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.