**King Fahd University of Petroleum & Minerals**

**College of Computer Science and Engineering**

**Information and Computer Science Department**

**ICS 202 – Data Structures**

Name: Mohammed Saad Alsahli

ID : 202167910

1. Introduction explaining the lab project, the choice of data structure, the design decision, and the motivation behind it:

Project Overview: This project involves designing a dictionary data structure, primarily used in a spell checker. It supports operations like adding new words, searching, removing words, and finding similar words.

Choice of Data Structure: The AVL Tree was chosen due to its self-balancing nature, which ensures that the tree remains balanced with every insert or delete operation, providing O(log n) time complexity for search, insert, and delete operations.

Design Decisions: The decision to use AVL Tree over other data structures was influenced by the need for efficient search operations, which is crucial for a spell checker. Handling duplicates is managed by disallowing the insertion of a word already present in the tree, ensuring uniqueness.

Motivation: The motivation behind this project was to explore and apply advanced data structures in a practical scenario like a spell checker, which is a common feature in text editors.

1. For each operation of the Dictionary, include the problem-solving strategy in terms of plain English or pseudo-code

Initialization: The dictionary can be initialized in three ways - with a single string, as an empty dictionary, or from a file. In the case of file initialization, the program reads each line (word) and adds it to the AVL Tree, skipping duplicates.

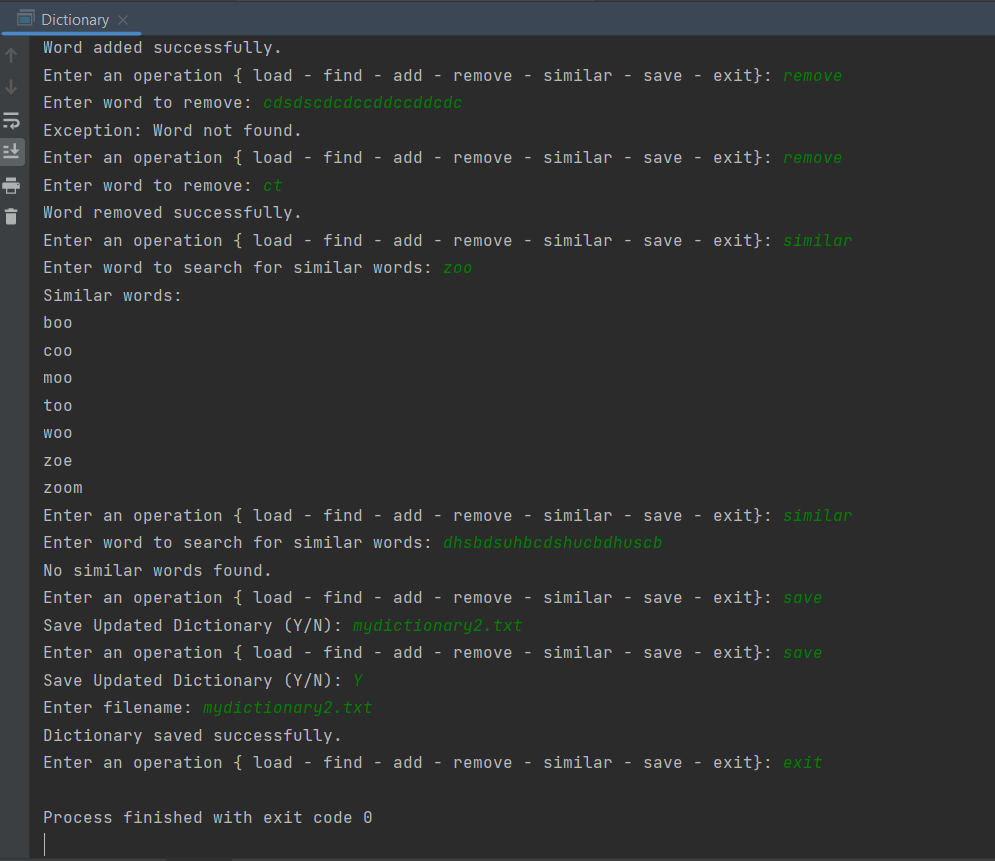
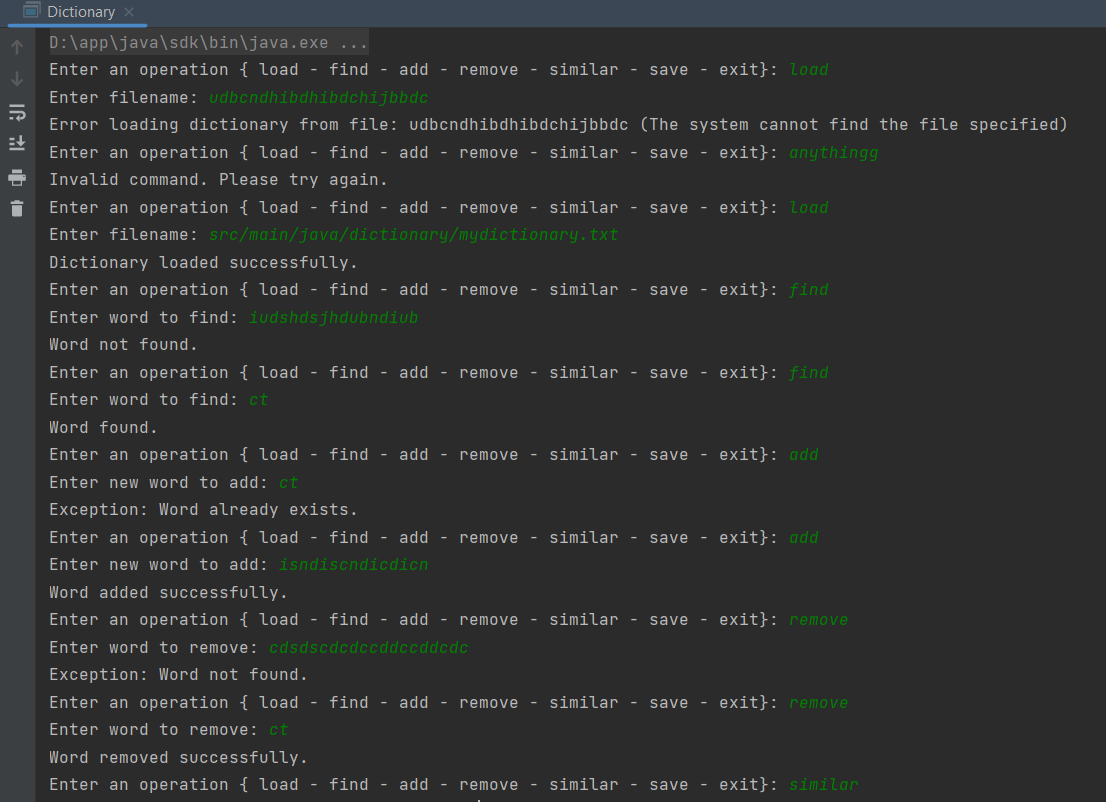
Add New Word: When adding a new word, the program first checks if the word already exists in the tree. If it does not, the word is inserted, and the tree rebalances itself if necessary.

Search for Word: The search operation traverses the AVL Tree, comparing the target word with the current node's word. Due to the tree's balanced nature, this operation is efficient.

Remove Word: To remove a word, the program searches for the word in the tree. If found, it's removed, and the tree is rebalanced. If the word is not found, an exception is thrown.

Search for Similar Words: This function traverses the entire tree, comparing each word with the target word to check if it differs by exactly one letter.

1. For each operation of the Dictionary, test results in terms of screenshots for at least three different scenarios

**

1. An explanation of the challenges you face in completing the project, and how you overcome it.

Challenge 1: Implementing the AVL Tree balancing logic.

Solution: Studied AVL Tree algorithms and carefully implemented and tested the left rotation, right rotation, and rebalancing logic.

Challenge 2: Efficiently finding similar words.

Solution: Developed a recursive algorithm to traverse the AVL Tree and compare each word with the target word based on the defined similarity criteria.