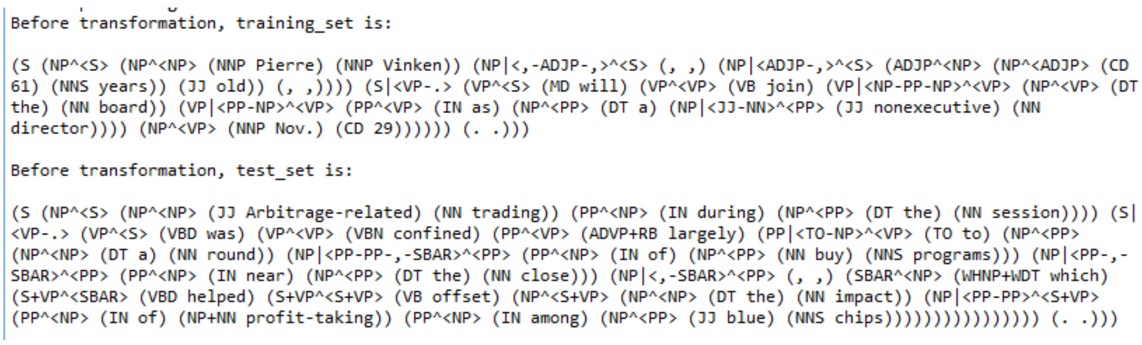
PROBLEM SET #4

MARIA MOYA

1. Preprocessing

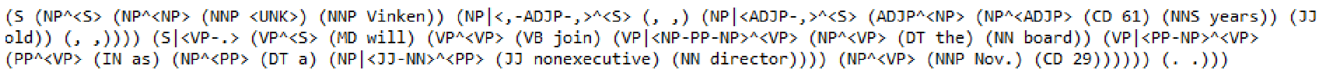
I first consider preprocessing the Penn treebank by removing any traces. Next, I define our vocabulary in line 63. We can take any CFG and tranform it into a CNF by recursively removing any empty and unirary rules as well as dividing up any rules that have more than 2 preterminals on the LHS. CNF's, maybe peculiar in structure however they allow for more efficient parsing. First we consider all functions defined in our template. Next wemodify the *TreebankNoTraces* function by considering a first order vertical markov *vertMarkov=1.* Thus we remove function tags, empties and uniaries. This yields the following output:



```
Before transformation, training_set is:

(S (NP^<S> (NP^<NP> (NNP Pierre) (NNP Vinken)) (NP|<,-ADJP-,>^<S> (, ,) (NP|<ADJP-,>^<S> (ADJP^<NP> (NP^<ADJP> (CD
61) (NNS years)) (JJ old)) (, ,)))) (S|<VP-.> (VP^<S> (MD will) (VP^<VP> (VB join) (VP|<NP-PP-NP>^<VP> (NP^<VP> (DT
the) (NN board)) (VP|<PP-NP>^<VP> (PP^<VP> (IN as) (NP^<PP> (DT a) (NP|<JJ-NN>^<PP> (JJ nonexecutive) (NN
director)))) (NP^<VP> (NNP Nov.) (CD 29)))))) (. .)))

Before transformation, test_set is:

(S (NP^<S> (NP^<NP> (JJ Arbitrage-related) (NN trading)) (PP^<NP> (IN during) (NP^<PP> (DT the) (NN session)))) (S|
<VP-.> (VP^<S> (VBD was) (VP^<VP> (VBN confined) (PP^<VP> (ADVP+RB largely) (PP|<TO-NP>^<VP> (TO to) (NP^<PP>
(NP^<NP> (DT a) (NN round)) (NP|<PP-PP-,-SBAR>^<PP> (PP^<NP> (IN of) (NP^<PP> (NN buy) (NNS programs))) (NP|<PP-,-
SBAR>^<PP> (PP^<NP> (IN near) (NP^<PP> (DT the) (NN close))) (NP|<,-SBAR>^<PP> (, ,) (SBAR^<NP> (WHNP+WDT which)
(S+VP^<SBAR> (VBD helped) (S+VP^<S+VP> (VB offset) (NP^<S+VP> (NP^<NP> (DT the) (NN impact)) (NP|<PP-PP>^<S+VP>
(PP^<NP> (IN of) (NP+NN profit-taking)) (PP^<NP> (IN among) (NP^<PP> (JJ blue) (NNS chips))))))))))))))) (. .)))
```

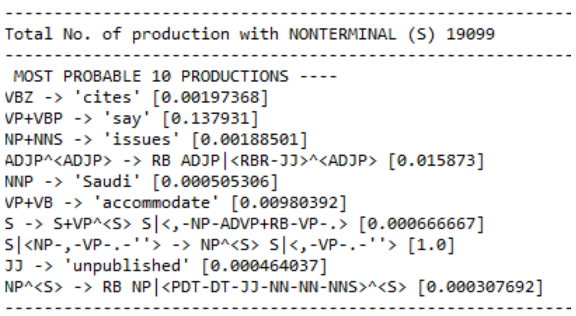
2. Training

Next, on lines 189-196 we consider learning our PCFG. I define my grammar to be the *induce_pcfg* function which considers my start nonterminal S and productions. After applying the transformation we obtain the following training set



```
(S (NP^<S> (NP^<NP> (NNP <UNK>) (NNP Vinken)) (NP|<,-ADJP-,>^<S> (, ,) (NP|<ADJP-,>^<S> (ADJP^<NP> (NP^<ADJP> (CD 61) (NNS years)) (JJ
old)) (, ,)))) (S|<VP-.> (VP^<S> (MD will) (VP^<VP> (VB join) (VP|<NP-PP-NP>^<VP> (NP^<VP> (DT the) (NN board)) (VP|<PP-NP>^<VP>
(PP^<VP> (IN as) (NP^<PP> (DT a) (NP|<JJ-NN>^<PP> (JJ nonexecutive) (NN director)))) (NP^<VP> (NNP Nov.) (CD 29)))))) (. .)))
```

Moreover we obtain the following output:



```
----------------------------------------------------------
Total No. of production with NONTERMINAL (S) 19099
----------------------------------------------------------
 MOST PROBABLE 10 PRODUCTIONS ----
VBZ -> 'cites' [0.00197368]
VP+VBP -> 'say' [0.137931]
NP+NNS -> 'issues' [0.00188501]
ADJP^<ADJP> -> RB ADJP|<RBR-JJ>^<ADJP> [0.015873]
NNP -> 'Saudi' [0.000505306]
VP+VB -> 'accommodate' [0.00980392]
S -> S+VP^<S> S|<,-NP-ADVP+RB-VP-.> [0.000666667]
S|<NP-,-VP-.-''> -> NP^<S> S|<,-VP-.-''> [1.0]
JJ -> 'unpublished' [0.000464037]
NP^<S> -> RB NP|<PDT-DT-JJ-NN-NN-NNS>^<S> [0.000307692]
----------------------------------------------------------
```

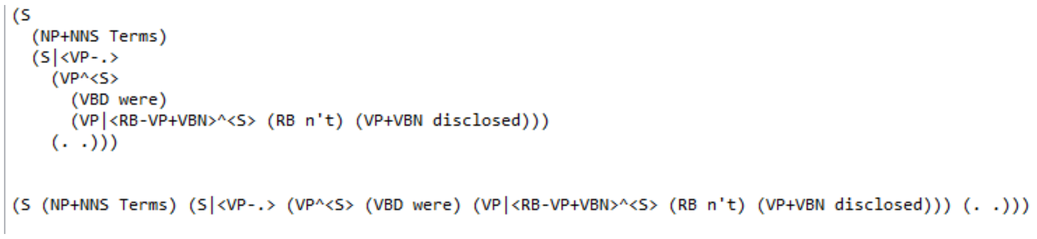
3. Testing

Now that we've learned our PCFG, next we consider the probabilistic CKY algorithm for parsing a test sentence. For this assignment, we will only consider nonzero probabilities and loop over the grammar rules associated with them. For PCF's, each rule has an associated probabilty. CKY is a cubic term algorithm through compact representation by using dynamic programming. CKY uses a parse triangle where the number entries that we have is order n^2 thus the algorithm itself runs in cubic time. CKY can handle uniaries and empty epsilons. Moreover, through Viterbi max scores we end up building in a binary fashion.

First I consider the template's *InvertedGrammar* function. Next I define a function labeled *BuildIndex* in lines 118-123 that will build an inverted index of the grammar that will map the RHS of all productions to their corresponding LHS. I've attached the index file to the zip file.

I implement the CKY algorithm by defining the *Parse* function on line 126. For this Parser, will consider the pseudo code found in slide 29. I consider 2 sub-spans, sub1 for span (i,k) and sub2 for span (k,j). Thus for each span and nonterminal, we're going to record the probability of a constituent of that type. We also consider the back pointers as way of knowing what was the best way of building every constituent over every span. I then take the log probabilities of the productions. Using the Inverted Grammar and our CKY Parser this gives us a log probability of -22.3343949387.

Next I construct a function called *BuildTree* on line 156. "This will build tree objects by following the back-pointers starting from the largest span (0, len(sent)) and recursing from larger spans (i, j) to smaller sub-spans (i, k), (k, j) and eventually bottoming out at the preterminal level (i, i+1)." The parse tree for the 5-token sentence Terms were nt disclosed" is. Using the inverted grammar and the Build Tree function I get:



```
(S
  (NP+NNS Terms)
  (S|<VP-.>
    (VP^<S>
      (VBD were)
      (VP|<RB-VP+VBN>^<S> (RB n't) (VP+VBN disclosed)))
    (. .)))


(S (NP+NNS Terms) (S|<VP-.> (VP^<S> (VBD were) (VP|<RB-VP+VBN>^<S> (RB n't) (VP+VBN disclosed))) (. .)))
```

Next, we will consider sentence buckets. The first bucket consist of sentences that are less than 10 words, the second bucket contains sentences less than 20 words and so on. Note that bucket 5 consist of sentences with more than 40 words(which isn't common in the English Language). This gives the following results:

Problem 3.3	
Bucket Number	Number of Sentences
Bucket 1	23
Bucket 2	134
Bucket 3	187
Bucket 4	86
Bucket 5	28

Lastly we want to generate our predicted trees and consider the gold standard where the gold standard consist of unbinarize trees. The table below list the Bracketing F Measure and Average crossing for each bucket. To no surprise our performances decreases as we increase the number of words per sentence.

Problem 3.4		
Bucket Number	Bracketing FMeasure	Average crossing
Bucket 1	86.70	0.05
Bucket 2	81.01	1.12
Bucket 3	72.02	2.98
Bucket 4	66.64	5.77
Bucket 5	62.27	8.54
Overall Test Set	71.27	3.17