**How to run the code:**

1. We have 2 input files
2. **pa3-cky/grammar\_rules.txt**

The grammar rules file has all the grammar rules as required. If you want to add more grammar rules, you can add new rules in next lines, with each term of the rule separated by a single ‘ ’ space character.

The program is generic enough, to handle new grammar rules, given grammar rules are formatted correctly and can be parsed by the program.

1. **pa3-cky/sents.txt**

The sents.txt file, contains all the input sentences. If you want to add new sentences or replace original, you can go ahead and put all the sentences in this file.

The program will read sentences from this file alone.

1. I have two python main code files.
2. **pa3-cky/sents.txt pa3-cky/parser/GrammarParser.py**

Grammar parser file has a class named GrammarParser, which is equipped with file I/O and parse features, and can extract relevant grammar rules and also the sentences from the input file.

1. **pa3-cky/parser/CKYAlgo.py**

CKYAlgo file, has a class named CKYAlgo, which uses GrammarParser class to parse the input file and run CKY Algorithm on it.

It also prints the results in the specified format on the terminal

To run simply type the following command. It will take all the input files by default.

*python CKYAlgo.py*

**Results**

**/usr/bin/python2.7 /home/avinsaxe/scratch/avinsaxe/TAMU/NLP/packyNLP/pa3-cky/parser/CKYAlgo.py**

**Rules are updated after parsing the file**

[['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'], ['']]

**Non-Terminals also updated**

set(['PP', '@VP\_V', 'N', 'VP', 'P', 'S', 'V', 'NP'])

Sentences are updated after parsing the file

[['fish', 'people', 'fish', 'tanks'], ['people', 'tanks', 'fish']]

**Sentence Under Consideration**

['fish', 'people', 'fish', 'tanks']

Span: fish

P( N->fish )= 0.2 P( VP->V )= 0.06 ( BackPointer = ( V ))

P( S->VP )= 0.006 ( BackPointer = ( VP ))

P( V->fish )= 0.6 P( NP->N )= 0.14 ( BackPointer = ( N ))

Span: people

P( N->people )= 0.5 P( VP->V )= 0.01 ( BackPointer = ( V ))

P( S->VP )= 0.001 ( BackPointer = ( VP ))

P( V->people )= 0.1 P( NP->N )= 0.35 ( BackPointer = ( N ))

Span: fish

P( N->fish )= 0.2 P( VP->V )= 0.06 ( BackPointer = ( V ))

P( S->VP )= 0.006 ( BackPointer = ( VP ))

P( V->fish )= 0.6 P( NP->N )= 0.14 ( BackPointer = ( N ))

Span: tanks

P( N->tanks )= 0.2 P( VP->V )= 0.03 ( BackPointer = ( V ))

P( S->VP )= 0.003 ( BackPointer = ( VP ))

P( V->tanks )= 0.3 P( NP->N )= 0.14 ( BackPointer = ( N ))

Span: fish people

P( VP->V , NP )= 0.105 ( BackPointer = ( (1, 'V', 'NP') ))

P( S->VP , NP )= 0.0189 ( BackPointer = ( (1, 'VP', 'NP') ))

P( NP->NP , NP )= 0.0049 ( BackPointer = ( (1, 'NP', 'NP') ))

Span: people fish

P( VP->NP , V )= 0.105 ( BackPointer = ( (2, 'NP', 'V') ))

P( S->NP , VP )= 0.0189 ( BackPointer = ( (2, 'NP', 'VP') ))

P( NP->NP , NP )= 0.0049 ( BackPointer = ( (2, 'NP', 'NP') ))

Span: fish tanks

P( VP->V , NP )= 0.042 ( BackPointer = ( (3, 'V', 'NP') ))

P( S->VP , NP )= 0.00756 ( BackPointer = ( (3, 'VP', 'NP') ))

P( NP->NP , NP )= 0.00196 ( BackPointer = ( (3, 'NP', 'NP') ))

Span: fish people fish

P( VP->V , NP )= 0.00147 ( BackPointer = ( (1, 'V', 'NP') ))

P( S->NP , VP )= 0.01323 ( BackPointer = ( (1, 'NP', 'VP') ))

P( NP->NP , NP )= 6.86e-05 ( BackPointer = ( (1, 'NP', 'NP') ))

Span: people fish tanks

P( VP->NP , V )= 0.000735 ( BackPointer = ( (3, 'NP', 'V') ))

P( S->NP , VP )= 0.01323 ( BackPointer = ( (2, 'NP', 'VP') ))

P( NP->NP , NP )= 6.86e-05 ( BackPointer = ( (3, 'NP', 'NP') ))

Span: fish people fish tanks

P( VP->V , NP )= 2.058e-05 ( BackPointer = ( (1, 'V', 'NP') ))

P( S->NP , VP )= 0.00018522 ( BackPointer = ( (2, 'NP', 'VP') ))

P( NP->NP , NP )= 9.604e-07 ( BackPointer = ( (1, 'NP', 'NP') ))

**Sentence Under Consideration**

**['people', 'tanks', 'fish']**

Span: people

P( N->people )= 0.5 P( VP->V )= 0.01 ( BackPointer = ( V ))

P( S->VP )= 0.001 ( BackPointer = ( VP ))

P( V->people )= 0.1 P( NP->N )= 0.35 ( BackPointer = ( N ))

Span: tanks

P( N->tanks )= 0.2 P( VP->V )= 0.03 ( BackPointer = ( V ))

P( S->VP )= 0.003 ( BackPointer = ( VP ))

P( V->tanks )= 0.3 P( NP->N )= 0.14 ( BackPointer = ( N ))

Span: fish

P( N->fish )= 0.2 P( VP->V )= 0.06 ( BackPointer = ( V ))

P( S->VP )= 0.006 ( BackPointer = ( VP ))

P( V->fish )= 0.6 P( NP->N )= 0.14 ( BackPointer = ( N ))

Span: people tanks

P( VP->V , NP )= 0.105 ( BackPointer = ( (1, 'V', 'NP') ))

P( S->VP , NP )= 0.0189 ( BackPointer = ( (1, 'VP', 'NP') ))

P( NP->NP , NP )= 0.0049 ( BackPointer = ( (1, 'NP', 'NP') ))

Span: tanks fish

P( VP->NP , V )= 0.105 ( BackPointer = ( (2, 'NP', 'V') ))

P( S->NP , VP )= 0.0189 ( BackPointer = ( (2, 'NP', 'VP') ))

P( NP->NP , NP )= 0.0049 ( BackPointer = ( (2, 'NP', 'NP') ))

P( VP->V , NP )= 0.042 ( BackPointer = ( (3, 'V', 'NP') ))

P( S->VP , NP )= 0.00756 ( BackPointer = ( (3, 'VP', 'NP') ))

P( NP->NP , NP )= 0.00196 ( BackPointer = ( (3, 'NP', 'NP') ))

Span: people tanks fish

P( VP->V , NP )= 0.00147 ( BackPointer = ( (1, 'V', 'NP') ))

P( S->NP , VP )= 0.033075 ( BackPointer = ( (1, 'NP', 'VP') ))

P( NP->NP , NP )= 0.0001715 ( BackPointer = ( (1, 'NP', 'NP') ))

P( VP->NP , V )= 0.000735 ( BackPointer = ( (3, 'NP', 'V') ))

P( S->NP , VP )= 0.01323 ( BackPointer = ( (2, 'NP', 'VP') ))

P( NP->NP , NP )= 6.86e-05 ( BackPointer = ( (3, 'NP', 'NP') ))

Process finished with exit code 0

**Analysis:**

1. We use a dynamic programming approach, instead of greedy approach, and compute and store all possible useful states

2. We compute all the possible states in parallel before finalizing a result.

3. The dynamic programming approach stores a 3 D table, and the values are reused throughout the cycle of computation

4. There can be instances of ambiguous parsing, where very close probabilities are reached for different possible solutions, and the ambiguity exists in the choice

5. I have used a dictionary based approach, for accessing all the relevant states of the dynamic programming approach in constant time. This has significantly improved the speed of computation

6. I have used a dictionary of list approach for storing all the rules after parsing from the rules file. This also improves searching of a particular rule for a sentence.

**Known Bugs, Limitations**

1. There is only one format in which input grammar rules and input sentences can be given. If the format of these rules and sentences is not correct, i.e. multiple sentences in one line, or grammar rules not ending with a \n character, then the program will fail