Daily Coding Problem #167

Problem

This problem was asked by Airbnb.

Given a list of words, find all pairs of unique indices such that the concatenation of the two words is a palindrome.

For example, given the list `["code", "edoc", "da", "d"]`, return `[(0, 1), (1, 0), (2, 3)]`.

Solution

The naive solution here would be to check each possible word pair for palindromicity and add their indices to the result:

```
def is_palindrome(word):
    return word == word[::-1]

def palindrome_pairs(words):
    result = []
    for i, word1 in enumerate(words):
        for j, word2 in enumerate(words):
            if i == j:
                continue
            if is_palindrome(word1 + word2):
                result.append((i, j))
    return result
```

This takes $O(n^2 * c)$ time where n is the number of words and c is the length of the longest word.

To speed it up, we can insert all words into a dictionary or hash table and then check each word's prefixes and postfixes. It will map from a word to its index in the list. If the reverse of a word's prefix/postfix is in the dictionary and its postfix/prefix is palindromic, then we add it to our list of results. For example, say we're looking at the word `aabc`. We check all its prefixes:

- Since `a` is a palindrome, we look for `cba` in the dictionary. If we find it, then we can make `cbaaabc`.
- Since `aa` is a palindrome, we look for `cb` in the dictionary. If we find it, then we can make `cbaabc`.
- `aab` and `aabc` are not palindromes, so we don't do anything.

And we do the same thing for the postfix.

```
def is_palindrome(word):
    return word == word[::-1]

def palindrome_pairs(words):
    d = {}
    for i, word in enumerate(words):
        d[word] = i

    result = []

    for i, word in enumerate(words):
        for char_i in range(len(word)):
            prefix, postfix = word[:char_i], word[char_i:]
            reversed_prefix, reversed_postfix = prefix[::-1], postfix[::-1]

            if is_palindrome(postfix) and reversed_prefix in d:
                if i != d[reversed_prefix]:
                    result.append((i, d[reversed_prefix]))

            if is_palindrome(prefix) and reversed_postfix in d:
                if i != d[reversed_postfix]:
                    result.append((d[reversed_postfix], i))

    return result
```

This should speed up the time to O(n * c^2). Since we will likely be constrained more by the number of words than the number of characters, this seems like an acceptable tradeoff.