Allison Sliter, homework 6 Sequence Analysis, Fall 2014

Part 1
The file associated with part one is trie.py. If you run that script from the folder you unzipped everything from, you should be able to see my nifty little interactive dictionary look up.

```
"Loading Trie, please wait...
Would you like to provide your own dictionary or use mine? Type 1 for mine, Type 2 to provide
your own 1
Please enter the prefix you're looking up: yu ['yuan', 'yuca', 'yucca', 'yuck', 'yuckel', 'yucker',
'yuckle', 'yucky', 'yuft', 'yugada', 'yuh', >'yukkel', 'yulan', 'yule', 'yuleblock', 'yuletide', 'yummy',
'yungan', 'yurt', 'yurta', 'yus', >'yusdrum', 'yutu', 'yuzlik', 'yuzluk'] Matches 25
Type q to quit, any other key to start over w "
```

This file produces the part1.fst Although I don't actually use it later - it's just easier to produce another trie for dictionary.

Part 2
This will take a little bit of explanation.
I chose to use compose to traverse the two fsts, however I could not get pyfst to determinize the fsts successfully, so produced the levenshtein automata with the file leventrie.py and then used openfst for the other operations. The two fsts it produces are dict.fst and hand.fst. Because in part 3 I did not change my dictionary producing code, it's commented it out and I just reused the ones I make here.

```
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ python leventrie.py
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ fstdeterminize hand.fst det_hand.fst
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ fstdeterminize dict.fst det_dict.fst
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ fstcompose det_hand.fst det_dict.fst
hand2edit.fst
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ fstinfo hand2edit.fst
fst type                                    vector
arc type                                     standard
input symbol table                           none
output symbol table                          none
\# of states                                293
\# of arcs                                  343
initial state                               0
\# of final states                            53
\# of input/output epsilons                    0
\# of input epsilons                         62
\# of output epsilons                         239
input label multiplicity                    1
output label multiplicity                    4.11953
\# of accessible states                      293
\# of coaccessible states                      293
\# of connected states                       293
\# of connected components                       1
\# of strongly conn components                   293
input matcher                              y
output matcher                              n
```

input lookahead	n
output lookahead	n
expanded	y
mutable	y
error	n
acceptor	n
input deterministic	y
output deterministic	n
input/output epsilons	n
input epsilons	y
output epsilons	y
input label sorted	y
output label sorted	n
weighted	n
cyclic	n
cyclic at initial state	n
top sorted	n
accessible	y
coaccessible	y
string	n

The string output is saved to the file list_of_words_2_editdst_from_hands

Part 3
The code can be found in weightedleventrie.py. The two weighted schemes I developed were based on keyboard key distances and letter frequency based on how frequently those letters show up in the English language.
Letter frequencies:
I used the chart here (http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html) to develop my table. I then used the inverse of that value (so the most frequent letter gets the lowest weight). That version is implemented in the "other_levenshtein" function.
Keyboard distance:
I implemented this basically by treating the qwerty keyboard as a three by ten grid, throwing in a little punctuation for balance. I then looked at each key and said "Okay, how many spots are you above, how many to the left or right" and added those two values, and then took a somewhat arbitrary divisor of the resulting sum. It's kind of like an average, but I chose the divisor specifically so that the average distance would be almost exactly 1. This is the method of weighting implemented in the levenshtein function.
As before, I did the determinizing and composing in openfst. The resulting files followed:
Non-determinized:
freqeightedbear.fst
freqweightedhands.fst
keyweightedbear.fst
keyweightedhands.fst
Determinized versions of the above:
det_freq_bear.fst
det_freq_hand.fst
det_key_bear.fst

det_key_hand.fst
Composed fsts with dictionary:
freq_bear_2edit.fst
freq_hand_2edit.fst
key_bear_2edit.fst
key_hand_2edit.fst
Textfiles containing word lists:
list_of_words_2ed_from_bearclaw_weighted_by_ltr_freq.txt
list_of_words_2ed_from_bearclaw_weighted_by_ltr_key_dist.txt
list_of_words_at_2edit_from_hands_weighted_by_ltr_frequency.txt
list_of_words_at_2edit_from_hands_weighted_by_ltr_keyboard_distance.txt

```
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ fstcompose det_freq_bear.fst det_dict.fst
freq_bear_2edit.fst
Allisons-MacBook-Air:sliterhomework6 allisonsliter$ fstinfo freq_bear_2edit.fst
fst type                                          vector
arc type                                          standard
input symbol table                                none
output symbol table                               none
# of states                                       7
# of arcs                                         6
initial state                                     0
# of final states                                 1
# of input/output epsilons                        0
# of input epsilons                               0
# of output epsilons                              5
input label multiplicity                          1
output label multiplicity                         1
# of accessible states                            7
# of coaccessible states                          7
# of connected states                             7
# of connected components                         1
# of strongly conn components                     7
input matcher                                     y
output matcher                                    y
input lookahead                                   n
output lookahead                                  n
expanded                                          y
mutable                                           y
error                                             n
acceptor                                          n
input deterministic                               y
output deterministic                              y
input/output epsilons                             n
input epsilons                                    n
output epsilons                                   y
input label sorted                                y
output label sorted                               y
weighted                                          y
```

cyclic	n
cyclic at initial state	n
top sorted	y
accessible	y
coaccessible	y
string	y