CS 1671: Human Language Technologies
Assignment 2

The first part of this assignment will give you experience with transducers and morphological processing, and the second will give you experience with N-grams and probabilities that are used in word prediction.

Please hand in a hard copy of your solution either in class or in my mailbox in the 5th Floor SS mail room.

For the first two questions, Use "other" the same way it is used in Figure 3.17 (though it will stand for different letters in your machines). Also, where appropriate, use "cons" to stand for any consonant and "vowel" to stand for any vowel.

Here, '$' is used in place of the caret symbol used in the book (it prints out more clearly).

1. (K insertion)

   (a) Write a transducer for the K-insertion spelling rule in English. For example, the surface form of "panic" + "ed" is "panicked".

   Similarly to the transducer in Figure 3.17:

   - All strings should be accepted
   - If the end of the string consists of a morpheme ending in 'c' followed by '&ed#', then the output of the transducer should be the input with the appropriate 'k' inserted. Also, all '$' in the input string should be removed from the output string.
   - Otherwise, the output should be the same as the input, but with all '$' removed.

   (b) Show a sequence of states your machine goes through to accept the following inputs. Please draw these as in class, with the states in a chain and the input symbols above the arrows and the output symbols below the arrows. (If something isn't working right, go back and fix your machine.)

   ```
   panic$ed#
   dog$s#
   panic$er$s#
   ```

   Also, for the first one, show all the other possible paths through the machine than the one you show above. If there are none, state that. Make sure your path does not accept "panic$ed#" while outputting "paniced#"!

   (c) Now write a transducer that maps between the lexical and intermediate levels. It should cover the following mappings. As in the Penn Treebank POS Tags, "D" is for preterite (past tense) (e.g., 'walked') and "P" is for non-3sg pres (please look these up in the table of Treebank tags in the front of the book):

```
panic +VB +D --> panic$ed#
panic +VB +P --> panic#
dog +N +Sg --> dog#
dog +N +Pl --> dog$s#
```

(d) Now use your cascade of transducers in the reverse order on the input "panicked". First show the mapping to the intermediate tape, then show the mapping from the intermediate to the lexical tape. As above, show a sequence of states each machine goes through to accept its input, showing the input and output (make sure I know which is which).

2. Recall that we are assuming that all the spelling rule transducers are run in parallel to each other. If instead they were cascaded, where the output of the first spelling rule is the input to the second, the output of the second is the input to the third, and so on, what would be wrong with the transducer in Figure 3.17 (and with yours too)?

3. (Consonant-doubling) Now we'll do the same things as above, but for consonant doubling. Note that there are exceptions; for example, the surface form of "sing" + "ing" is "singing", not "singging". Referring to Figures 3.14 and 3.17, figure out how/where you should handle such exceptions.

   (a) Write a transducer for the consonant-doubling spelling rule in English. For example, the surface form of "beg" + "ing" is "begging".

   Like the transducer in Figure 3.17:

   - All strings should be accepted
   - If the end of the string consists of a morpheme ending in a consent followed by '$ing#', then the output of the transducer should be the input with the appropriate consonant doubled. Also, all '$' in the input string should be removed from the output string.
   - Otherwise, the output should be the same as the input, but with all '$' removed.

   (b) Show a sequence of states your machine goes through to accept the following inputs. Please draw these as in class, with the states in a chain and the input symbols above the arrows and the output symbols below the arrows. (If something isn't working right, go back and fix your machine.)

   ```
   beg$ing#
   in$eligibility$s#
   un$see$ing#
   ```

   Also, for the first one, show all the other possible paths through the machine than the one you show above. If there are none, state that. Make sure your path does not accept "beg$ing# while outputting "beging#"!

(c) Now write a transducer that maps between the lexical and intermediate levels. As in the Penn Treebank POS Tags, "G" is for gerund (e.g., "seeing").

Your transducer should cover the lexical inputs "beg +V +G", "beg +V +P", "sing +V +G", "sing +V +P" (for this question, you need to determine what the output should be).

(d) Now use your cascade of transducers in the reverse order on the inputs "begging" and "singing". First show the mapping to the intermediate tape, then show the mapping from the intermediate to the lexical tape. As above, show a sequence of states each machine goes through to accept its input, showing the input and output (make sure I know which is which).

(e) In our machines, 'cons', 'vowel', 'other', and (in Figure 3.14) $s# are shorthands, to reduce busywork and to keep the figures understandable. In an actual (formal) Finite State Transducer, how would they need to be handled? (Remember the point made in lecture that regular-expression facilities in programming languages are actually more powerful than (formal) regular expressions. Think about why this point is relevant to this question.) Don't actually draw the modifications; just describe what type(s) of modifications would be needed. Be careful – it might not be the same answer for all four of them.

4. Write out the equation for trigram probability estimation (modifying Eq. 4.14).

5. The training corpus for this question is in the file assign2trainingdata.txt on the schedule (it is output of a program that generates random sentences according to a grammar).

(a) Create the (one-row) table of unigram counts, and the table of bigram counts as in Figure 4.1. If you are manually entering the information, just leave the 0 entries blank.

We'll work with one trigram, 'a red table' Give the count for this trigram.

(b) Create the table of bigram probabilities as in Figure 4.2. If you are manually entering them, give the table entries as fractions (don't bother dividing them out to get the probabilities).

Also, give the trigram probability for the trigram 'a red table'

(c) For these bigrams: 'a red' 'thinks that' 'man the' 'happy man', give (1) the smoothed bigram counts (see Figure 4.1), (2) the smoothed bigram probabilities (see Figure 4.6), and (3) the reconstructed counts (see Figure 4.7).