```
import string
from util import *

def word_ladder_neighbors(current_word, valid_words):
    '''
    Given a word (current_word) as a string and a set of valid words,
    returns a list of words that are all one letter different
    than current_word.
    '''
    alphabet = string.ascii_lowercase
    neighboring_words = []
    alphabet = string.ascii_lowercase
    for i in range(len(current_word)):
        for letter in alphabet:
            new_word = current_word[:i] + letter + current_word[i+1:]
            if new_word in valid_words:
                neighboring_words.append(new_word)
    return neighboring_words

def word_ladder_search(valid_words, start_word, end_word):
    '''
    Given a list of valid words, a starting word (string), and an ending
    word (string), returns the path of of strings representing the word ladder from
    start_word to end_word.
    '''
    start = (start_word, [start_word])
    q = [start]
    expanded = set()

    while q:
        word, path = q.pop(0)
        if word == end_word:
            return path,len(expanded)
        if word not in expanded:
            expanded.add(word)
            for nextWord in word_ladder_neighbors(word, valid_words):
                if nextWord not in expanded:
                    q.append((nextWord, path+[nextWord]))

def better_word_ladder_search(valid_words, start_word, end_word):
    '''
    Given a list of valid words, a starting word (string), and an ending
    word (string), returns the path of of strings representing the word ladder from
    start_word to end_word. This search should use heuristic_function to speed up
    the search.
    '''
    start = (start_word, [start_word])
    q = PriorityQueue()
    q.update(start, 0)
    expanded = set()

    while q:
        word, path = q.pop()
```

```
        if word == end_word:
            return path,len(expanded)
        if word not in expanded:
            expanded.add(word)
            for nextWord in word_ladder_neighbors(word, valid_words):
                if nextWord not in expanded:
                    newState = (nextWord, path+[nextWord])
                    heuristicValue = heuristic_function(word, end_word)
                    q.update(newState, heuristicValue)

def heuristic_function(word, end_word):
    incorrect_letters=0
    for i in range(len(word)):
        if word[i]!=end_word[i]:
            incorrect_letters+=1
    return incorrect_letters


if __name__=="__main__":
    # valid_words is a set containing all strings that should be considered valid
    # words (all in lower-case)
    with open('words_alpha.txt') as f:
        valid_words = {i.strip() for i in f}

    print("You have", len(valid_words), "words to work with.")

    start = "start"
    end = "final"

    path, numExplored = word_ladder_search(valid_words, start, end)
    for word in path:
        print(word)
    print("You explored", numExplored, "words to get here!")
    print("Path length:",len(path))
```