Data Structure Assignment #2

1. Overview

This assignment focuses on understanding and completing a Tree (BST, TRIE) implementation in C. Trees are fundamental data structures in computer science, enabling efficient searching, insertion, and deletion of data.

2. Assignment_2-1: Binary Search Tree (BST) Implementation

1) Core Concepts

- Binary Search Tree: A tree-like structure where each node has at most two children (left and right). Nodes in the left subtree have data values less than the parent node, while nodes in the right subtree have data values greater than the parent node.
- Nodes: The building blocks of a BST. Each node stores data and pointers to its left and right children.
- Insertion: Adding a new node to the tree while maintaining the BST property.
- Searching: Locating a specific data value within the tree.
- Deletion: Removing a node while preserving the BST property.

2) TODO Tasks

Fill in the TODO tasks in the given C file. The guides are given as follows.

- Insert Function Conditions: Implement the insert logic to insert a data in the BST.1

- Determine the condition to insert a new node in subtrees.

- Search Function Implementation: Implement the search logic to find a specific data value (data).
- Delete Function Implementation: Implement the logic for deleting a node with the specified data (data).

- Implement deletion on Leaf node, Node with one child, Node with two children.

3. Assignment_2-2: Trie Data Structure Implementation

1) Core Concepts

This assignment dives into implementing a Trie, a specialized tree-like data structure designed for efficient prefix-based operations on strings. Tries are commonly used for auto-complete, spell checking, and IP routing.

What is a Trie?

A Trie (pronounced "try" or sometimes "tree") is a specialized tree-like data structure that excels at storing and retrieving strings efficiently. It's also known as a prefix tree, radix tree, or digital tree. The fundamental idea is that each node in the trie represents a single character within a string. A path from the root of the trie to a specific node corresponds to a prefix of one or more stored strings.

Structure

- Trie: A tree where each node represents a character in a string. Paths from the root to nodes represent prefixes, and complete words are marked with a special flag (isEndOfWord in this implementation).
- Nodes: Store pointers to child nodes (one for each possible character in the alphabet) and a flag to indicate if the node marks the end of a word.
- Insertion: Adding a word character by character, creating new nodes as needed, and marking the last node as a word end.
- Search: Traversing the trie along the path corresponding to the search key. The word is found if the traversal reaches a node marked as a word end.
- Deletion: Removing a word by unmarking the word end and pruning any unnecessary nodes.
- Auto-Complete: Finding all words in the trie that share a given prefix.

2) TODO Tasks

Fill in the TODO tasks in the given C file. The guides are given as follows.

- Insertion Traversal: Implement the logic to traverse the trie for inserting a word (key).
- Search Traversal: Implement the logic to traverse the trie for searching a word (key).
- Delete Traversal: Implement the recursive traversal to find and delete nodes associated with the word (key).
    - Recursively delete child nodes, but only delete the current node if it's the end of a word AND has no remaining children.
- Suggestions Traversal: Implement the recursive traversal to find and print all words with the given prefix (currentPrefix).
    - When a node marked as a word end is reached, print the accumulated currentPrefix.
    - Recursively explore all child nodes to find additional words.