LIN 353C: Introduction to Computational Linguistics, Spring 2016, Erk

Homework 2: Regular expressions and finite state automata

Due: Thursday Feb 18, 2016

This homework comes with a file, `lastname_firstname_hw2.py`. This is a (basically empty) file, called a *stub file*. Please record all the answers in the appropriate places of this file. Please make sure that you save the file in *plain text*, not something like the Word format.

For submission, please rename the file such that it reflects your name. So for example, if your name was "Alonzo Church", you should rename it to

```
church_alonzo_hw2.py
```

Also do not forget to put your name and EID in the second line of the file.

For the part of the homework that requires you to write Python code, we need to see the code, but not necessarily the output that the code produced (unless the problem explicitly asks for it). Please put your Python code into `lastname_firstname_hw2.py`. You can omit statements that produced an error or that did not form part of the eventual solution, but please include all the Python code that formed part of your solution. **Please submit this part of the homework electronically using Canvas.**

The part of the homework that is about finite state automata/-transducers can be done on paper or electronically. If you would like to do it electronically, submit it on Canvas. Otherwise, submit it on paper in class. If any of these instructions do not make sense to you, please get in touch with the instructor right away.

A perfect solution to this homework will be worth *100* points.

1. **Finding 'one-syllable words' in H.G. Wells' War of the Worlds (10 points)**

   For this problem, you will again analyze H.G. Wells' *The War of the Worlds*, but this time you will use the entire text, which is available for free on Project Gutenberg, `http://www.gutenberg.org/wiki/Main_Page`. Please download the text, and **make sure you get the plain text version, not the HTML version**.

   What you will do is see how many words in the War of the Worlds file are "one-syllable words".

   (a) For this problem, please use the heuristic that one-syllable words consist of zero or more consonants, followed by one or more vowels, followed by zero or more consonants (i.e., there is only one group of vowels in the word).

   Write a Python program that reads in the *War of the Worlds* from a file and then uses a regular expression to count the number of one-syllable words in it. You can split the text into words before you do the analysis, and you are also welcome to lowercase words before you apply a regular expression, to make the regular expression simpler. (There is a Python string method that does this.) Record the answer to the question (only the count, not the whole list of words, please!), as well as the Python program that you used to obtain it.

   (b) This description of one-syllable words is imperfect: There are words that are actually one-syllable words but that are not caught by the rule. There are also words that are not one-syllable words but that the rule reports as one-syllables.

   Give two examples of this: one word which is *not* a one-syllable word according to the heuristic but should be, and one which is a one-syllable word according to the heuristic but isn't. These two examples can be words that actually occur in the *War of the Worlds*, but they need not be.

2. **Regular expressions again: Searching in the War of the Worlds (20 points)**

   Please again read the *War of the Worlds* from a file. Split it into a list of lines.

   Please provide code that uses regular expressions to perform the following searches in the *War of the Worlds*. In your solution, provide *only the code*, not the results you got!

   (a) All lines that contain an integer or floating point number, for example:

   ```
   5
   3.1415
   -0.8
   .8
   ```

   (b) All integer or floating point numbers, extracted from the War of the World lines

   Make sure not to extract sequences like “1.”, so no final period without following digits.

   (c) All lines that contain a word form of the lemma “forget”. (At a minimum, cover inflectional morphology. If you want to add some derivational morphology, feel free to do so.)

   (d) All lines that contain words with exactly three consecutive vowels (that is, lines with four consecutive vowels should not be found unless they also contain a separate sequence of three consecutive vowels)

3. **Finite state automata. (30 points)**

   Draw finite state automata that recognize the following languages, where the alphabet is $\{a, b\}$. For this problem, please use **only deterministic automata**. That is, you cannot have two different transitions labeled with the same symbol going out of the same state.

   (a) $\{\ w \mid w$ contains at least three b's $\}$
   (e.g. *abbab* and *ababaababbb* should be accepted, but *aaabaaa* and *ab* should not)

   (b) $\{\ w \mid$ every odd position of w is a $b\ \}$
   (Your machine should also accept the empty string.)

   (c) $\{\ w \mid w$ contains the substring $bba\ \}$

   Extra credit (10 points): $\{\ w \mid w$ does not contain the substring $bba\ \}$

4. **Finite state automata, again (20 points)**

   Write a finite-state automaton for time-of-day expressions like

   - "eleven o'clock",
   - "twelve-thirty",
   - "midnight", or
   - "a quarter to ten".

   It should cover (at least) all the variants of those four expressions above that you get by substituting different numbers. Feel free to add others.

   Make the "alphabet" that this automaton uses English words, that is, you could for example have a single transition that says "eleven".

   This automaton can be nondeterministic, if you like.

   The finite-state automaton in Figure 2.15 of the Jurafsky and Martin book, which describes words for English numbers 1-99, may be useful for this problem. It's on page 31.

5. **A finite state transducer (20 points)**

In English, if a verb ends in ⟨vowel⟩*c*, add a -k before suffixes -ed and -ing (but not before -s). So from "panic" you go to "panicked", not "paniced". And you get "panics" and not "panicks".

Write a finite state transducer that takes an intermediate-level representation, either

panic^ed#

or

panic^ing#

or "panic" followed by anything else, and maps it to a surface-level representation. In the first case, it should produce "panicked", in the second case "panicking", and if it is "panic" followed by anything else, it should be left unchanged (except that ^ should be removed). The "^" stands for a morpheme boundary, and the "#" stands for the end of the string.

Your transducer can be nondeterministic.

For an example of a finite-state transducer that goes from intermediate to surface level (though for a different rule), look at Figure 3.17 in the Jurafsky and Martin book, page 64.

As in the transducer in Figure 3.17, you can leave the "#" symbol undeleted, if you like.