grade programs that cannot be run on the linux-based CADE machines.

---

## GRADING CRITERIA

Your program will be graded based on new input files! **So please test your program thoroughly to evaluate the generality and correctness of your code!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on new input.

**Please** exactly follow the formatting instructions specified in this assignment. We will deduct points if you fail to follow the specifications because it makes our job much more difficult to grade programs that do not conform to the same standards.

**IMPORTANT:** You may not use ANY external software packages or dictionaries to complete this assignment except for *basic* libraries. For example, libraries for general I/O handling, math functions, and regular expression matching are ok to use. NumPy is ok to use. But you may not use libraries or code from any NLP-related software packages, or external code that performs any functions specifically related to syntactic parsing. All submitted code *must be your own.*

## HELPFUL HINTS

**TAR FILES:** First, put all of the files that you want to submit in a directory named "program2". Then from the parent directory where the "program2" folder resides, issue the following command:

tar cvfz program2.tar.gz program2/

This will put everything that is inside the "program2/" directory into a single file called "program2.tar.gz". This file will be "archived" to preserve structure (e.g., subdirectories) inside the "program2/" directory and then compressed with "gzip".

FYI, to unpack the gzipped tarball, move the program2.tar.gz to a new location and issue the command:

tar xvfz program2.tar.gz

This will create a new directory called "program2" and restore the original structure and contents.

For more general information on the "tar" command, this web site may be useful:
https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/