CS114 (Spring 2015) Homework 5 Part One
Statistical Parser: CKY Algorithm

Due March 20, 2015

Late submission of this part will receive at most half credit because we need to release the solution right after. You will need the perfect solution for this part to work on part 2 of the homework.

Introduction

The starter code consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download all the code from the github repo.

Files you will edit:

- `parser.py` The place where you'll put your code

Files you might want to look at:

- `grammar.py` The place where we put the grammar
- `tree.py` The tree data structure we'll use. (same as nltk tree)
- `util.py` A handful of useful utility functions.

Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

When you are done, submit your solution from 1.1 and your version of `parser.py` .

1 Probabilistic CKY algorithm

If you think you already understand CKY, do 1.2 first and then use your code to do 1.1. If you are still shaky, do 1.1 first and 1.2 will be quite straightforward.

1.1 By hand

You are given a small grammar. Your task is to write down all of the possible subtrees for 'time flies like an arrow.' CKY algorithm will help you do this efficiently, and it is really not painful to do by hand. The grammar can be obtained from the starter code.

```
>>> from grammar import timeFliesPCFG
>>> print str(timeFliesPCFG)
Noun => flies | 0.4
Noun => arrow | 0.4
Noun => time | 0.2
TOP => S | 1.0
Det => an | 1.0
VP => Verb PP | 0.2
VP => Verb NP_PP | 0.1
VP => Verb NP | 0.6
VP => Verb | 0.1
S => VP | 0.2
S => VP PP | 0.1
S => NP VP_PP | 0.2
S => NP VP | 0.5
VP_PP => VP PP | 1.0
NP_PP => NP PP | 1.0
Prep => like | 1.0
PP => Prep NP | 1.0
Verb => flies | 0.5
Verb => time | 0.3
Verb => like | 0.2
NP => Noun | 0.3
NP => Det Noun | 0.7
```

1. (Base case) Write down all of the possible subtrees and their probability that span over just one word. Don't forget the unary rules. In CKY terms, write down the subtrees and their probabilities in the cell (0,1), (1,2), (2,3), (3,4), and (4,5).
2. (Recursive case) Write down all of the possible subtrees and their probability that span over 2 words. In CKY terms, write down the subtrees and their probabilities in the cell (0,2), (1,3), (2,4), (3,5).

For example, for cell (0,2), you will find the rules that can combine subtrees in (0,1) with the subtrees in (1,2), which you have derived in the previous step. Then apply all of those rules to generate bigger subtrees.

3. (Recursive case) Write down all of the possible subtrees and their probability that span over 3 words. In CKY terms, write down the subtrees and their probabilities in the cell (0,3), (1,4), (2,5).

   For example, for cell (1,4), you will find the rules that can combine subtrees from (1, 2) and (2, 4) and the rules that can combine subtrees from (1, 3) and (3, 4). Then apply all of those rules to generate bigger subtrees.

4. (Recursive case) In CKY terms, write down the subtrees (spanning 4 words) and their probabilities in the cell (0,4), (1,5)

5. (Termination) Write down the trees (and their probabilities) that span the entire sentence i.e. apply rules that can combine (0,1) & (1,5), (0,2) & (2,5), (0,3) & (3,5), and (0,4) & (4,5). The complete tree must have `TOP` as the root.

1.2 By machine

Open up `parser.py` . We have given you an almost-complete implementation of CKY. It initializes the bottom cells of the chart, then applies unary rules there, and applies binary rules in the entire chart. (Hint: the solution only needs around 7 lines of code) All you have to do is apply unary rules in the recursive case. If you've correctly implemented this, and loaded all the relevant files, you should be able to run:

```
>>> print timeFliesSent
['time', 'flies', 'like', 'an', 'arrow']

>>> parse(timeFliesPCFG, timeFliesSent)
(TOP: (S: (NP: (Noun: 'time')) (VP: (Verb: 'flies')
              (PP: (Prep: 'like') (NP: (Det: 'an') (Noun: 'arrow'))))))
```

Note that you cannot get this output without correctly handling unary rules, because you'll never be able to get the `TOP=>S` at the top.