Daily Coding Problem #259

Problem

This problem was asked by Two Sigma.

Ghost is a two-person word game where players alternate appending letters to a word. The first person who spells out a word, or creates a prefix for which there is no possible continuation, loses. Here is a sample game:

- Player 1: `g`
- Player 2: `h`
- Player 1: `o`
- Player 2: `s`
- Player 1: `t` [loses]

Given a dictionary of words, determine the letters the first player should start with, such that with optimal play they cannot lose.

For example, if the dictionary is `["cat", "calf", "dog", "bear"]`, the only winning start letter would be `b`.

Solution

This is a case where using the right data structure gets you most of the way to a solution.

For any prefix, we want to be able to efficiently find out what words can be created later on in the game. So a trie will be ideal. We can build this as follows:

```
class Trie:
    def __init__(self, words=[]):
        self.trie = {}
        for word in words:
            self.add(word)

    def add(self, word):
        root = self.trie
        for letter in word:
            if letter in root:
                root = root[letter]
            else:
                root = root.setdefault(letter, {})
        root['#'] = '#'

    def get(self, prefix):
        root = self.trie
        for letter in prefix:
            if letter in root:
                root = root[letter]
            else:
                return None
        return root
```

When we initialize this trie with the words above, the resulting dictionary will look like this:

```
{'d':
    {'o':
        {'g': {'#': '#'}}},
 'b':
    {'e':
        {'a':
            {'r': {'#': '#'}}}},
 'c':
    {'o':
        {'a':
            {'t': {'#': '#'}}},
     'a':
```

```
        {'t': {'#': '#'}}}}
```

Next, we must figure out what the winning prefixes are. Here, we can work backwards. Any prefix which is itself a word is clearly a losing prefix. Going one step up, if every next letter that can be added to a given prefix creates a loss, then this must be a winning prefix. For example, `do` is losing, since the only continuation is `g`, which creates a word.

Therefore, we can recursively determine the optimal starting letters by figuring out which ones have only losing children.

```
def is_winning(trie, prefix):
    root = trie.get(prefix)

    if '#' in root:
        return False
    else:
        next_moves = [prefix + letter for letter in root]
        if any(is_winning(trie, move) for move in next_moves):
            return False
        else:
            return True

def optimal_starting_letters(words):
    trie = Trie(words)
    winners = []

    starts = trie.trie.keys()
    for letter in starts:
        if is_winning(trie, letter):
            winners.append(letter)

    return winners
```

Constructing the trie will take `O(N * k)` time, where `N` is the number of words in the dictionary, and `k` is their average length. To find the optimal first letters, we must traverse each path in the trie, which again takes `O(N * k)`. Therefore, this algorithm runs in `O(N * k)` time.