# LC 127. Word Ladder

## Question

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

**Note:**

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

**Example 1:**

```
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
```

**Example 2:**

```
Input:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.
```

## Solution

```
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) ->
int:
        #Solution
        visited = set()
        wordList = set(wordList)
        q = collections.deque([(beginWord, 1)])
        alpha = string.ascii_lowercase
        while q:
            word, length = q.popleft()
            if word == endWord:
                return length
            for i in range(len(word)):
                for ch in alpha:
                    new_word = word[:i] + ch + word[i + 1:]
                    if new_word in wordList and new_word not in visited:
                        visited.add(new_word)
                        q.append((new_word, length + 1))
        return 0
```