[**Implement Trie – II**](https://takeuforward.org/data-structure/implement-trie-ii/)

**Problem Statement:**  Implement a data structure ”TRIE” from scratch. Complete some functions.

1) Trie(): Initialize the object of this “TRIE” data structure.

2) insert(“WORD”): Insert the string “WORD”  into this “TRIE” data structure.

3) countWordsEqualTo(“WORD”): Return how many times this “WORD” is present in this “TRIE”.

4) countWordsStartingWith(“PREFIX”): Return how many words are there in this “TRIE” that have the string “PREFIX” as a prefix.

5) erase(“WORD”): Delete this string “WORD” from the “TRIE”.

**Note:**

1. If erase(“WORD”) function is called then it is guaranteed that the “WORD” is present in the “TRIE”.

2. If you are going to use variables with dynamic memory allocation then you need to release the memory associated with them at the end of your solution.

**Example:**

**Input:**

1

6

insert samsung

insert samsung

insert vivo

erase vivo

countWordsEqualTo samsung

countWordsStartingWith vi

**Output:**

2

0

**Explanation:**

Insert “samsung”: we are going to insert the word “samsung” into the “TRIE”.

Insert “samsung”: we are going to insert the word “samsung” again into the “TRIE”.

Insert “vivo”: we are going to insert the word “vivo” into the “TRIE”.

Erase “vivo”: we are going to delete the word “vivo” from the “TRIE”.

countWordsEqualTo “samsung”: There are two instances of “samsung” is present in “TRIE”.

countWordsStartWith “vi”:There is not a single word in the “TRIE” that starts from the prefix “vi”.

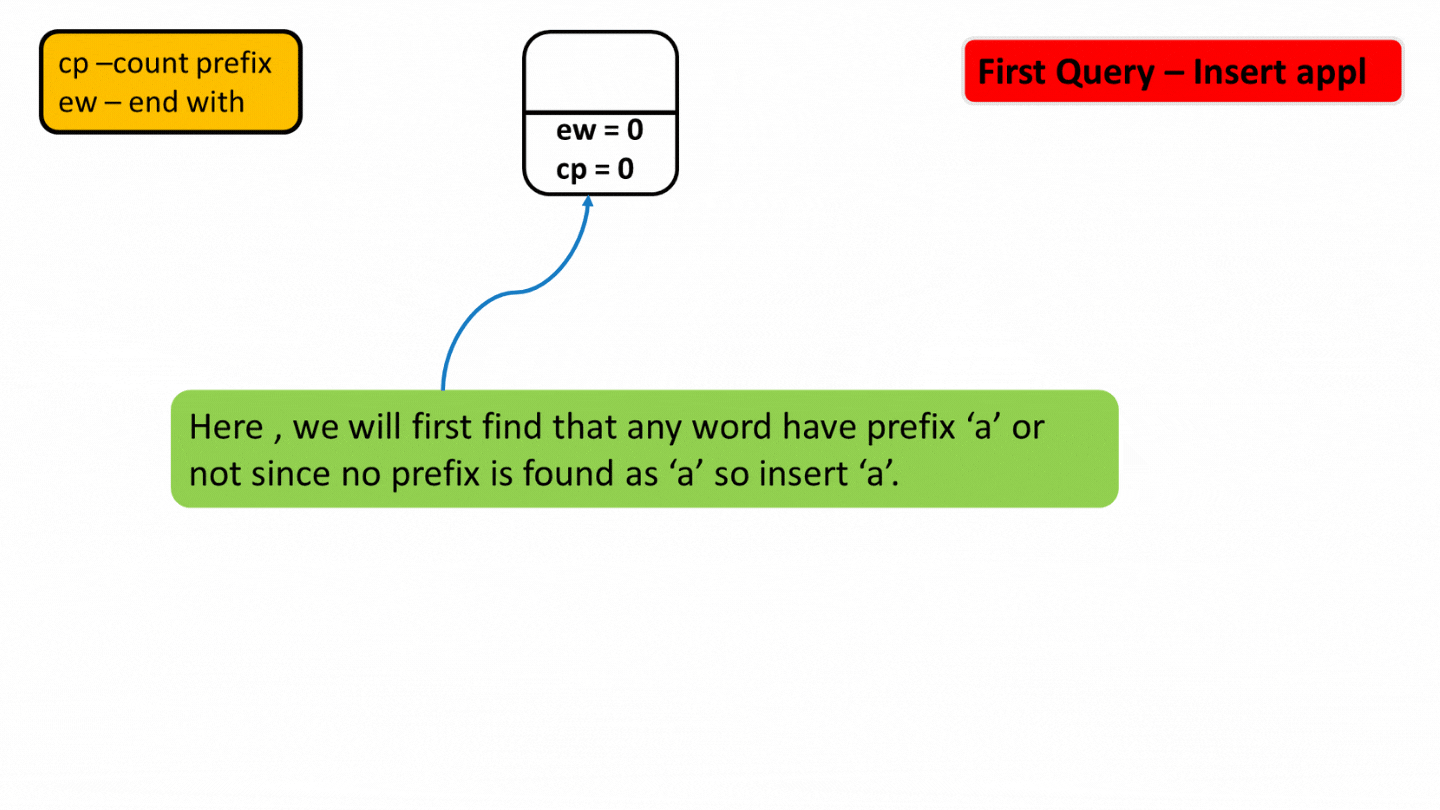
**Solution**

***Disclaimer***: *Don’t jump directly to the solution, try it out yourself first.*

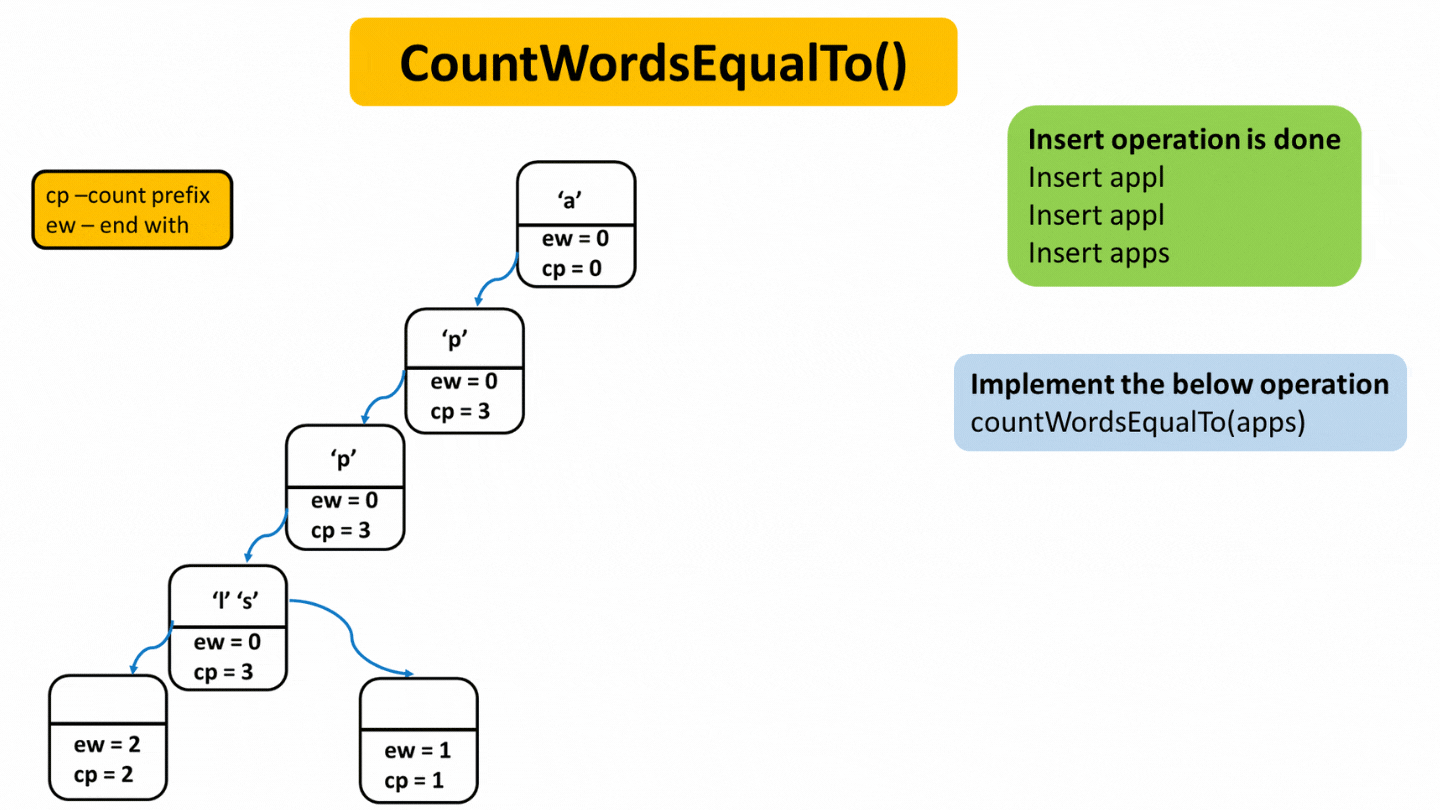
**Approach**:

1. **Insert**

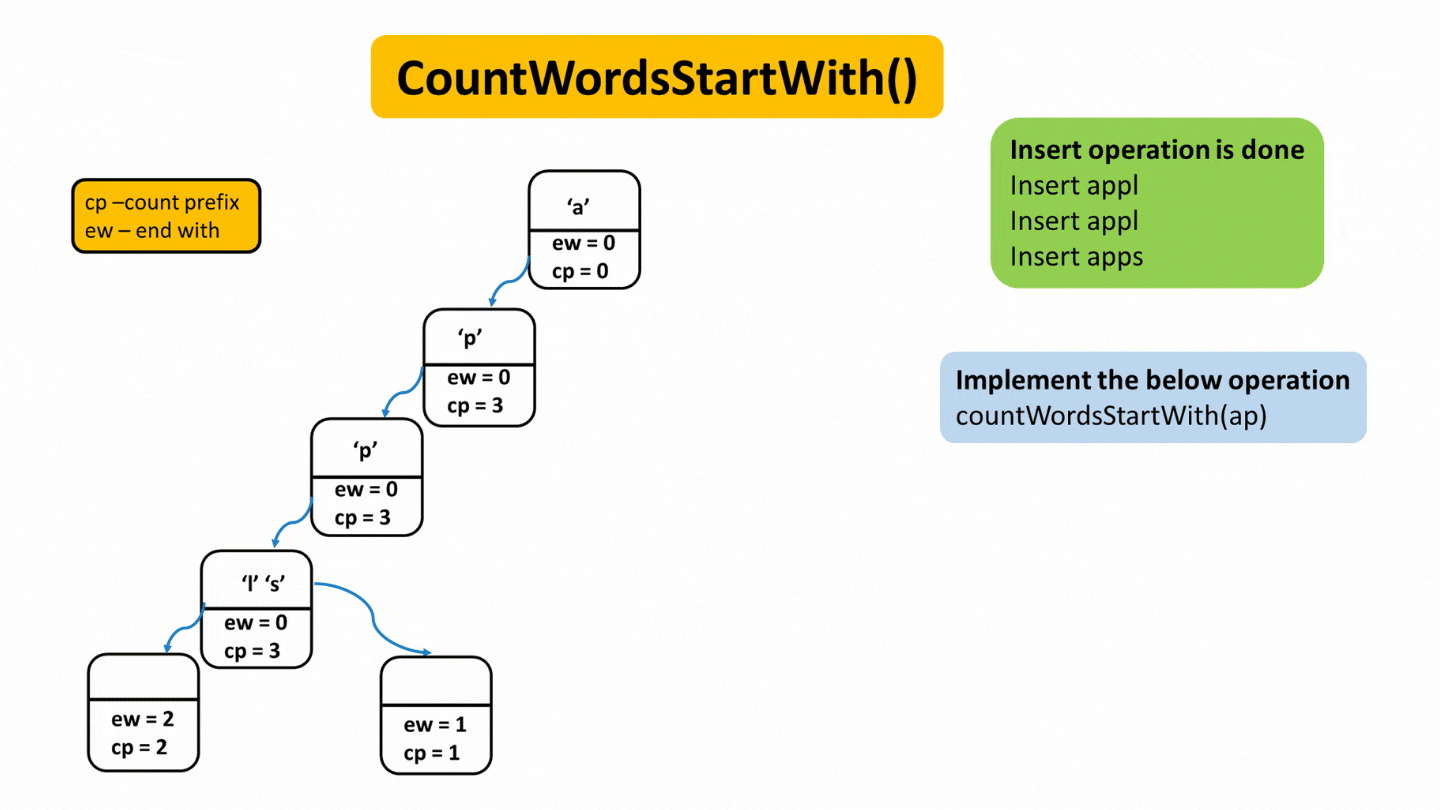
Here we will maintain **ew** -> end with  and  **cp** -> count prefix,  
We will insert letter by letter by creating a new track every time. Starting from the root, fill letters of the word one by one and subsequently increase cp. At the end of every word, increase ew which denotes that word finished here. At the time a different prefix is found, we will create a new track by increasing cp.



**2. countwordsEqualTo()** – This function will count complete words which are present in the Trie. Starting from the root, we will check for the given letter of the word to be found, if we find that cp >0 every time and at the end if (ew>0) this means word ends here means the complete word is present here.



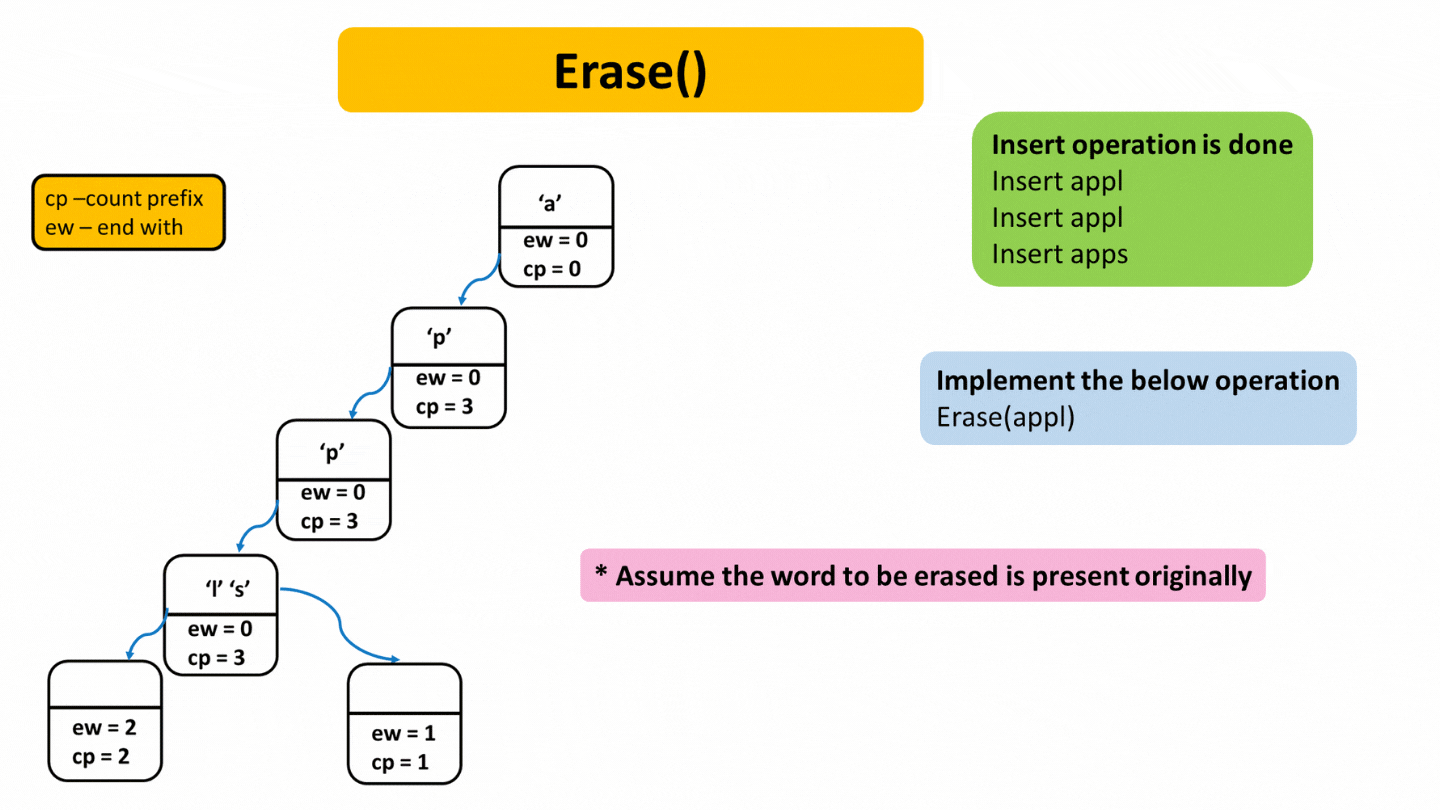
**3. countWordsStartWith()** – This function will count words starting with given string. Similarly, here also we begin from the root and check for the given letter one by one and check for cp>0. If it is, this means the given words are present here.

****

**4.** **Erase() –**It will delete the string given from the Trie.

Note- **In Erase() function**, it is assumed that the **given letter to be erased is present in the trie.**

Starting from the root, cp for the given prefix element is greater than 0, reduce cp by 1 and move on to a new track, and at last, reduce ew by 1  to mark that. word is fully deleted.

****

**Time Complexity:**O(N), where n denotes the length of the longest string that we are inserting.

**Space Complexity:**O(1), since constant size array created for links.