Word Ladder Problem Documentation

Objective:

Given a starting word (beginWord), an ending word (endWord), and a dictionary of words (wordList), determine the number of words in the shortest transformation sequence from beginWord to endWord. Each transformation must change exactly one letter at a time, and each intermediate word must be in the wordList. If no such sequence exists, return 0.

Rules:

1. Each adjacent pair of words in the sequence differs by exactly one letter.
2. Each intermediate word (s1, s2, ..., sk) must be in the wordList.
3. The sequence starts with beginWord and ends with endWord (sk == endWord).

Example 1:

Input:

beginWord: "hit"

endWord: "cog"

wordList: ["hot", "dot", "dog", "lot", "log", "cog"]

Output: 5

Explanation: The shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> "cog", which consists of 5 words.

Example 2:

Input:

beginWord: "hit"

endWord: "cog"

wordList: ["hot", "dot", "dog", "lot", "log"]

Output: 0

Explanation: The endWord "cog" is not in wordList, so there is no valid transformation sequence.

Constraints:

- 1 <= beginWord.length <= 10
- endWord.length == beginWord.length
- 1 <= wordList.length <= 5000
- wordList[i].length == beginWord.length
- beginWord, endWord, and wordList[i] consist of lowercase English letters.
- beginWord is not equal to endWord.
- All the words in wordList are unique.

Approach:

The problem can be solved using Breadth-First Search (BFS), which is suitable for finding the shortest path in an unweighted graph. In this context, words can be viewed as nodes in a graph, with edges connecting nodes that can be transformed into one another by changing a single letter.

Steps:

1. **Initialization:**

- Check if endWord is in wordList. If not, return 0.
- Use a set for wordList for O(1) lookup times.
- Initialize a queue to perform BFS, starting with beginWord.

2. **BFS Execution:**

- Dequeue the front word and its transformation level.
- For each character in the word, attempt to change it to every letter from 'a' to 'z'.
- Generate a new word by changing one letter at a time.
- If the new word matches endWord, return the current level plus one.
- If the new word is in the wordSet, add it to the queue for further exploration and remove it from the wordSet to prevent re-processing.

3. **Completion:**

- If the queue is exhausted without finding endWord, return 0.

This approach ensures that the shortest transformation sequence is found, as BFS explores all nodes at the present depth level before moving on to nodes at the next depth level.

Conclusion

By employing BFS, the algorithm efficiently finds the shortest path from beginWord to endWord or determines that no such path exists. This method leverages the properties of BFS to explore the graph level by level, ensuring the shortest sequence is discovered.