WordLadder.java

```
1  package com.example;
2
3  import java.util.HashSet;
4  import java.util.LinkedList;
5  import java.util.List;
6  import java.util.Queue;
7
8  /*
9      **Problem Statement:**
10     A transformation sequence from word beginWord to word endWord using a dictionary wordList is a
11    sequence of words beginWord -> s1 -> s2 -> ... -> sk such that:
12
13     Every adjacent pair of words differs by a single letter.
14     Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
15     sk == endWord
16     Given two words, beginWord and endWord, and a dictionary wordList, return the number of words in
17    the shortest transformation sequence from beginWord to endWord, or 0 if no such sequence exists.
18
19     **Example 1:**
20     Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
21     Output: 5
22     Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog",
23    which is 5 words long.
24
25     **Example 2:**
26     Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
27     Output: 0
28     Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation
29    sequence.
30
31     **Constraints:**
32     1 <= beginWord.length <= 10
33     endWord.length == beginWord.length
34     1 <= wordList.length <= 5000
35     wordList[i].length == beginWord.length
36     beginWord, endWord, and wordList[i] consist of lowercase English letters.
37     beginWord != endWord
38     All the words in wordList are unique.
39  */
40
41 class WordLadder {
42
43     /**
44      * This function finds the ladderLength
45      *
46      * @param beginWord: Starting word of the ladder
47      * @param endWord: Ending word of the ladder
48      * @param wordList: This list contains the words which needs to be included
49      * in ladder.
50      * @return ladderLength: This function will return the ladderLength(level)
51      * if the endword is there. Otherwise, will return the length as 0.
52      */
53     public static int ladderLength(String beginWord, String endWord, List<String> wordList) {
54         HashSet<String> set = new HashSet(wordList);
55
56 1       if (!set.contains(endWord)) {
57             return 0;
58         }
59
60         Queue<String> queue = new LinkedList();
61         queue.offer(beginWord);
62         int level = 1;
63
64 1       while (!queue.isEmpty()) {
65             int size = queue.size();
66 3           for (int i = 0; i < size; i++) {
67                 String curr = queue.poll();
68                 char[] words_chars = curr.toCharArray();
69 3              for (int j = 0; j < words_chars.length; j++) {
70                     char original_chars = words_chars[j];
71 3                  for (char c = 'a'; c <= 'z'; c++) {
72 1                      if (words_chars[j] == c) {
```

```
73                                  continue;
74                              }
75                              words_chars[j] = c;
76                              String new_word = String.valueOf(words_chars);
77 1                            if (new_word.equals(endWord)) {
78 2                                return level + 1;
79                              }
80 1                            if (set.contains(new_word)) {
81                                  set.remove(new_word);
82                                  queue.offer(new_word);
83                              }
84                          }
85                          words_chars[j] = original_chars;
86                      }
87                  }
88 1                level++;
89              }
90              return 0;
91          }
92      }
```

Mutations

```
56  1. negated conditional → KILLED
64  1. negated conditional → KILLED
    1. changed conditional boundary → KILLED
66  2. Changed increment from 1 to -1 → KILLED
    3. negated conditional → TIMED_OUT
    1. changed conditional boundary → KILLED
69  2. Changed increment from 1 to -1 → KILLED
    3. negated conditional → KILLED
    1. changed conditional boundary → SURVIVED
71  2. Replaced integer addition with subtraction → KILLED
    3. negated conditional → KILLED
72  1. negated conditional → KILLED
77  1. negated conditional → KILLED
    1. Replaced integer addition with subtraction → KILLED
78  2. replaced int return with 0 for com/example/WordLadder::ladderLength → KILLED
80  1. negated conditional → KILLED
88  1. Changed increment from 1 to -1 → KILLED
```

Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Tests examined

- com.example.WordLadderTest.testLadderLengthWithEmptyEndWord(com.example.WordLadderTest) (1 ms)
- com.example.WordLadderTest.testLadderLengthWithEmptyWordList(com.example.WordLadderTest) (1 ms)
- com.example.WordLadderTest.testLadderLengthWithValidTransformation(com.example.WordLadderTest) (1 ms)
- com.example.WordLadderTest.testLadderLengthWithInvalidTransformation(com.example.WordLadderTest) (0 ms)
- com.example.WordLadderTest.testLadderLengthWithEmptyBeginWord(com.example.WordLadderTest) (2 ms)

Report generated by PIT 1.5.0