Assignment 7: More Dynamic Programming
CS 758/858, Fall 2024
Due **11:30pm on Wed, Oct 9**

Implementation

The skeleton code on the course web page is the start of a program for parsing sequences of tokens according to a context-free grammar. The code will hopefully eventually include a hopelessly inefficient parsing algorithm that tries all possible parses until finding one that matches. Implement a dynamic programming approach for parsing, and compare its performance to the provided enumeration approach. Those in 858 need to return all possible parses. Those in 758 will instead implement probabilistic parsing and merely return a most probable parse. In this context, the DP approach is called 'the Cocke-Kasami-Younger algorithm' or 'CKY.'

The skeleton code parses the input string into an array of tokens. Your output must include the parse in bracketed notation, so you will need to keep track of which non-terminals and spans are used in the parses you construct.

The input grammar (in Chomsky Normal Form) will be formatted in a manner such as:

```
S NP VP 1
NP ART N .5
NP N N 0.3
NP NP PP 0.2
VP V NP .4
VP VP INOBJ 0.3
VP NP PP 0.3
INOBJ NP NP 1
PP P NP 1
NP fish 1
N fish 1
ART the 1
V fish 1
VP fish 1
P about 1
```

The start symbol for the grammar is required to be `S`. All nonterminals are required to be in capital letters. All terminals are required to be in lower case. The number following each production is the probability associated with that production.

In the above grammar, the word "fish" can be used as either a noun or a verb, and as a result, there are quite a few valid parses for the sentence `fish fish fish fish fish`. Some selected examples include:

```
S(NP(fish) VP(VP(V(fish) NP(fish)) INOBJ(NP(fish) NP(fish)))) 0.120000
S(NP(N(fish) N(fish)) VP(V(fish) NP(N(fish) N(fish)))) 0.036000
```

The number is the probability associated with that particular parse.

Undergraduate students are to output only a single parse, one of the (possibly multiple) most probable ones, along with the associated probability. Graduate students are expected to output all valid parses of the sentence, and do not need to output any probability. We recommend getting the undergrad solution to work first, before adapting it into a grad solution.

Testing

`parser` The skeleton can be run as `./parser grammar.cnf <alg>` where `<alg>` may be either `rdp` (recursive descent parser) or `cyk`, which is the name of the dynamic programming parsing algorithm

Example:

```
echo the fish fish fish | ./parser-reference data/fish.cnf cyk
```

`harness.py` The harness for this assignment is a python script. Usage:

```
harness.py [-h] [--binary BIN] [--ref REF] [--sentence SEN] [--grammar GRAMMAR]
[--grad]
```

The arguments are:

-h, –help	show usage and exit
–binary BIN	binary to use
–ref REF	reference solution to use
–sentence SEN	sentence to try and parse
–grammar GRAMMAR	grammar to use
–grad	use undergraduate student mode

The harness checks the output from the student solution and the reference solution, and outputs any errors it finds. When it is done, it writes the word "Done". If this is the only output it produces, it found no problems, otherwise it will output information about any discrepancies as well.

Example:

```
python2 ./harness.py --binary ./parser-reference --sentence fish.sentence --grammar
data/fish.cnf
```

generating grammars There is a utility java program that can be used to convert context free grammars to CNF. It comes as a runnable jar, as well as a collection of java files.

It can be run as `java -jar cfg_convert.jar data/english.cfg`.

This will take the grammar in `data/english.cfg` and convert it to a grammar in Chomsky Normal Form. In the process of adding grammars, it will often be required to add its own symbols. It uses the prefix ADDED followed by a letter, so don't use nonterminals like that in the original grammar!

Written Problems

1. Briefly list any parts of your program which are not fully working. Include transcripts showing the successes or failures. Is there anything else that we should know when evaluating your implementation work?
2. What is the time complexity of your program? How does this compare to the enumeration approach?
3. (Those in 858 only) Problem 14–6 in CLRS. If you get totally stumped, one design technique is to start with exhaustive enumeration, then memoize, then convert to DP.
4. Exercise 15.1–2 in CLRS.
5. Exercise 15.1–3 in CLRS.
6. Exercise 15.2–1 in CLRS.
7. Exercise 15.2–4 in CLRS.
8. Part a of problem 15–2 in CLRS.
9. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit your work using the script on agate (eg, `~cs758/scripts/sub758 7-undergrad your-asn7-dir`).

Evaluation

In addition to correctness, your work will be evaluated on clarity and efficiency.
Tentative breakdown:

5–6 parsing

4–5 written problems