LING/CSCI 5832 Fall 2016 Homework 5 - PCFG parsing with the CKY algorithm

Due **Tue Nov 8** before class (**2pm**) on D2L.

In this homework, you will implement a CKY parser for probabilistic context-free grammars. You're given an already binarized grammar in Chomsky Normal Form, although with unary rules, and the task is only to implement the grid-filling part of the CKY algorithm. You do not need to retrieve a parse; rather, the only thing you need to do is to calculate the probability of most likely parse. If you implement CKY correctly, this will correspond to the probability associated with `S` in the top grid cell. See the textbook, the lecture slides, and the bottom of this page for the pseudocode of CKY. You don't need to use log-probabilities, although their use would be warranted if you wanted the ability to parse long sentences (>25 words). You need to also implement the unary-rule search component discussed in lecture for rules such as `('NP','N')` where the right-hand side is a non-terminal. That means you need to loop over all possible unary rules after filling in the possible binary rules in the CKY-grid, for each cell.

The grammar

The grammar you need to use is as follows (`S` is the start symbol):

```
grammar = {
    ('S','NP','VP'):0.9,
    ('S','VP'):0.1,
    ('VP','V','NP'):0.5,
    ('VP','V'):0.1,
    ('VP','V','@VP_V'):0.3,
    ('VP','V','PP'):0.1,
    ('@VP_V','NP','PP'):1.0,
    ('NP','NP','NP'):0.1,
    ('NP','NP','PP'):0.2,
    ('NP','N'):0.7,
    ('PP','P','NP'):1.0,
    ('N','people'):0.5,
    ('N','fish'):0.2,
    ('N','tanks'):0.2,
    ('N','rods'):0.1,
    ('V','people'):0.1,
    ('V','fish'):0.6,
```

```
    ('V','tanks'):0.3,
    ('P','with'):1.0
}
```

Note that this grammar has been binarized for you. For example, the rule pair `('VP','V','@VP_V')` and `('@VP_V','NP','PP')` was originally a single rule `VP → V NP PP`.

You should create a function `CKY` which can be called with a tokenized sentence as a list, and a grammar in the format given above.

The last seven lines of your homework should be as follows:

```
print CKY(['fish','people','fish','tanks'], grammar)
print CKY(['people','with','fish','rods','fish','people'], grammar)
print CKY(['fish','with','fish','fish'], grammar)
print CKY(['fish','with','tanks','people','fish'], grammar)
print CKY(['fish','people','with','tanks','fish','people','with','tanks'], grammar)
print CKY(['fish','fish','fish','fish','fish'], grammar)
print CKY(['rods','rods','rods','rods'], grammar)
```

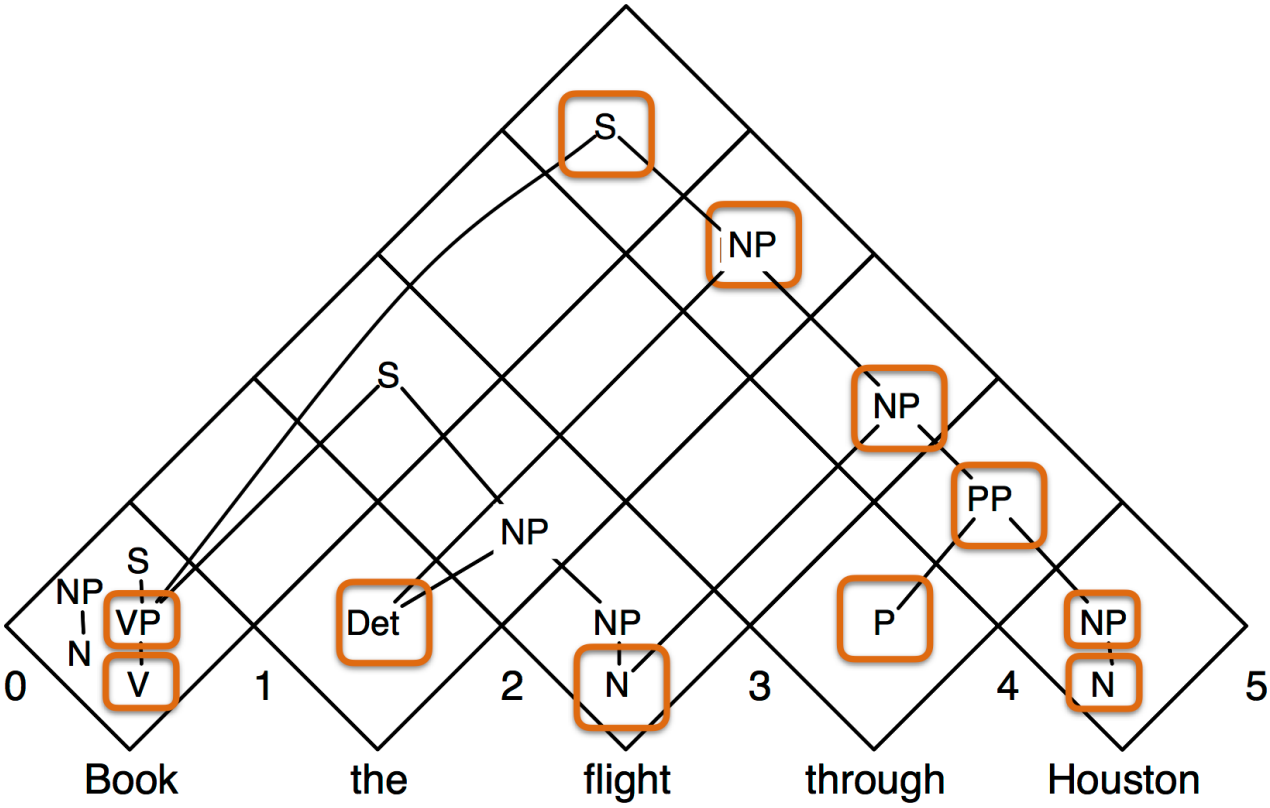
And the output should be:

```
0.00018522
6.4827e-06
0.00021168
2.4696e-05
1.0890936e-06
1.037232e-06
0.0
```

As in previous homeworks, the parameters that you pass to `CKY` do not need to be exactly the ones above. Perhaps you want to preprocess the grammar in some way, and send a list of terminals and nonterminals separately, or some such thing. That is fine, but you should at least pass the sentence to the `CKY` algorithm.

Extra credit for parse retrieval

You can get extra credit if you also retrieve the parse from the CKY-grid in some form. A normal way of doing this is to store backpointers for every rule used in the grid, then start with the `S`-element at the top, and recursively traverse the tree, outputting a left bracket and the node name, e.g. `[S` whenever you reach a new node, and a right bracket and the node name, e.g. `]S` whenever you pop a node.



For example, imagine you had the above CKY-grid filled in, with backpointers, and ignoring probabilities, you would output the parse (participating nodes shown highlighted) as:

```
[S [VP [V book ]V ]VP [NP [Det the ]Det [NP [N flight ]N [PP [P through ]P [NP [N H
```

What external resources can I use

You may use `numpy` for the grid. This is indeed recommended, since indexing and grid initialization is convenient. For example, to initialize a 3d array of dimensions `i,j,k` you just do this (assuming first having imported `numpy` by `import numpy as np`):

```
grid = np.zeros((i,j,k))
```

After that, you can access an index `i,j,k` by `grid[i,j,k]`.

What to hand in

You should hand in an archive `firstname.lastname.5832.hw5.tar.gz` or `firstname.lastname.5832.hw5.zip` consisting at least of a Python file `firstname.lastname.5832.hw5.py` that runs as decribed. Put a `README` in the archive if you

did anything outside a straightforward implementation of the above. If you're also retrieving the best parse, your `CKY` -function can return both the parse in some format (probably a string, as described above) and the probability.

CKY pseudocode with unary search

```
function CKY(words, grammar) returns most probable parse/prob score
for i=0; i<#(words); i++  // single word case
    for A in nonterms
        if A -> words[i] in grammar
            score[i][i+1][A] = P(A -> words[i])

    //handle unaries
    boolean added = true
    while added
        added = false
        for A, B in nonterms
            if score[i][i+1][B] > 0 && A->B in grammar
                prob = P(A->B)*score[i][i+1][B]
                if(prob > score[i][i+1][A])
                    score[i][i+1][A] = prob
                    added = true

for span = 2 to #(words)  // main CKY loop
    for begin = 0 to #(words)- span
        end = begin + span
        for split = begin+1 to end-1
            for A,B,C in nonterms
                prob = score[begin][split][B]*score[split][end][C]*P(A->BC)
                if(prob > score[begin][end][A])
                    score[begin][end][A] = prob

            //handle unaries
            boolean added = true
            while added
                added = false
                for A,B in nonterms
                    prob = P(A->B)*score[begin][end][B]
                    if(prob > score[begin][end][A])
                        score[begin][end][A] = prob
                        added = true
```