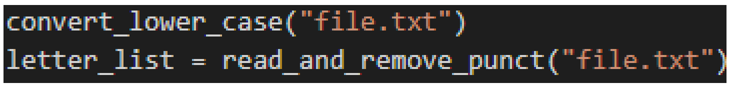
CSE 654- NATURAL LANGUAGE PROCESSING HW2 REPORT

In this homework we will develop a statistical language model of Turkish that will use N-grams of Turkish letters

1-)Filtering the Turkish Wikipedia Dump:

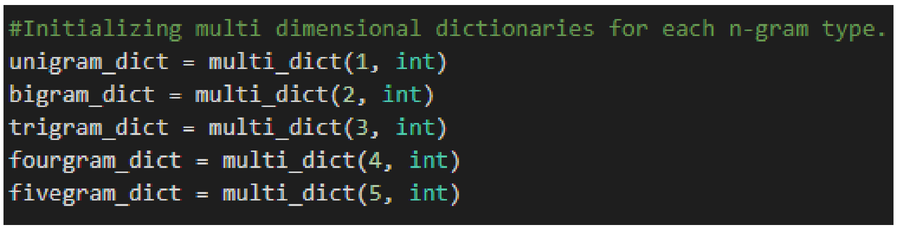
In order to start working with our set, we need to make some adjusments on that. First we need to convert all the letters to lower case letters. Then we need to get rid off punctuations. After doing these two simple operations our set will be ready for processing. These filtering operations has done with X and Y functions in my code.



```
convert_lower_case("file.txt")
letter_list = read_and_remove_punct("file.txt")
```

2-)Keeping the ngrams:

After reading and storing whole the letters in a file, now we can traverse the whole file to store ngrams. In order to accomplish that we can store our ngrams in a multi-dimensional dictionary. For unigrams we'll be using 1D dictionary, for bigrams we'll be using 2D dictionary and for trigrams 3D dictionary...



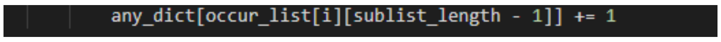
```
#Initializing multi dimensional dictionaries for each n-gram type.
unigram_dict = multi_dict(1, int)
bigram_dict = multi_dict(2, int)
trigram_dict = multi_dict(3, int)
fourgram_dict = multi_dict(4, int)
fivegram_dict = multi_dict(5, int)
```

3-)Filling the ngrams:

After deciding how to keep our ngrams now we can fill them. `fill_ngrams` function takes whole file as parameter and traverses each letter and adding those letter counts to each of our ngrams types in this manner:

4-)Keeping the frequencies of counts:

In order to keep frequencies we have to gather this information from our ngram dictionaries. `parse` function takes a dictionary and flattens it as a list. This list will contain values but for whole dictionaries there will be a lot of counts so we need to store this information in an another dictionary so we can reach desired count in O(1) time.



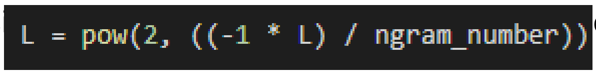
```
            any_dict[occur_list[i][sublist_length - 1]] += 1
```

5-)Calculating Perplexity:

`perplexity` function takes three parameters: sentence as our sentence, ngram_count as total letter count for a specific ngram type and ngram type for specific ngram type.

It applies the markov-chain formula. In this example perplexity function calculates the perplexity of an unigram:

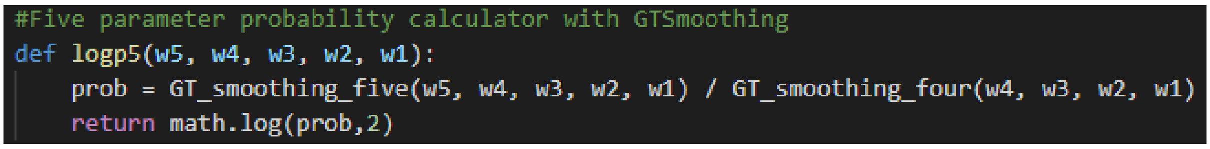
Since we're us exity) like this:



```
L = pow(2, ((-1 * L) / ngram_number))
```

6-)Calculating single probability:

In our perplexity function we've used functions like logp1(w1), logp2(w2,w1). This functions are returning log probability with the help of GT Smoothing function.



```python
#Five parameter probability calculator with GTSmoothing
def logp5(w5, w4, w3, w2, w1):
    prob = GT_smoothing_five(w5, w4, w3, w2, w1) / GT_smoothing_four(w4, w3, w2, w1)
    return math.log(prob,2)
```

7-)Calculating GT Smoothing:

In order to apply this Markov chain formula first we need to apply smoothing.

$$PP(s) = 2^{log_2^{PP(s)}} = 2^{-\frac{1}{n}log(p(s))}$$

let $l = \frac{1}{n}log(p(s))$

For unigram $l = \frac{1}{n}(logp(w_1) + \cdots + logp(w_n))$

For bigram $l = \frac{1}{n}(logp(w_1) + logp(w_2|w_1) + \cdots + logp(w_n|w_{n-1}))$

Here is GT Smoothing Formula:

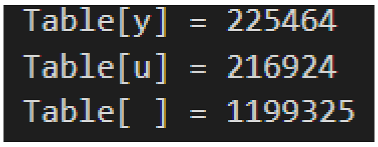
$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

I applied this one in my code but unfortunately I realised this formula won't work if we don't get the desired N count. That's why my perplexity values are inaccurate.

7-)Testing:

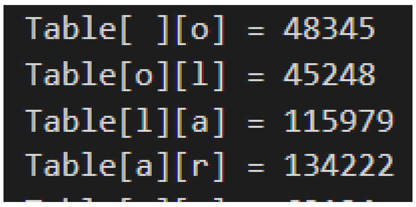
You can print the whole ngram tables. There is a function provided. So the output is like this

Unigram:



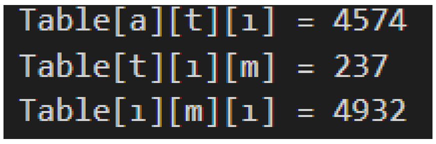
```
Table[y] = 225464
Table[u] = 216924
Table[ ] = 1199325
```

Bigram:



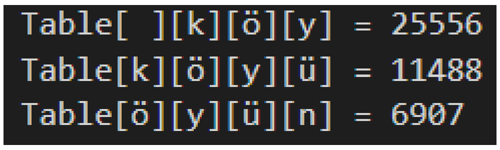
```
Table[ ][o] = 48345
Table[o][l] = 45248
Table[l][a] = 115979
Table[a][r] = 134222
```

Trigram:



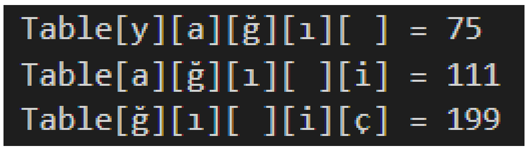
```
Table[a][t][ı] = 4574
Table[t][ı][m] = 237
Table[ı][m][ı] = 4932
```

Fourgram:



```
Table[ ][k][ö][y] = 25556
Table[k][ö][y][ü] = 11488
Table[ö][y][ü][n] = 6907
```

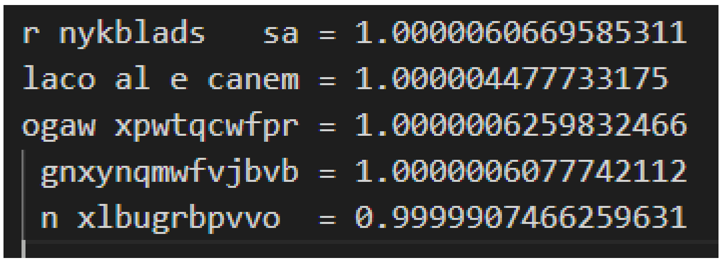
Fivegram:



```
Table[y][a][ğ][ı][ ] = 75
Table[a][ğ][ı][ ][i] = 111
Table[ğ][ı][ ][i][ç] = 199
```

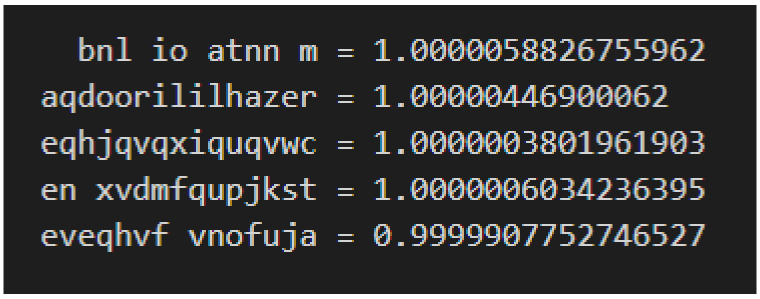
Created Random sentences:

First Test:



```
r nykblads   sa = 1.0000060669585311
laco al e canem = 1.000004477733175
ogaw xpwtqcwfpr = 1.0000006259832466
 gnxynqmwfvjbvb = 1.0000006077742112
 n xlbugrbpvvo  = 0.9999907466259631
```

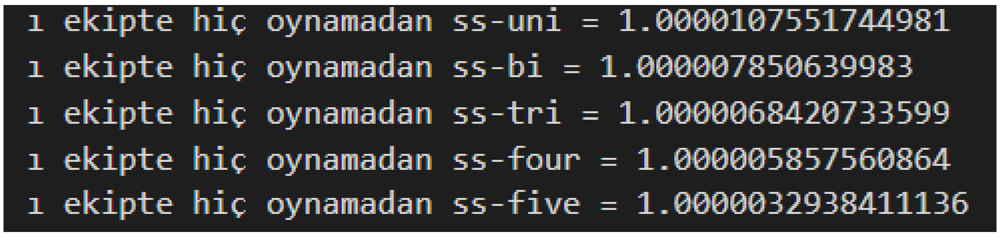
Second Test:



```
  bnl io atnn m = 1.0000058826755962
aqdoorililhazer = 1.000004469000062
eqhjqvqxiquqvwc = 1.0000003801961903
en xvdmfqupjkst = 1.0000006034236395
eveqhvf vnofuja = 0.9999907752746527
```

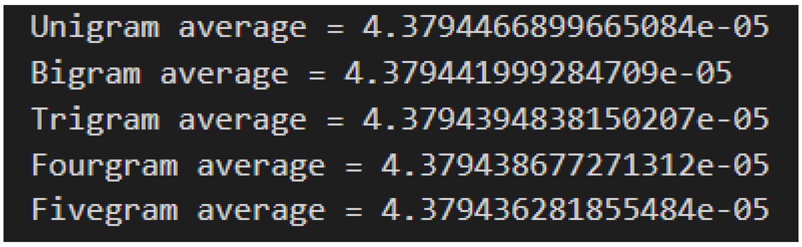
All ngrams and their frequenices:

After calculating whole sentences for all ngram types I printed the sentences and their perplexity values like this:



```
ı ekipte hiç oynamadan ss-uni = 1.0000107551744981
ı ekipte hiç oynamadan ss-bi = 1.000007850639983
ı ekipte hiç oynamadan ss-tri = 1.0000068420733599
ı ekipte hiç oynamadan ss-four = 1.000005857560864
ı ekipte hiç oynamadan ss-five = 1.0000032938411136
```

For 10 MB dataset and 700 KB test set I got the average values for each ngram perplexity like this:



```
Unigram average = 4.37944668996650084e-05
Bigram average = 4.379441999284709e-05
Trigram average = 4.3794394838150207e-05
Fourgram average = 4.379438677271312e-05
Fivegram average = 4.379436281855484e-05
```

So the table is like

UNIGRAM	4.37944668996650084e-05
BIGRAM	4.379441999284709e-05
TRIGRAM	4.3794394838150207e-05
FOURGRAM	4.379438677271312e-05
FIVEGRAM	4.379436281855484e-05

Conclusion:

- I’ve learned about Ngrams, Markov Chain rule, smoothing methods and perplexity.
- Since my GT smoothing implementation isn’t successful I couldn’t manage to compare result even though I’ve lower perplexity values for fivegrams. But it’s all of luck.