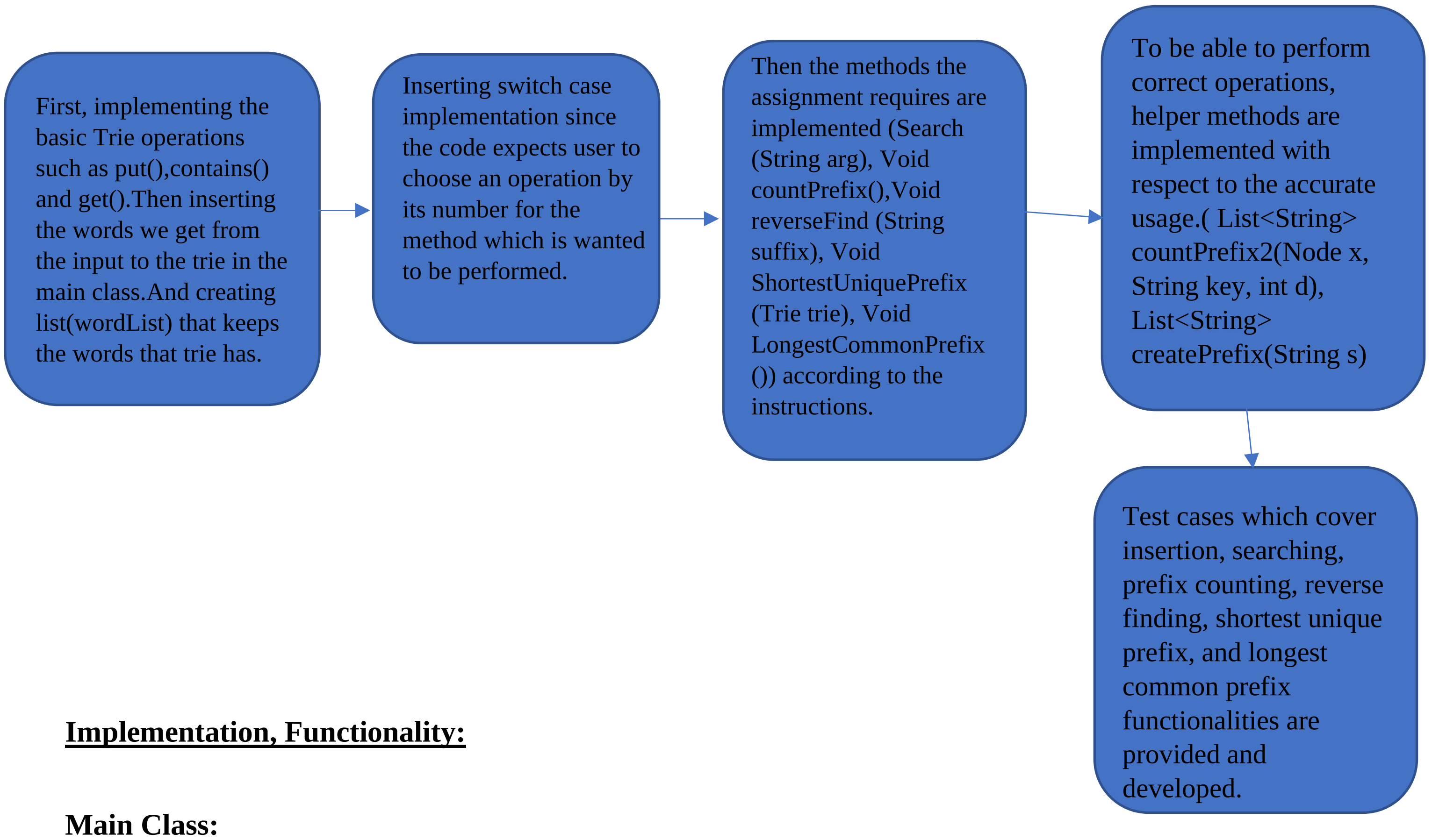
PELİN HAMDEMİR

Problem Statement and Code Design :

This assignment necessitates the implementation of a Trie data structure capable of supporting various operations such as searching, counting prefixes, finding words with a given suffix, determining the shortest unique prefix, and identifying the longest common prefix. The code design follows a modular approach, distinguishing between two main classes: Main and TrieST, facilitating user interaction and encapsulating Trie operations, respectively.

Top-down, stepwise refinement of the problem solution:



First, implementing the basic Trie operations such as put(),contains() and get().Then inserting the words we get from the input to the trie in the main class.And creating list(wordList) that keeps the words that trie has.

Inserting switch case implementation since the code expects user to choose an operation by its number for the method which is wanted to be performed.

Then the methods the assignment requires are implemented (Search (String arg), Void countPrefix(),Void reverseFind (String suffix), Void ShortestUniquePrefix (Trie trie), Void LongestCommonPrefix ()) according to the instructions.

To be able to perform correct operations, helper methods are implemented with respect to the accurate usage.(List<String> countPrefix2(Node x, String key, int d), List<String> createPrefix(String s)

Test cases which cover insertion, searching, prefix counting, reverse finding, shortest unique prefix, and longest common prefix functionalities are provided and developed.

Implementation, Functionality:

Main Class:

-Reads input and populates the Trie data structure.

-Accepts user input for various operations using a switch-case approach.

-Invokes corresponding methods in the TrieST class based on user input.

1. Switch Statements:

Operation Execution: Utilizes a switch statement to execute the chosen operation based on the user input.
Case 1 (Search): Reads a key from the user and invokes the Search method in the trie.
Case 2 (countPrefix): Invokes the countPrefix method.
Case 3 (reverseFind): Reads a suffix from the user and invokes the reverseFind method.
Case 4 (ShortestUniquePrefix): Invokes the ShortestUniquePrefix method, passing the trie object.
Case 5 (LongestCommonPrefix): Invokes the LongestCommonPrefix method, passing the number of inputs.
Case 6 (Exit): Exits the program.

TrieST Class:

-Implements a Trie data structure with nodes and basic operations like put, get, and contains.

-Supports additional operations like searching, counting prefixes, finding words with a given suffix, finding the shortest unique prefix, and finding the longest common prefix.

TrieST Methods:

1.put(String key, Value val)
Description: Inserts a key-value pair into the Trie.
Parameters:
key: The key to be inserted.
value: The corresponding value.
Algorithm:
Traverse the Trie, creating nodes as necessary for each character in the key.
Set the value of the last node to the provided value.

2. contains(String key)
Description: Checks if the Trie contains a given key.
Parameters:
key: The key to be checked.
Returns:
true if the key is present in the Trie, false otherwise.

3. Search(String arg)
Description: Searches for a key and prints True if found, False otherwise.
Parameters:
arg: The key to be searched.

4. countPrefix()
Description: Counts the occurrences of each prefix in the Trie and prints the results.
Algorithm:
Utilizes a recursive helper function (countPrefix2) to traverse the Trie and generate a list of all prefixes.
For each prefix, counts the occurrences by comparing with other terms in the list.

5. reverseFind(String suffix)
Description: Finds and prints words that end with a given suffix.
Parameters:
suffix: The suffix to search for.
Algorithm:
Iterates through the list of words, identifying those that end with the provided suffix.
Sorts the list lexicographically before printing.

6. ShortestUniquePrefix(TrieST trie)
Description: Finds and prints the shortest unique prefix for each word.
Parameters:
trie: The TrieST object.
Algorithm:
Iterates through the list of words.
For each word, creates a list of prefixes(profixList) and checks uniqueness by comparing with other words in the Trie.

7. createPrefix(String s)

Description: Creates a list of prefixes for a given string.
Parameters:
s: The input string.
Returns:
A list containing all prefixes of the input string.

8. LongestCommonPrefix(int NumOfInput)

Description: Finds and prints the longest common suffix among the words.
Parameters:
NumOfInput: The number of inputs that have been given from the user.
Algorithm:
Iterates through the list of words.
For each word, creates a list of prefixes and counts occurrences to find the longest common suffix.

Testing: The accurate insertion of words into the Trie (put), and word presence with the contains method are verified initially. Various Trie operations, including prefix counting (countPrefix), reverse finding (reverseFind),determining the shortest unique prefix (ShortestUniquePrefix) and the LongestCommonPrefix method for correct identification of the longest common suffix are checked if the output holds for different inputs.In addition their helper methods have been tested if the correct information has been delivered or not . The case that ShortestUniquePrefix(),reverseFind() and LongestCommonPrefix() has is “not exists” has been performed as test case. Assessing the swich-case user interaction, ensuring proper execution of operations and graceful program exit is controled. A meticulous testing approach is crucial for validating Trie operations and overall program functionality.

Final Assessments:

The assignment posed challenges in comprehending Trie operations, particularly in addressing complexities related to prefix counting and identifying the shortest unique prefix. The most demanding part involved crafting an efficient algorithm for the ShortestUniquePrefix() method, necessitating careful consideration of various cases and getting the correct output from every word. Despite the challenges, the assignment was beneficial in reinforcing understanding of Trie data structures and imroving Java programming skills.