II. Programming (70 pts)

Your task for this assignment is to build N-gram language models. Your language modeling program should accept the following command-line arguments:

```
python3 ngrams.py [training file] [test file]
```

For example, we should be able to run your program like this:

```
python3 ngrams.py train.txt test.txt
```

Given these arguments, your program should create language models from the training file and apply the language models to the sentences in the test file, as described below.

Input Files

The training file will consist of sentences, one sentence per line. For example, a training file might look like this:

```
I love natural language processing .
This assignment looks like fun !
```

You should divide each sentence into unigrams based solely on white space. Note that this can produce isolated punctuation marks (when white space separates a punctuation mark from adjacent words) as well as words with punctuation symbols that are still attached (when white space does not separate a punctuation mark from an adjacent word). For example, consider the following sentence (we show white space explicitly using ␣):

"This␣is␣a␣funny-looking␣sentence"␣,␣she said␣!

This sentence should be divided into exactly nine unigrams:

(1) "This (2) is (3) a (4) funny-looking (5) sentence" (6), (7) she (8) said (9)!

The test file will have exactly the same format as the training file and it should be divided into unigrams exactly the same way.

Building the Language Models

To create the language models, you will need to generate tables of frequency counts from the training corpus for unigrams (1-grams) and bigrams (2-grams). An N-gram should not cross sentence boundaries. All of your N-gram tables should be case-insensitive (i.e., "the", "The", and "THE" should be treated as the same word).

You should create three different types of language models:

(a) A unigram language model with no smoothing.

(b) A bigram language model with no smoothing.

(c) A bigram language model with add-one smoothing.

You can assume that the set of unigrams found in the training corpus is the entire universe of unigrams. We will not give you test sentences that contain unseen unigrams. So the vocabulary V for this assignment is the set of all unique unigrams that occur in the training corpus, including punctuation marks.

However, we will give you test sentences that contain bigrams that did not appear in the training corpus. The n-grams will consist entirely of unigrams that appeared in the training corpus, but there may be new (previously unseen) combinations of the unigrams. The first two language models (a) and (b) do not use smoothing, so unseen bigrams should be assigned a probability of zero. For the last language model (c), you should use add-one smoothing to compute the probabilities for all of the bigrams.

For bigrams, you will need to have a special pseudo-word "<s>" as a beginning-of-sentence symbol. Bigrams of the form "<s> w_i" mean that word w_i occurs at the beginning of the sentence. Do NOT include <s> as a word in your vocabulary for the unigram language model or include <s> in the sentence probability for the unigram model.

For simplicity, just use the unigram frequency count of w_{k-1} to compute the conditional probability $\mathrm{P}(w_k \mid w_{k-1})$. (This means you won't have to worry about cases where w_{k-1} occurs at the end of the sentence and isn't followed by anything.) For example, just compute $\mathrm{P}(w_k \mid w_{k-1}) = \mathrm{count}(w_{k-1}w_k) \,/\, \mathrm{count}(w_{k-1})$.

You should NOT use an end-of-sentence symbol. The last bigram for a sentence of length n should represent the last 2 words of the sentence: "$w_{n-1}w_n$".

For each of the language models, you should create a function that computes the probability of a sentence $\mathrm{P}(w_1 \ldots w_n)$ using that language model. Since the probabilities will get very small, you must do the probability computations in log space (as discussed in class, also see the lecture slides). Please do these calculations using log base 2.

Output Specifications

Your program should print the following information to standard output for each test sentence. When printing the logprob numbers, please print **exactly 4 digits** after the decimal point. For

example, print -8.9753864210 as -8.9754 . The programming language will have a mechanism for controlling the number of digits that are printed. If P(S) = 0, then the logarithm is not defined, so print "logprob(S) = undefined". Be sure to use a function that rounds, not truncates. For example, 1.33337 should be rounded to 1.3334 . You should only do the rounding as the **last step** in your calculations, just before printing. Do not round numbers during intermediate calculations or your final results will not be fully accurate. Print an empty line between different sentences.

Please print the following information, formatted exactly like this:

S = <sentence>
Unsmoothed Unigrams, logprob(S)= #
Unsmoothed Bigrams, logprob(S) = #
Smoothed Bigrams, logprob(S) = #

For example, your output might look like this (the example below is just for illustration):

```
S = Elvis has left the building .
Unsmoothed Unigrams, logprob(S)= -7.5025
Unsmoothed Bigrams, logprob(S) = undefined
Smoothed Bigrams, logprob(S) = -12.1393

S = Who ?
Unsmoothed Unigrams, logprob(S)= -2.9527
Unsmoothed Bigrams, logprob(S) = 0.0000
Smoothed Bigrams, logprob(S) = -1.2333
```

GRADING CRITERIA

1. Please use Canvas to submit **one** source code named "ngrams.py". We will run your program on the new files to evaluate the generality and correctness of your code. So please test your program thoroughly! Even if your program works perfectly on the examples that we give you, this does not guarantee that it will work perfectly on different test cases.

2. Please exactly follow the formatting instructions specified in this assignment and use the correct answer files "trace[#].txt" to make sure that your program conforms exactly to the input and output specifications. **You will get a 0 grade if you fail to follow the specifications.**

We provide a grading script "grading.sh" that will help you compare your output with the true answer trace file. The "grading.sh" should be put in the same directory as your "ngrams.py" and other example input files. Then run "bash grading.sh". If you are using Linux or Mac OS, it should all work smoothly. If you are using Windows, you need a bash shell first. For example, install Windows Subsystem for Linux (WSL)[1] and run the same commands.

3. You can **NOT** use any external software packages or dictionaries to complete this assignment except for **Python3.9 Standard Library**.[2] For example, libraries for general I/O handling, math

[1] `https://learn.microsoft.com/en-us/windows/wsl/install`
[2] `https://docs.python.org/3.9/library/index.html`

functions, and regular expression matching are ok to use. But you may not use libraries or code from any ngrams related software packages, or external code that performs any functions specifically related to NLP. All submitted code *must be your own*.