Implementation of he NO OF WORDS IN A TRIE.

Mr. Vikas Kumar
Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	Competitive Coding
Lesson Title	Implementation of he NO OF WORDS IN A TRIE.

Lesson Objectives
To build a Trie data structure for efficient word storage.
To insert multiple words and mark the end of each word node.
To traverse the Trie and count all nodes where isEndOfWord = true.
To display the total number of distinct words stored in the Trie.

Problem Statement:

Write a program to count the total number of words stored in a Trie data structure.

1. Create Trie Node Structure
2. Insert Words into the Trie
3 . Define the Counting Function
4. Count Total Words
5. Return the total number of words

Concept

What is a Trie?

- A Trie (Prefix Tree) is a tree-like data structure used to store strings or words efficiently.
- Each node represents a character, and paths from the root form words.
- Each word ends at a node marked as endOfWord = true

Key Operations

- Insert(word) – Add a word to the Trie.
- Search(word) – Check if a word exists.
- CountWords() – Traverse the Trie to count how many words are stored.

Concept

Why Use Trie?

- Provides fast prefix-based searching (O(L), where L = word length).
- Useful for word auto-completion, spell checking, and dictionary applications.

Algorithm/Logic

1. Create Trie Node Structure

- Each Trie node contains:
  - children[26] → to store references for each alphabet letter (a–z)
  - isEndOfWord → boolean flag to mark the end of a word

2. Insert Words into the Trie

- For each word in the list:
  - Start from the root node.
  - For each character in the word:
    - Check if the child node for that character exists.
    - If not, create a new Trie node.
    - Move to the child node.
  - After the last character, mark isEndOfWord = true.

Algorithm/Logic

3. Define the Counting Function

- Create a recursive function countWords(node) that:
  - Initializes count = 0.
  - If node.isEndOfWord == true, increment count.
  - Recursively call countWords() for all non-null child nodes.
  - Add up all returned counts and return the total.

4. Count Total Words

- Call countWords(root) on the root node of the Trie.
- The function will return the total number of distinct words stored in the Trie.

Visualization

Let's insert → app, apple, bat, ball

Word Inserted	**Trie Path Created**	**isEndOfWord Set**	**Total Words So Far**
app	a → p → p	Yes (at last p)	1
apple	a → p → p → l → e	Yes (at e)	2
bat	b → a → t	Yes (at t)	3
ball	b → a → l → l	Yes (at last l)	4

Code Implementation

```
class TrieNode {
    TrieNode[] children;
    boolean isEndOfWord;

    public TrieNode() {
        children = new TrieNode[26];
        isEndOfWord = false;
    }
}

class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }
```

```
public void insert(String word) {
    TrieNode node = root;
    for (char ch : word.toCharArray()) {
        int index = ch - 'a';
        if (node.children[index] == null)
            node.children[index] = new TrieNode();
        node = node.children[index];
    }
    node.isEndOfWord = true;
}

// Count total words stored in the Trie
public int countWords() {
    return countWordsUtil(root);
}
```

```
private int countWordsUtil(TrieNode node) {
        if (node == null)
            return 0;

        int count = 0;
        if (node.isEndOfWord)
            count++;

        for (int i = 0; i < 26; i++)
            count += countWordsUtil(node.children[i]);

        return count;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Trie trie = new Trie();
        trie.insert("apple");
        trie.insert("app");
        trie.insert("bat");
        trie.insert("ball");

        System.out.println("Number of words in
Trie: " + trie.countWords());
    }
}
```

Output

```
Total words in Trie = 4
```

Example walkthrough

Let's insert -> cat, can , dog, dot, door

Word Inserted	Path Created in Trie	Path Created in Trie	Total Words So Far
cat	c → a → t	Yes (at 't')	1
can	c → a → n	Yes (at 'n')	2
dog	d → o → g	Yes (at 'g')	3
dot	d → o → t	Yes (at 't')	4
door	d → o → o → r	Yes (at 'r')	5

Time & Space Complexity

Operation	Time Complexity	Space Complexity	Description
Insertion	O(L)	O(L) (for new nodes)	Insert one word of length L
Counting Words	O(N)	O(h) (recursion stack, h = height of Trie)	Traverse all nodes to count words
Search (optional)	O(L)	O(1)	Check if a word exists
Total Space Used	-	O(N * alphabet size)	For all nodes in the Trie (26 pointers per node)

Summary

- A Trie (prefix tree) is a tree-based data structure used to store and search strings efficiently.
  Each node represents a character, and paths from root to terminal nodes form words.
  To count words, we traverse the Trie — each node that marks the end of a word increases the count.
  The algorithm uses:
- insert() – adds words letter by letter.
- countWords() – recursively or iteratively counts nodes marked as end-of-word.
  This provides fast insertion and word counting in O(n) time, where n is the total number of characters.

Summary

- This approach keeps all operations efficient (O(n) time, O(k) space).

- Advantage: Quick word insertion and counting without repeated string comparisons.
  Use Case: Ideal for dictionary-based applications like autocomplete or spell check.

Practice Questions:

1. Implement Trie (prefix Tree)— LeetCode #208

🔗 https://leetcode.com/problems/implement-trie-prefix-tree/

Concept: Implement a Trie data structure that efficiently stores and counts words.

Why Practice: This problem strengthens understanding of tree traversal and string storage optimization.
It's also useful for applications like autocomplete, dictionary word count, and spell checking.

Practice Questions:

2. Implement Trie – ii – prefix tree— LeetCode #1804

🔗 https://leetcode.com/problems/implement-trie-ii-prefix-tree/

Concept: Count exact words and prefixes using Trie nodes with word and prefix counters.

Why Practice: Strengthens understanding of efficient word storage, prefix counting, and Trie traversal.

Thanks