Problem 1804: Implement Trie II (Prefix Tree)

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

A

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 * 10

4

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)
```

Python:

```
class Trie(object):

def __init__(self):


def insert(self, word):
"""
:type word: str
:rtype: None
```

```
"""


def countWordsEqualTo(self, word):
"""
:type word: str
:rtype: int
"""


def countWordsStartingWith(self, prefix):
"""
:type prefix: str
:rtype: int
"""


def erase(self, word):
"""
:type word: str
:rtype: 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)
```

JavaScript:

```
var Trie = function() {

};

/**
* @param {string} word
* @return {void}
```

```
*/
Trie.prototype.insert = function(word) {

};

/**
* @param {string} word
* @return {number}
*/
Trie.prototype.countWordsEqualTo = function(word) {

};

/**
* @param {string} prefix
* @return {number}
*/
Trie.prototype.countWordsStartingWith = function(prefix) {

};

/**
* @param {string} word
* @return {void}
*/
Trie.prototype.erase = function(word) {

};

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

TypeScript:

```
class Trie {
constructor() {
```

```
}

insert(word: string): void {

}

countWordsEqualTo(word: string): number {

}

countWordsStartingWith(prefix: string): number {

}

erase(word: string): void {

}
}

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

C#:

```
public 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);
*/
```

C:

```
typedef struct {

} Trie;


Trie* trieCreate() {

}

void trieInsert(Trie* obj, char* word) {

}

int trieCountWordsEqualTo(Trie* obj, char* word) {

}
```

```
int trieCountWordsStartingWith(Trie* obj, char* prefix) {

}

void trieErase(Trie* obj, char* word) {

}

void trieFree(Trie* obj) {

}

/**
* Your Trie struct will be instantiated and called as such:
* Trie* obj = trieCreate();
* trieInsert(obj, word);

* int param_2 = trieCountWordsEqualTo(obj, word);

* int param_3 = trieCountWordsStartingWith(obj, prefix);

* trieErase(obj, word);

* trieFree(obj);
*/
```

Go:

```
type Trie struct {

}


func Constructor() Trie {

}


func (this *Trie) Insert(word string) {

}
```

```
func (this *Trie) CountWordsEqualTo(word string) int {

}


func (this *Trie) CountWordsStartingWith(prefix string) int {

}


func (this *Trie) Erase(word string) {

}


/**
* Your Trie object will be instantiated and called as such:
* obj := Constructor();
* obj.Insert(word);
* param_2 := obj.CountWordsEqualTo(word);
* param_3 := obj.CountWordsStartingWith(prefix);
* obj.Erase(word);
*/
```

Kotlin:

```
class Trie() {

fun insert(word: String) {

}

fun countWordsEqualTo(word: String): Int {

}

fun countWordsStartingWith(prefix: String): Int {

}
```

```
fun erase(word: String) {

}

}

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

Swift:

```
class Trie {

init() {

}

func insert(_ word: String) {

}

func countWordsEqualTo(_ word: String) -> Int {

}

func countWordsStartingWith(_ prefix: String) -> Int {

}

func erase(_ word: String) {

}
}

/**
```

```
* Your Trie object will be instantiated and called as such:
* let obj = Trie()
* obj.insert(word)
* let ret_2: Int = obj.countWordsEqualTo(word)
* let ret_3: Int = obj.countWordsStartingWith(prefix)
* obj.erase(word)
*/
```

Rust:

```
struct Trie {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Trie {

fn new() -> Self {

}

fn insert(&self, word: String) {

}

fn count_words_equal_to(&self, word: String) -> i32 {

}

fn count_words_starting_with(&self, prefix: String) -> i32 {

}

fn erase(&self, word: String) {

}
}
```

```
/**
* Your Trie object will be instantiated and called as such:
* let obj = Trie::new();
* obj.insert(word);
* let ret_2: i32 = obj.count_words_equal_to(word);
* let ret_3: i32 = obj.count_words_starting_with(prefix);
* obj.erase(word);
*/
```

Ruby:

```
class Trie
def initialize()

end


=begin
:type word: String
:rtype: Void
=end
def insert(word)

end


=begin
:type word: String
:rtype: Integer
=end
def count_words_equal_to(word)

end


=begin
:type prefix: String
:rtype: Integer
=end
def count_words_starting_with(prefix)

end
```

```
=begin
:type word: String
:rtype: Void
=end
def erase(word)

end


end

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

PHP:

```
class Trie {
/**
*/
function __construct() {

}

/**
* @param String $word
* @return NULL
*/
function insert($word) {

}

/**
* @param String $word
* @return Integer
*/
function countWordsEqualTo($word) {
```

```
}

/**
* @param String $prefix
* @return Integer
*/
function countWordsStartingWith($prefix) {

}

/**
* @param String $word
* @return NULL
*/
function erase($word) {

}
}

/**
* Your Trie object will be instantiated and called as such:
* $obj = Trie();
* $obj->insert($word);
* $ret_2 = $obj->countWordsEqualTo($word);
* $ret_3 = $obj->countWordsStartingWith($prefix);
* $obj->erase($word);
*/
```

Dart:

```
class Trie {

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 = Trie();
* obj.insert(word);
* int param2 = obj.countWordsEqualTo(word);
* int param3 = obj.countWordsStartingWith(prefix);
* obj.erase(word);
*/
```

Scala:

```
class Trie() {

def insert(word: String): Unit = {

}

def countWordsEqualTo(word: String): Int = {

}

def countWordsStartingWith(prefix: String): Int = {

}

def erase(word: String): Unit = {

}

}
```

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

Elixir:

```
defmodule Trie do
@spec init_() :: any
def init_() do

end

@spec insert(word :: String.t) :: any
def insert(word) do

end

@spec count_words_equal_to(word :: String.t) :: integer
def count_words_equal_to(word) do

end

@spec count_words_starting_with(prefix :: String.t) :: integer
def count_words_starting_with(prefix) do

end

@spec erase(word :: String.t) :: any
def erase(word) do

end
end

# Your functions will be called as such:
# Trie.init_()
# Trie.insert(word)
```

```
# param_2 = Trie.count_words_equal_to(word)
# param_3 = Trie.count_words_starting_with(prefix)
# Trie.erase(word)

# Trie.init_ will be called before every test case, in which you can do some
necessary initializations.
```

Erlang:

```
-spec trie_init_() -> any().
trie_init_() ->
.

-spec trie_insert(Word :: unicode:unicode_binary()) -> any().
trie_insert(Word) ->
.

-spec trie_count_words_equal_to(Word :: unicode:unicode_binary()) ->
integer().
trie_count_words_equal_to(Word) ->
.

-spec trie_count_words_starting_with(Prefix :: unicode:unicode_binary()) ->
integer().
trie_count_words_starting_with(Prefix) ->
.

-spec trie_erase(Word :: unicode:unicode_binary()) -> any().
trie_erase(Word) ->
.


%% Your functions will be called as such:
%% trie_init_(),
%% trie_insert(Word),
%% Param_2 = trie_count_words_equal_to(Word),
%% Param_3 = trie_count_words_starting_with(Prefix),
%% trie_erase(Word),

%% trie_init_ will be called before every test case, in which you can do some
necessary initializations.
```

Racket:

```
(define trie%
(class object%
(super-new)

(init-field)

; insert : string? -> void?
(define/public (insert word)
)
; count-words-equal-to : string? -> exact-integer?
(define/public (count-words-equal-to word)
)
; count-words-starting-with : string? -> exact-integer?
(define/public (count-words-starting-with prefix)
)
; erase : string? -> void?
(define/public (erase word)
)))

;; Your trie% object will be instantiated and called as such:
;; (define obj (new trie%))
;; (send obj insert word)
;; (define param_2 (send obj count-words-equal-to word))
;; (define param_3 (send obj count-words-starting-with prefix))
;; (send obj erase word)
```

Solutions

C++ Solution:

```
/*
* Problem: Implement Trie II (Prefix Tree)
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
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 Solution:

```
/**
* Problem: Implement Trie II (Prefix Tree)
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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 Solution:

```
"""
Problem: Implement Trie II (Prefix Tree)
Difficulty: Medium
```

```
Tags: string, tree, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Trie:

def __init__(self):


def insert(self, word: str) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Trie(object):

def __init__(self):


def insert(self, word):
"""
:type word: str
:rtype: None
"""


def countWordsEqualTo(self, word):
"""
:type word: str
:rtype: int
"""


def countWordsStartingWith(self, prefix):
"""
:type prefix: str
:rtype: int
```

```
"""



def erase(self, word):
"""
:type word: str
:rtype: 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)
```

JavaScript Solution:

```
/**
* Problem: Implement Trie II (Prefix Tree)
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/


var Trie = function() {

};

/**
* @param {string} word
* @return {void}
*/
Trie.prototype.insert = function(word) {
```

```
};

/**
* @param {string} word
* @return {number}
*/
Trie.prototype.countWordsEqualTo = function(word) {

};

/**
* @param {string} prefix
* @return {number}
*/
Trie.prototype.countWordsStartingWith = function(prefix) {

};

/**
* @param {string} word
* @return {void}
*/
Trie.prototype.erase = function(word) {

};

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

TypeScript Solution:

```
/**
* Problem: Implement Trie II (Prefix Tree)
* Difficulty: Medium
* Tags: string, tree, hash
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Trie {
constructor() {

}

insert(word: string): void {

}

countWordsEqualTo(word: string): number {

}

countWordsStartingWith(prefix: string): number {

}

erase(word: string): void {

}
}

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

C# Solution:

```
/*
* Problem: Implement Trie II (Prefix Tree)
```

```
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public 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);
*/
```

C Solution:

```c
/*
* Problem: Implement Trie II (Prefix Tree)
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/



typedef struct {

} Trie;


Trie* trieCreate() {

}

void trieInsert(Trie* obj, char* word) {

}

int trieCountWordsEqualTo(Trie* obj, char* word) {

}

int trieCountWordsStartingWith(Trie* obj, char* prefix) {

}

void trieErase(Trie* obj, char* word) {

}

void trieFree(Trie* obj) {

}

/**
```

```
* Your Trie struct will be instantiated and called as such:
* Trie* obj = trieCreate();
* trieInsert(obj, word);

* int param_2 = trieCountWordsEqualTo(obj, word);

* int param_3 = trieCountWordsStartingWith(obj, prefix);

* trieErase(obj, word);

* trieFree(obj);
*/
```

Go Solution:

```
// Problem: Implement Trie II (Prefix Tree)
// Difficulty: Medium
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

type Trie struct {

}


func Constructor() Trie {

}


func (this *Trie) Insert(word string) {

}


func (this *Trie) CountWordsEqualTo(word string) int {

}
```

```
func (this *Trie) CountWordsStartingWith(prefix string) int {

}


func (this *Trie) Erase(word string) {

}


/**
* Your Trie object will be instantiated and called as such:
* obj := Constructor();
* obj.Insert(word);
* param_2 := obj.CountWordsEqualTo(word);
* param_3 := obj.CountWordsStartingWith(prefix);
* obj.Erase(word);
*/
```

Kotlin Solution:

```
class Trie() {

fun insert(word: String) {

}

fun countWordsEqualTo(word: String): Int {

}

fun countWordsStartingWith(prefix: String): Int {

}

fun erase(word: String) {

}
```

```
}

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

Swift Solution:

```
class Trie {

init() {

}

func insert(_ word: String) {

}

func countWordsEqualTo(_ word: String) -> Int {

}

func countWordsStartingWith(_ prefix: String) -> Int {

}

func erase(_ word: String) {

}
}

/**
* Your Trie object will be instantiated and called as such:
* let obj = Trie()
* obj.insert(word)
```

```
* let ret_2: Int = obj.countWordsEqualTo(word)
* let ret_3: Int = obj.countWordsStartingWith(prefix)
* obj.erase(word)
*/
```

Rust Solution:

```
// Problem: Implement Trie II (Prefix Tree)
// Difficulty: Medium
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

struct Trie {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Trie {

fn new() -> Self {

}

fn insert(&self, word: String) {

}

fn count_words_equal_to(&self, word: String) -> i32 {

}

fn count_words_starting_with(&self, prefix: String) -> i32 {

}
```

```
fn erase(&self, word: String) {

}
}

/**
* Your Trie object will be instantiated and called as such:
* let obj = Trie::new();
* obj.insert(word);
* let ret_2: i32 = obj.count_words_equal_to(word);
* let ret_3: i32 = obj.count_words_starting_with(prefix);
* obj.erase(word);
*/
```

Ruby Solution:

```
class Trie
def initialize()

end


=begin
:type word: String
:rtype: Void
=end
def insert(word)

end


=begin
:type word: String
:rtype: Integer
=end
def count_words_equal_to(word)

end
```

```
=begin
:type prefix: String
:rtype: Integer
=end
def count_words_starting_with(prefix)

end


=begin
:type word: String
:rtype: Void
=end
def erase(word)

end


end

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

PHP Solution:

```
class Trie {
/**
*/
function __construct() {

}

/**
* @param String $word
* @return NULL
*/
function insert($word) {
```

```
}

/**
* @param String $word
* @return Integer
*/
function countWordsEqualTo($word) {

}

/**
* @param String $prefix
* @return Integer
*/
function countWordsStartingWith($prefix) {

}

/**
* @param String $word
* @return NULL
*/
function erase($word) {

}
}

/**
* Your Trie object will be instantiated and called as such:
* $obj = Trie();
* $obj->insert($word);
* $ret_2 = $obj->countWordsEqualTo($word);
* $ret_3 = $obj->countWordsStartingWith($prefix);
* $obj->erase($word);
*/
```

Dart Solution:

```
class Trie {
```

```
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 = Trie();
* obj.insert(word);
* int param2 = obj.countWordsEqualTo(word);
* int param3 = obj.countWordsStartingWith(prefix);
* obj.erase(word);
*/
```

Scala Solution:

```
class Trie() {

def insert(word: String): Unit = {

}

def countWordsEqualTo(word: String): Int = {

}
```

```
def countWordsStartingWith(prefix: String): Int = {

}

def erase(word: String): Unit = {

}

}

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

Elixir Solution:

```
defmodule Trie do
@spec init_() :: any
def init_() do

end

@spec insert(word :: String.t) :: any
def insert(word) do

end

@spec count_words_equal_to(word :: String.t) :: integer
def count_words_equal_to(word) do

end

@spec count_words_starting_with(prefix :: String.t) :: integer
def count_words_starting_with(prefix) do
```

```
end

@spec erase(word :: String.t) :: any
def erase(word) do

end
end

# Your functions will be called as such:
# Trie.init_()
# Trie.insert(word)
# param_2 = Trie.count_words_equal_to(word)
# param_3 = Trie.count_words_starting_with(prefix)
# Trie.erase(word)

# Trie.init_ will be called before every test case, in which you can do some
necessary initializations.
```

Erlang Solution:

```
-spec trie_init_() -> any().
trie_init_() ->
.

-spec trie_insert(Word :: unicode:unicode_binary()) -> any().
trie_insert(Word) ->
.

-spec trie_count_words_equal_to(Word :: unicode:unicode_binary()) ->
integer().
trie_count_words_equal_to(Word) ->
.

-spec trie_count_words_starting_with(Prefix :: unicode:unicode_binary()) ->
integer().
trie_count_words_starting_with(Prefix) ->
.

-spec trie_erase(Word :: unicode:unicode_binary()) -> any().
trie_erase(Word) ->
.
```

```
%% Your functions will be called as such:
%% trie_init_(),
%% trie_insert(Word),
%% Param_2 = trie_count_words_equal_to(Word),
%% Param_3 = trie_count_words_starting_with(Prefix),
%% trie_erase(Word),

%% trie_init_ will be called before every test case, in which you can do some
necessary initializations.
```

Racket Solution:

```
(define trie%
(class object%
(super-new)

(init-field)

; insert : string? -> void?
(define/public (insert word)
)
; count-words-equal-to : string? -> exact-integer?
(define/public (count-words-equal-to word)
)
; count-words-starting-with : string? -> exact-integer?
(define/public (count-words-starting-with prefix)
)
; erase : string? -> void?
(define/public (erase word)
)))

;; Your trie% object will be instantiated and called as such:
;; (define obj (new trie%))
;; (send obj insert word)
;; (define param_2 (send obj count-words-equal-to word))
;; (define param_3 (send obj count-words-starting-with prefix))
;; (send obj erase word)
```