Spell Checker:

A Python class called "Spell Checker" provides three major operations to quickly verify and correct word spelling: The Spell Checker receives the dictionary terms from a text file and saves them in a trie data structure.

1. Store the dictionary in an appropriate data structure:
The TrieNode class is used to implement the trie. Different file encodings are supported using the build_dictionary_from_file method, which also handles reading the file and creating the trie.

2. Find the nearest 4 words if the input word is not in the dictionary:
The find_nearest_words_for_misspelled technique is used by the Spell Checker to verify a word's correct spelling. The closest four terms in lexicographic order are returned if the input word cannot be found in the dictionary. This is done by looking through the trie and discovering the two words that come before and after the input word. 3. Add the input word to the dictionary:

3. Add the input word to the dictionary:
The add_to_dictionary method is available in the spell checker to add new terms to the trie-based dictionary. When this method is called, the input word is added into the trie, adding the new term to the dictionary. Additionally, it adds the new word to the dictionary's original file, guaranteeing that the dictionary is always up to date.

Time and Space Complexity Analysis:

Storing the dictionary in a trie:
Time Complexity: O(M * N), where M is the typical dictionary word length and N is the total number of dictionary terms. Each word in the dictionary must be traversed in order to build the trie.

Due to the trie's necessity to store every character associated with every word in the dictionary, the space complexity is O(M * N).

For a misspelt word, find the four closest words:

O(M), where M is the input word's length, is the measure of time complexity. In the worst situation, it goes through the depth of the trie once for each input word.

Space Complexity: O(M), since the length of the input word restricts the depth of the recursion.