Group Members:

- John Le
- Chen Kai Zhang
- Jay Patel
- Brad Byun

# Problem 1: Programming n-gram language modeling

For our procedure, when taking the probability of the unigrams, we would take the total count of the words over all words. As for bigram, we would take the count of the the word occurring in succession after the previous word, over the count of the previous word. And for trigram, we did the same, except with the count of the word occurring in succesion after the two previous words, over the count of the two previous words. We stored all of these in a dictionary in their respective n-grams.

We then took the total summation of the log of the probability of the n-gram for each word in a sentence, multiplied by 1 over the total number of words in the corpus. Then finding perplety of the n-gram was having the log base to the power of the total.

**1. Report the perplexity scores of the unigram, bigram, and trigram language models for your training, development, and test sets. Briefly discuss the experimental results.**

```
 ===== Preprocessed Data =====
Unique Unigram Tokens: 26602
Unique Bigram Tokens: 615644
Unique Trigram Tokens: 1178733
Total Words: 1622905
===== Train Data =====
Unigram Perplexity: 983.4429002966516
Bigram Perplexity: 74.12469927377359
Trigram Perplexity: 5.197149726733204
===== Dev Data =====
Unigram Perplexity: 900.1971583364631
Bigram Perplexity: 19.99466904307459
Trigram Perplexity: 1.9943620259470918
===== Test Data =====
Unigram Perplexity: 904.3618500568698
Bigram Perplexity: 20.09882880437638
Trigram Perplexity: 1.0
```