Daily Coding Problem #11

Problem

This problem was asked by Twitter.

Implement an autocomplete system. That is, given a query string `s` and a set of all possible query strings, return all strings in the set that have `s` as a prefix.

For example, given the query string `de` and the set of strings [`dog`, `deer`, `deal`], return [`deer`, `deal`].

Hint: Try preprocessing the dictionary into a more efficient data structure to speed up queries.

Solution

The naive solution here is very straightforward: we need to only iterate over the dictionary and check if each word starts with our prefix. If it does, then add it to our set of results and then return it once we're done.

```
WORDS = ['foo', 'bar', ...]
def autocomplete(s):
    results = set()
    for word in WORDS:
        if word.startswith(s):
            results.add(word)
    return results
```

This runs in O(N) time, where N is the number of words in the dictionary. Let's think about making this more efficient. We can preprocess the words, but what data structure would be best for our problem?

If we pre-sort the list, we could use binary search to find the first word that includes our prefix and then the last, and return everything in between.

Alternatively, we could use a tree for this. Not a binary tree, but a tree where each child represents one character of the alphabet. For example, let's say we had the words 'a' and 'dog' in our dictionary. Then the tree would look like this:

```
  x
 / \
a   d
     \
      o
       \
        g
```

Then, to find all words beginning with 'do', we could start at the root, go into the 'd' child, and then the 'o', child, and gather up all the words under there. We would also some sort of terminal value to mark whether or not 'do' is actually a word in our dictionary or not. This data structure is known as a trie.

So the idea is to preprocess the dictionary into this tree, and then when we search for a prefix, go into the trie and get all the words under that prefix node and return those. While the worst-case runtime would still be O(n) if all the search results

have that prefix, if the words are uniformly distributed across the alphabet, it should be much faster on average since we no longer have to evaluate words that don't start with our prefix.

```
ENDS_HERE = '__ENDS_HERE'

class Trie(object):
    def __init__(self):
        self._trie = {}

    def insert(self, text):
        trie = self._trie
        for char in text:
            if char not in trie:
                trie[char] = {}
            trie = trie[char]
        trie[ENDS_HERE] = True

    def elements(self, prefix):
        d = self._trie
        for char in prefix:
            if char in d:
                d = d[char]
            else:
                return []
        return self._elements(d)

    def _elements(self, d):
        result = []
        for c, v in d.items():
            if c == ENDS_HERE:
                subresult = ['']
```

```
            else:
                subresult = [c + s for s in self._elements(v)]
            result.extend(subresult)
        return result

trie = Trie()
for word in words:
    trie.insert(word)

def autocomplete(s):
    suffixes = trie.elements(s)
    return [s + w for w in suffixes]
```