Computational Linguistics 1 - Fall 2011
CMSC/LING723, LBSC 744
Homework 1 - Due 20 Sept

Submit your responses and solutions to: `compling723.fall2011@gmail.com`
Homework write-ups may be in either plain text or .pdf format. Be sure include your full name, e-mail, *and any code* you write in your submission, and clearly indicate each problem number in your solution set.

There are two problems for this homework; Problem 1 is worth 10 points, and Problem 2 is worth 15 points.

Note that there are a lot of files for this homework; each is linked from this pdf file, but they can also all be found at:
`http://www.umiacs.umd.edu/~hollingk/classes/hw1.CL1-f11/`

Problem 1: Soundex (10 points)

The Soundex algorithm is a phonetic algorithm commonly used by libraries and the Census Bureau to represent peoples names as they are pronounced in English. It has the advantage that name variations with minor spelling differences will map to the same representation, as long as they have the same pronunciation in English. Here is how the algorithm works:

1. Keep the first letter of the name. This may be uppercased or lowercased.
2. Remove all non-initial occurrences of the following letters: `a, e, h, i, o, u, w, y`. (i.e., remove all occurrences of the given characters except when they occur in the first position.)
3. Replace the remaining letters (except the first) with numbers:
   - `/bfpv/1/` (replace b, f, p, or v with the number 1)
   - `/cgjkqsxz/2/`
   - `/dt/3/`
   - `/l/4/`
   - `/mn/5/`
   - `/r/6/`

   If two or more letters from the same number group were adjacent in the *original* name, then *only* replace the first of those letters with the corresponding number and ignore the others.
4. If there are more than 3 digits in the resulting output, then drop the extra ones.
5. If there are fewer than 3 digits, then pad at the end with the required number of trailing zeros.

The final output of applying the Soundex algorithm to any input string should be of the form `Letter Digit Digit Digit`. Table 1 shows the output of the Soundex algorithm for some example names.

Input	Output
Jurafsky	J612
Jarovski	J612
Resnik	R252
Reznick	R252
Euler	E460
Peterson	P362

Table 1: Example outputs for the Soundex algorithm.

See how the first two pairs of names, which are spelled differently but pronounced the same, are mapped to the same output by the Soundex algorithm?

Now, for the homework, **code an FST that implements the Soundex algorithm**. Note that it would be non-trivial to implement a single FST for the entire algorithm and therefore, the strategy we suggest you adopt is a bottom-up one: implement multiple transducers, each performing a simpler task, and then compose them together to get the final output. One possibility is to partition the algorithm across three transducers:

1. `Transducer 1`: Performs steps 1-3 of the algorithm, i.e, retaining the first letter, removing letters and replacing letters with numbers.
2. `Transducer 2`: Performs step 4 of the algorithm, i.e., truncating extra digits.
3. `Transducer 3`: Performs step 5 of the algorithm, i.e., padding with zeros as required.

Write code that takes a list of names on input, and uses a series of transducers (such as Transducers 1-3 suggested above) to output the Soundex representation of the name. Note that you *must* implement a composition of transducers in your code for this assignment, though you may choose to break down the problem in different ways and thus implement different transducers than those suggested above.[1]

Turn in your code, and its output given `names.txt` (provided on the course webpage) as input. How many unique Soundex outputs are there for our list of names?

Note that you may write your code in any language,[2] but be sure that we can run it—we will be testing it on new names to be sure it works.

[1] Yes, there is code to implement the Soundex algorithm available on the web. No, that code is probably not implementing transducers, and no, obviously you should not try to turn that code in as your own. You may, however, use the online implementations to verify your code; you can also verify your code against the names in Table 1.

[2] If you choose to write your code using the NLTK toolkit, then you might want to check out Jimmy Lin's tutorial on FSTs in NLTK. You may also choose to use the OpenFST library.

Problem 2: Pronunciation-Lexicon Dictionary (15 points)

Download the PronLex dictionary from the course webpage. Take a look at a few of the entries in this file: the first column represents full words, the second of the tab-delimited columns represents a pronunciation of that word using the ARPA phonetic alphabet, and the final column defines various functions of the word.

Look up the entry for "`almond`". Why are there 3 entries for this word? Find another word with multiple pronunciations.

Look up the entries for "`bare`" and "`bear`". What do you notice about the pronunciation entries? Find another pair of words with identical pronunciations and different spellings. Test these words in your Soundex code from Problem 1—does it predict identical pronunciations too?

Create an FST out of the PronLex dictionary, with words as input and phonetic pronunciation sequences as output on the transducer. You should be able to re-use the transducer data structure that you constructed for Problem 1 of this homework.[3] Note that for Problem 1, you probably defined all the arcs and states in your FSTs by hand; now you need to write code to do so automatically, from the PronLex dictionary file.

Now compose your PronLex FST with the sequence
"`the quick brown fox jumps over the lazy dog`"
(i.e., create an automata from the given sentence, such that "the" is the label on the transition arc from state 1 to 2, "quick" is the label from state 2 to 3, etc., then *compose* that automata with your PronLex FST). The result should be an FST, with our sentence on the input side of the transitions, and the phonetic pronunciation of our sentence on the output. Is the FST deterministic, or non-deterministic? (Actually, you can answer the non-determinism question without writing any code...) According to this dictionary, how would this sentence be pronounced, i.e., what is its phonetic sequence?

[3] If you chose to solve Problem 1 using OpenFST, then you may want to look at the `pronlex2fst.pl` script on the course webpage, which can be used along with the `fstcompose` function to create an OpenFST object.