Problem 1804: Implement Trie II (Prefix Tree)

Problem Information

Difficulty: Medium
Acceptance Rate: 63.14%
Paid Only: Yes
Tags: Hash Table, String, Design, Trie

Problem Description

A [**trie**](https://en.wikipedia.org/wiki/Trie) (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

* `Trie()` Initializes the trie object. * `void insert(String word)` Inserts the string `word` into the trie. * `int countWordsEqualTo(String word)` Returns the number of instances of the string `word` in the trie. * `int countWordsStartingWith(String prefix)` Returns the number of strings in the trie that have the string `prefix` as a prefix. * `void erase(String word)` Erases the string `word` from the trie.

Example 1:

Input ["Trie", "insert", "insert", "countWordsEqualTo", "countWordsStartingWith", "erase", "countWordsEqualTo", "countWordsStartingWith", "erase", "countWordsStartingWith"] [[], ["apple"], ["apple"], ["apple"], ["app"], ["apple"], ["apple"], ["app"], ["apple"], ["app"]] **Output** [null, null, null, 2, 2, null, 1, 1, null, 0] **Explanation** Trie trie = new Trie(); trie.insert("apple"); // Inserts "apple". trie.insert("apple"); // Inserts another "apple". trie.countWordsEqualTo("apple"); // There are two instances of "apple" so return 2. trie.countWordsStartingWith("app"); // "app" is a prefix of "apple" so return 2. trie.erase("apple"); // Erases one "apple". trie.countWordsEqualTo("apple"); // Now there is only one instance of "apple" so return 1. trie.countWordsStartingWith("app"); // return 1 trie.erase("apple"); // Erases "apple". Now the trie is empty. trie.countWordsStartingWith("app"); // return 0

Constraints:

* `1 <= word.length, prefix.length <= 2000` * `word` and `prefix` consist only of lowercase English letters. * At most `3 * 104` calls **in total** will be made to `insert`, `countWordsEqualTo`, `countWordsStartingWith`, and `erase`. * It is guaranteed that for any function call to `erase`, the string `word` will exist in the trie.

Code Snippets

C++:

```
class Trie {
public:
Trie() {

}

void insert(string word) {

}

int countWordsEqualTo(string word) {

}

int countWordsStartingWith(string prefix) {

}

void erase(string word) {

}
};

/**
* Your Trie object will be instantiated and called as such:
* Trie* obj = new Trie();
* obj->insert(word);
* int param_2 = obj->countWordsEqualTo(word);
* int param_3 = obj->countWordsStartingWith(prefix);
* obj->erase(word);
*/
```

Java:

```
class Trie {

public Trie() {

}

public void insert(String word) {

}

public int countWordsEqualTo(String word) {

}

public int countWordsStartingWith(String prefix) {

}

public void erase(String word) {

}
}

/**
* Your Trie object will be instantiated and called as such:
* Trie obj = new Trie();
* obj.insert(word);
* int param_2 = obj.countWordsEqualTo(word);
* int param_3 = obj.countWordsStartingWith(prefix);
* obj.erase(word);
*/
```

Python3:

```
class Trie:

def __init__(self):


def insert(self, word: str) -> None:
```

```
def countWordsEqualTo(self, word: str) -> int:


def countWordsStartingWith(self, prefix: str) -> int:


def erase(self, word: str) -> None:



# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
# param_2 = obj.countWordsEqualTo(word)
# param_3 = obj.countWordsStartingWith(prefix)
# obj.erase(word)
```