Design Rationale

To compute word ladder recursively, we focused on doing a major part of the computation up front. Namely, we created a HashMap called WordMap, which takes in the dictionary and finds words that differ by one letter, called candidates, which are stored in an ArrayList. WordMap is a HashMap of words, their candidates, and the indices of the letter changed. We achieved this by created a class called wordsAndIndicies that contains an ArrayList of candidates and an ArrayList of which index was changed. This way, we do the majority of the work before we start to compute ladder and thus we don't have to run through the dictionary looking for candidates every time we find a new word, we just look-up the word in the WordMap and instantly get it's candidates. Our MakeLadder algorithm runs by checking a given word's candidates, sorting them by similarity to the endWord, and then going through them one-by-one until a match is found. A match is when the word is one letter different to the previous word and that letter is not in the same place as the previous changed letter (the first time we set changed index to -1). Once a match is found, we add the word to ladder and repeat the process. We try candidates most similar to the final word first so that we have a higher chance of finding the final word, and thus a shorter ladder. If the process is repeated and there is no match for the final word, we remove the word from the ladder and check other candidates until a match is found. If we have exhausted all candidates, we return false and ComputeLadder throws a NoSuchLadderException.