Word Ladder II Problem Documentation

Problem Description

A transformation sequence from a word beginWord to a word endWord using a dictionary wordList is defined as a sequence of words [beginWord, s1, s2, ..., sk] such that:

- Every adjacent pair of words differs by a single letter.
- Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
- sk == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return all the shortest transformation sequences from beginWord to endWord, or an empty list if no such sequence exists. Each sequence should be returned as a list of the words [beginWord, s1, s2, ..., sk].

Example 1

- **Input:**
  - ➢ beginWord = "hit"
  - ➢ endWord = "cog"
  - ➢ wordList = ["hot", "dot", "dog", "lot", "log", "cog"]
- **Output:**
  - ➢ [["hit", "hot", "dot", "dog", "cog"], ["hit", "hot", "lot", "log", "cog"]]h
- **Explanation:**
  - ➢ *There are 2 shortest transformation sequences:*
    - ✓ "hit" -> "hot" -> "dot" -> "dog" -> "cog"

Example 2

- **Input:**
  - beginWord = "hit"
  - endWord = "cog"
  - wordList = ["hot", "dot", "dog", "lot", "log"]
- **Output:**
  - []
- **Explanation:**
  - The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

Constraints

- 1 <= beginWord.length <= 5
- endWord.length == beginWord.length
- 1 <= wordList.length <= 500
- wordList[i].length == beginWord.length
- beginWord, endWord, and wordList[i] consist of lowercase English letters.
- beginWord != endWord
- All the words in wordList are unique.
- The sum of all shortest transformation sequences does not exceed 10^5.

Solution Overview

The solution involves the following key steps:

1. **Neighbor Generation:**

- A function neighbor is defined to generate all possible words that differ by one letter from the given word.

2. BFS Initialization:

- A dictionary words is initialized to keep track of the current level of words and their transformation sequences.
- A set unvisited is created from wordList to keep track of words that have not been visited yet.

3. BFS Execution:

- A while loop performs a breadth-first search (BFS). The loop continues until no more words are processed or the endWord is found.
- For each word in the current level, all possible neighbours are generated.
- If a neighbour is unvisited, its transformation sequences are updated based on the current word's sequences.

4. Return Result:

- If the endWord is found in the dictionary, its sequences are returned. Otherwise, an empty list is returned.

This approach ensures that the shortest transformation sequences are found by exploring all possible paths level by level, thus guaranteeing the shortest path due to the nature of BFS.