CS-5340/6340, Programming Assignment #2
Due: Thursday, September 27, 2018 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 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, optionally followed by a "-prob" flag. If "-prob" is absent, then your program should use the ordinary (non-probabilistic) CKY algorithm. If "-prob" is present, then your program should use the probabilistic CKY algorithm. Consequently, we should be able to run your program as:

cky <pcfg_file> <sentences_file>
OR
cky <pcfg_file> <sentences_file> -prob

The PCFG File

The PCFG file will consist of a probabilistic context-free grammar rules in CNF form. Therefore you can assume that each rule consists 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 *both* the non-probabilistic CKY parsing algorithm, as well as the probabilistic CKY parsing algorithm. For non-probabilistic CKY, your parser **should ignore** the probabilities in the PCFG and identify all possible parses for each sentence. For probabilistic CKY, your parser should use the PCFG to determine the most probable parse for each sentence.

Output Specifications

For each sentence in the input file, your program should print three things:

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 (-).

   When applying probabilistic CKY, you should also print the probability of each constituent in parentheses. Please print ALL 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 .

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 using non-probabilistic CKY:

```
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
```

And here 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)
```

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. 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.*

SUBMISSION INSTRUCTIONS

Please use CANVAS to submit a gzipped tarball file named "cky.tar.gz". (This is an archived file in "tar" format and then compressed with gzip. Instructions appear below if you're not familiar with tar files.) Your tarball should contain the following items:

1. The source code files for your program. Be sure to include all files that we will need to compile and run your program!

   REMINDER: your program **must** be written in Python or Java, and it **must** compile and run on the Linux-based CADE (lab1 or lab2) machines! We will not grade programs that cannot be run on the Linux-based CADE machines.

2. An executable *shell script* named `cky.sh` that contains the exact commands needed to compile and run your CKY program on the data files provided on CANVAS. We should be able to execute this file on the command line, and it will compile and run your code. For example, if your code is in Python and does not need to be compiled, then your script file might look like this:

   ```
   python cky.py pcfg.txt sentences.txt
   ```

   If your code is in Java, then your script file might look something like this:

   ```
   javac cky/*.java
   java cky/MainClass pcfg.txt sentences.txt
   ```

3. A `README.txt` file that includes the following information:

   - Which CADE machine you tested your program on
     (this info may be useful to us if we have trouble running your program)
   - Any known idiosyncracies, problems, or limitations of your program.

4. **Submit TWO trace files.** One file should be called **cky.trace** and contain the output of your program for non-probabilistic CKY using the sample input files provided on Canvas. The second file should be called **cky-prob.trace** and contain the output of your program for probabilistic CKY using the sample input files provided on Canvas.

HELPFUL HINTS

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

tar cvfz cky.tar.gz cky/

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

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

tar xvfz cky.tar.gz

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

For 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/

TRACE FILES: You can generate a trace file in (at least) 2 different ways: (1) print your program's output to standard output and then pipe it to a file (e.g., `cky pcfg.txt sentences.txt > cky.trace`), or (2) print your output to standard output and use the unix *script* command before running your program on the test files. The sequence of commands to use is:

```
script cky.trace
cky pcfg.txt sentences.txt
exit
```

This will save everything that is printed to standard output during the session to a file called `cky.trace`.