Outline of Serial Algorithm for Word Correction

Reference/Credits:
SymSpell (Wolf Garbe)
blog.faroo.com/2012/06/07/improved-edit-distance-based-spelling-correction/
Python Damerau-Levenshtein distance implementation (Michael Homer)
http://mwh.geek.nz/2009/04/26/python-damerau-levenshtein-distance/

---

1. CREATE DICTIONARY ("CORPUS")

Note: The Corpus contains assumed "good" words, occurring at a frequency representative of how often they appear in normal usage

Input: .txt file
Output: dict of: words in corpus and derived words with 1..*max_d* deleted characters

```
for each line in file:
        parse line to get words
        for each WORD in line:
                <add WORD to dictionary if not yet in dictionary>
        print total words from corpus added to dictionary, and total dictionary terms
```

1a. <add WORD to dictionary if not yet in dictionary>

```
if WORD is already in dictionary:
        retrieve dictionary[WORD] = ([suggested_corrections], freq_in_corpus)
        increment freq_in_corpus   # count times WORD is in corpus (it's a real word)
else:
        create dictionary[WORD] = ([], 1)  # initialize
        adjust counter of longest dictionary word if necessary

if WORD's freq_in_corpus is 1 :
        # this is WORD's first appearance in corpus
        # (n.b. word may already be in dictionary as a derived word but
        #  freq_in_corpus is not incremented in those cases)
        flag word being added to dictionary
        <obtain list of all derivations of the WORD ("deletes")>
        for each "delete":
                if "delete" already in dictionary:
                        add WORD to "delete's" suggestion list if not already in that list
                else:
                        create dictionary["delete"] = ([], 0)  # note freq_in_corpus=0

return flag (default=False)
```

1b. **<obtain list of all derivations of the WORD ("deletes")>**

```
initialize "deletes" list
initialize queue with WORD

for d_index in max_d:  # number of deleted characters
        initialize temp_queue
        for each item in queue:
                if item has more than one character:
                        for each character in item:
                                remove ith character
                                add word to "deletes" list if not in there
                                add word to temp_queue if not in there

        reset queue with temp_queue

return "deletes" list
```

2. PROCESS STRING-TO-CORRECT

Input: STRING to correct
Output: list of correction suggestions within *max_d* distance
(sorted by shortest distance and greatest frequency)

```
return empty list if STRING is longer than longest word in dictionary by max_d
initialize suggestions list
initialize s_dictionary of items added to suggestions list
initialize queue with STRING
initialize q_dictionary of items added to queue

while queue is not empty:
        get q_item from queue
        remove q_item from queue
        if q_item is already in dictionary:
                retrieve dictionary[q_item]
                if q_item's freq_in_corpus>0 AND q_item not already in s_dictionary:
                        # q_item is a real word, so add it to suggestions list
                        mark q_item as added in s_dictionary
                        calculate distance of q_item from STRING
                        add q_item to suggestions list: (q_item, freq_in_corpus, distance)

                        for each sc_item of q_item's [suggested_corrections]:
                                #each sc_item also a valid longer correction possibility
                                if sc_item not already in s_dictionary:
                                        mark sc_item as added in s_dictionary
                                        <calculate distance of sc_item from STRING>
                                        if distance <= max_d:
                                                if sc_item already in dictionary:
                                                        add sc_item to suggestions list
                                                        (sc_item, freq_in_corpus, distance)

        # now generate deletes from queue items (substrings of STRING)
        # as additional strings to check
        if length difference between STRING and q_item is less than max_d:
                if length of q_item is greater than 1:  # added this
                        for each character in q_item:
                                remove ith character
                                add word to queue if not marked in q_dictionary
                                mark as added to queue in q_dictionary

# queue is now empty
sort suggestions list (ascending order of edit distance, decreasing word frequency)
return suggestions list for output
```

2a. **<calculate distance of sc_item from STRING>**

```
initialize distance = 0
if sc_item is not the STRING:  # this may always be true? (assert?)
        if sc_item and q_item are the same length
                distance = STRING length – q_item length
        else if STRING and q_item are the same length
                distance = sc_item length - q_item length
        else
                # optionally, trim common prefixes and suffixes
                # between STRING and sc_item for optimization
                get <Damerau-Levenshtein Distance> between STRING, sc_item
```