Daily Coding Problem #170

Problem

This problem was asked by Facebook.

Given a `start` word, an `end` word, and a dictionary of valid words, find the shortest transformation sequence from `start` to `end` such that only one letter is changed at each step of the sequence, and each transformed word exists in the dictionary. If there is no possible transformation, return null. Each word in the dictionary have the same length as `start` and `end` and is lowercase.

For example, given `start = "dog"`, `end = "cat"`, and `dictionary = {"dot", "dop", "dat", "cat"}`, return `["dog", "dot", "dat", "cat"]`.

Given `start = "dog"`, `end = "cat"`, and `dictionary = {"dot", "tod", "dat", "dar"}`, return null as there is no possible transformation from `dog` to `cat`.

Solution

We can model this problem as a graph: the nodes will be the words in the dictionary, and we can form an edge between two nodes if and only if one character can be modified in one word to get to the other.

Then we can do a typical breadth-first search starting from `start` and finishing once we encounter `end`:

```
def word_ladder(start, end, words):
    queue = deque([(start, [start])])

    while queue:
        word, path = queue.popleft()
        if word == end:
            return path
```

```
        for i in range(len(word)):
            for c in ascii_lowercase:
                next_word = word[:i] + c + word[i + 1:]
                if next_word in words:
                    words.remove(next_word)
                    queue.append([next_word, path + [next_word]])

    return None
```

This takes $O(n^2)$ time and O(n) space.