A. Diacritic Restoration. In this problem, you will use language modeling to restore missing accents in French words.

1. Ensure that you have current versions of OpenFST and OpenGRM installed. Elements of this assignment have been problematic on earlier versions. I recommend OpenFST 1.3.3 and OpenGrm Ngram 1.1.0.

2. Download the following files:
   http://cs.nyu.edu/~eugenew/asr13/data/train.txt
   http://cs.nyu.edu/~eugenew/asr13/data/test.txt
   http://cs.nyu.edu/~eugenew/asr13/data/dev-truth.txt
   http://cs.nyu.edu/~eugenew/asr13/data/dev.txt

   Note: the data should be sufficiently text-normalized to serve as input to the tools required in this assignment. However, feel free to normalize further as you see fit.

3. In this problem, we will be working with the data on the character level, not on the word level. Identify a list of characters occurring in the training data and construct a symbol table. This will be your vocabulary.

4. Train a 5-gram language model $G$ on the training data. Report the following statistics about your model:
   - The number of states.
   - The number of transitions.
   - The number of epsilon transitions.
   - The number of n-grams for all orders of n (include backoff n-grams).

5. Compute the perplexity of your language model on the character sequences found in dev-truth.txt.

6. Compute the OOV rate. Does the result surprise you?

7. Shrink the language model using a shrinking heuristic of your choice and give the statistics again.
8. Train a trigram language model and give the statistics.

9. Construct a transducer $T$ accepting character sequences with possibly missing
diacritics on the input, and transducing them to all possible diacritized versions of the
character sequence on the output. For example, given the input character sequence “ca”, the character sequences output by the transducer should include
- c a
- c à
- c â
- c ä
- ç a
- ç à
- ç â
- ç ä

The full transducer $T$ may be too big to draw. Draw a small version of $T$ with only a
subset of the transitions, and give a text listing of the full $T$.

10. Use $G$ (trigram, unshrunk) and $T$ to produce the most likely version of each word in file
dev.txt (hint: once again, composition is your friend).

11. Compare your output with the reference diacritized words found in dev-truth.txt. What is
your word error rate (percentage of words that are not diacritized completely correctly)?
What is your character error rate?

12. Now repeat steps 8 and 9 with the pruned 5-gram and trigram language model. How do
the results compare? Do the results surprise you?

In the speech recognition community, it’s common to attempt to improve the quality of an
algorithm or a system on a development data set, and then to report the final quality result on a
testing data set. Because the testing data set (and especially its corresponding ground truth) is
never seen during the development process, this ensures that one has not created a system
that has overfit the testing data. This keeps the experimentation fair.

13. We have provided the test data in test.txt, but not its associated ground truth. Submit the
output of your best system when applied to test.txt by email to the grader. The answer
must be in the same form as test.txt - one word per line. We will score your answer
against the ground truth. In addition to the output, please submit the transducer files and
the exact command sequence required to run your script.

14. Extra credit: try to improve the quality of your diacritics restoration system by looking on
the Internet for extra data sources or by improving your language model trained on the
data set provided. Use the error rates on dev.txt to guide you in this process, and submit
the best answer you can come up with for test.txt. The three submissions with the lowest
word error rate (WER) on the hidden ground truth test-truth.txt will receive extra credit
and a prize!

B. Changing numbers from “written” to “spoken” or “read” form.
1. Ensure that you have a current version of Thrax installed. I recommend 1.1.0.
2. Compile the grammar in src/grammars/numbers.grm and demonstrate by using the thrax tools with some sample numbers that the correct spoken-form strings are produced.
3. Modify numbers.grm so that instead of ordinal spoken-form numbers, the grammar produces cardinal ones. In addition, your grammar should handle any number between 1 and 9999. For example, if the input string is “1234”, the spoken-form numbers should be “one thousand two hundred thirty fourth”. In the process, remove all parts of the grammar not directly related to this task, such as reading four-digit strings as years and reading digit strings over four digits long digit-by-digit. Make sure you test your grammar carefully with plenty of examples, and submit a few examples. Also, submit your final numbers-card.grm file and the commands you used to compile and test it.
4. Modify your grammar further so that it handles numbers longer than four digits in the following way. Your grammar should split digit strings into four-digit chunks and read each chunk separately, and adjacent digit strings should be split by a period and a space. If there is an ambiguity about where to put the period, your grammar should prefer to use as few periods as possible. So the digit string 12345678 should produce an output like “one thousand two hundred thirty fourth. five thousand six hundred seventy eighth.” whereas the input 12345 should produce, with equal preference (you will need to get the tester tool to produce multiple alternatives):

   one thousand two hundred thirty fourth. fifth.
   twelfth. three hundred forty fifth.
   first. two thousand three hundred forty fifth.
   one hundred twenty third. forty fifth.

Submit your final numbers-card-punct.grm grammar in addition to the commands you used to test it, as well as a few test cases for your grammar. Test extensively to ensure correctness!