Problem 745: Prefix and Suffix Search

Problem Information

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

Problem Description

Design a special dictionary that searches the words in it by a prefix and a suffix.

Implement the

WordFilter

class:

WordFilter(string[] words)

Initializes the object with the

words

in the dictionary.

f(string pref, string suff)

Returns

the index of the word in the dictionary,

which has the prefix

pref

and the suffix

suff

. If there is more than one valid index, return

the largest

of them. If there is no such word in the dictionary, return

-1

.

Example 1:

Input

["WordFilter", "f"] [[["apple"]], ["a", "e"]]

Output

[null, 0]

Explanation

WordFilter wordFilter = new WordFilter(["apple"]); wordFilter.f("a", "e"); // return 0, because the word at index 0 has prefix = "a" and suffix = "e".

Constraints:

1 <= words.length <= 10

4

1 <= words[i].length <= 7

1 <= pref.length, suff.length <= 7

words[i]

,

pref

and

suff

consist of lowercase English letters only.

At most

10

4

calls will be made to the function

f

.

Code Snippets

C++:

```
class WordFilter {
public:
WordFilter(vector<string>& words) {

}

int f(string pref, string suff) {

}
};
```

```
/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter* obj = new WordFilter(words);
* int param_1 = obj->f(pref,suff);
*/
```

Java:

```
class WordFilter {

public WordFilter(String[] words) {

}

public int f(String pref, String suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter obj = new WordFilter(words);
* int param_1 = obj.f(pref,suff);
*/
```

Python3:

```
class WordFilter:

def __init__(self, words: List[str]):


def f(self, pref: str, suff: str) -> int:



# Your WordFilter object will be instantiated and called as such:
# obj = WordFilter(words)
# param_1 = obj.f(pref,suff)
```

Python:

```
class WordFilter(object):

def __init__(self, words):
"""
:type words: List[str]
"""


def f(self, pref, suff):
"""
:type pref: str
:type suff: str
:rtype: int
"""



# Your WordFilter object will be instantiated and called as such:
# obj = WordFilter(words)
# param_1 = obj.f(pref,suff)
```

JavaScript:

```
/**
* @param {string[]} words
*/
var WordFilter = function(words) {

};

/**
* @param {string} pref
* @param {string} suff
* @return {number}
*/
WordFilter.prototype.f = function(pref, suff) {

};

/**
* Your WordFilter object will be instantiated and called as such:
* var obj = new WordFilter(words)
* var param_1 = obj.f(pref,suff)
```

```
*/
```

TypeScript:

```
class WordFilter {
constructor(words: string[]) {

}

f(pref: string, suff: string): number {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* var obj = new WordFilter(words)
* var param_1 = obj.f(pref,suff)
*/
```

C#:

```
public class WordFilter {

public WordFilter(string[] words) {

}

public int F(string pref, string suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter obj = new WordFilter(words);
* int param_1 = obj.F(pref,suff);
*/
```

C:

```
typedef struct {

} WordFilter;


WordFilter* wordFilterCreate(char** words, int wordsSize) {

}

int wordFilterF(WordFilter* obj, char* pref, char* suff) {

}

void wordFilterFree(WordFilter* obj) {

}

/**
* Your WordFilter struct will be instantiated and called as such:
* WordFilter* obj = wordFilterCreate(words, wordsSize);
* int param_1 = wordFilterF(obj, pref, suff);

* wordFilterFree(obj);
*/
```

Go:

```
type WordFilter struct {

}


func Constructor(words []string) WordFilter {

}


func (this *WordFilter) F(pref string, suff string) int {

}
```

```
/**
* Your WordFilter object will be instantiated and called as such:
* obj := Constructor(words);
* param_1 := obj.F(pref,suff);
*/
```

Kotlin:

```
class WordFilter(words: Array<String>) {

fun f(pref: String, suff: String): Int {

}

}

/**
* Your WordFilter object will be instantiated and called as such:
* var obj = WordFilter(words)
* var param_1 = obj.f(pref,suff)
*/
```

Swift:

```
class WordFilter {

init(_ words: [String]) {

}

func f(_ pref: String, _ suff: String) -> Int {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* let obj = WordFilter(words)
* let ret_1: Int = obj.f(pref, suff)
```

```
*/
```

Rust:

```
struct WordFilter {

}


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

fn new(words: Vec<String>) -> Self {

}

fn f(&self, pref: String, suff: String) -> i32 {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* let obj = WordFilter::new(words);
* let ret_1: i32 = obj.f(pref, suff);
*/
```

Ruby:

```
class WordFilter

=begin
:type words: String[]
=end
def initialize(words)

end
```

```
=begin
:type pref: String
:type suff: String
:rtype: Integer
=end
def f(pref, suff)

end


end

# Your WordFilter object will be instantiated and called as such:
# obj = WordFilter.new(words)
# param_1 = obj.f(pref, suff)
```

PHP:

```
class WordFilter {
/**
* @param String[] $words
*/
function __construct($words) {

}

/**
* @param String $pref
* @param String $suff
* @return Integer
*/
function f($pref, $suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* $obj = WordFilter($words);
* $ret_1 = $obj->f($pref, $suff);
*/
```

Dart:

```
class WordFilter {

WordFilter(List<String> words) {

}

int f(String pref, String suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter obj = WordFilter(words);
* int param1 = obj.f(pref,suff);
*/
```

Scala:

```
class WordFilter(_words: Array[String]) {

def f(pref: String, suff: String): Int = {

}

}

/**
* Your WordFilter object will be instantiated and called as such:
* val obj = new WordFilter(words)
* val param_1 = obj.f(pref,suff)
*/
```

Elixir:

```
defmodule WordFilter do
@spec init_(words :: [String.t]) :: any
def init_(words) do

end
```

```
@spec f(pref :: String.t, suff :: String.t) :: integer
def f(pref, suff) do

end
end

# Your functions will be called as such:
# WordFilter.init_(words)
# param_1 = WordFilter.f(pref, suff)

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

Erlang:

```
-spec word_filter_init_(Words :: [unicode:unicode_binary()]) -> any().
word_filter_init_(Words) ->
.

-spec word_filter_f(Pref :: unicode:unicode_binary(), Suff ::
unicode:unicode_binary()) -> integer().
word_filter_f(Pref, Suff) ->
.


%% Your functions will be called as such:
%% word_filter_init_(Words),
%% Param_1 = word_filter_f(Pref, Suff),

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

Racket:

```
(define word-filter%
(class object%
(super-new)

; words : (listof string?)
(init-field
words)
```

```
; f : string? string? -> exact-integer?
(define/public (f pref suff)
)))

;; Your word-filter% object will be instantiated and called as such:
;; (define obj (new word-filter% [words words]))
;; (define param_1 (send obj f pref suff))
```

Solutions

C++ Solution:

```
/*
* Problem: Prefix and Suffix Search
* Difficulty: Hard
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class WordFilter {
public:
WordFilter(vector<string>& words) {

}

int f(string pref, string suff) {

}
};

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter* obj = new WordFilter(words);
* int param_1 = obj->f(pref,suff);
*/
```

Java Solution:

```
/**
* Problem: Prefix and Suffix Search
* Difficulty: Hard
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class WordFilter {

public WordFilter(String[] words) {

}

public int f(String pref, String suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter obj = new WordFilter(words);
* int param_1 = obj.f(pref,suff);
*/
```

Python3 Solution:

```
"""
Problem: Prefix and Suffix Search
Difficulty: Hard
Tags: array, string, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class WordFilter:

def __init__(self, words: List[str]):
```

```
def f(self, pref: str, suff: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class WordFilter(object):

def __init__(self, words):
"""
:type words: List[str]
"""


def f(self, pref, suff):
"""
:type pref: str
:type suff: str
:rtype: int
"""



# Your WordFilter object will be instantiated and called as such:
# obj = WordFilter(words)
# param_1 = obj.f(pref,suff)
```

JavaScript Solution:

```
/**
* Problem: Prefix and Suffix Search
* Difficulty: Hard
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* @param {string[]} words
*/
var WordFilter = function(words) {

};

/**
* @param {string} pref
* @param {string} suff
* @return {number}
*/
WordFilter.prototype.f = function(pref, suff) {

};

/**
* Your WordFilter object will be instantiated and called as such:
* var obj = new WordFilter(words)
* var param_1 = obj.f(pref,suff)
*/
```

TypeScript Solution:

```
/**
* Problem: Prefix and Suffix Search
* Difficulty: Hard
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class WordFilter {
constructor(words: string[]) {

}

f(pref: string, suff: string): number {
```

```
}
}

/**
* Your WordFilter object will be instantiated and called as such:
* var obj = new WordFilter(words)
* var param_1 = obj.f(pref,suff)
*/
```

C# Solution:

```
/*
* Problem: Prefix and Suffix Search
* Difficulty: Hard
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class WordFilter {

public WordFilter(string[] words) {

}

public int F(string pref, string suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter obj = new WordFilter(words);
* int param_1 = obj.F(pref,suff);
*/
```

C Solution:

```
/*
* Problem: Prefix and Suffix Search
* Difficulty: Hard
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} WordFilter;


WordFilter* wordFilterCreate(char** words, int wordsSize) {

}

int wordFilterF(WordFilter* obj, char* pref, char* suff) {

}

void wordFilterFree(WordFilter* obj) {

}

/**
* Your WordFilter struct will be instantiated and called as such:
* WordFilter* obj = wordFilterCreate(words, wordsSize);
* int param_1 = wordFilterF(obj, pref, suff);

* wordFilterFree(obj);
*/
```

Go Solution:

```
// Problem: Prefix and Suffix Search
// Difficulty: Hard
```

```
// Tags: array, string, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type WordFilter struct {

}


func Constructor(words []string) WordFilter {

}


func (this *WordFilter) F(pref string, suff string) int {

}


/**
* Your WordFilter object will be instantiated and called as such:
* obj := Constructor(words);
* param_1 := obj.F(pref,suff);
*/
```

Kotlin Solution:

```
class WordFilter(words: Array<String>) {

fun f(pref: String, suff: String): Int {

}

}

/**
* Your WordFilter object will be instantiated and called as such:
* var obj = WordFilter(words)
* var param_1 = obj.f(pref,suff)
```

```
*/
```

Swift Solution:

```
class WordFilter {

init(_ words: [String]) {

}

func f(_ pref: String, _ suff: String) -> Int {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* let obj = WordFilter(words)
* let ret_1: Int = obj.f(pref, suff)
*/
```

Rust Solution:

```
// Problem: Prefix and Suffix Search
// Difficulty: Hard
// Tags: array, string, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct WordFilter {

}


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

```
impl WordFilter {

fn new(words: Vec<String>) -> Self {

}

fn f(&self, pref: String, suff: String) -> i32 {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* let obj = WordFilter::new(words);
* let ret_1: i32 = obj.f(pref, suff);
*/
```

Ruby Solution:

```
class WordFilter

=begin
:type words: String[]
=end
def initialize(words)

end


=begin
:type pref: String
:type suff: String
:rtype: Integer
=end
def f(pref, suff)

end


end
```

```
# Your WordFilter object will be instantiated and called as such:
# obj = WordFilter.new(words)
# param_1 = obj.f(pref, suff)
```

PHP Solution:

```
class WordFilter {
/**
* @param String[] $words
*/
function __construct($words) {

}

/**
* @param String $pref
* @param String $suff
* @return Integer
*/
function f($pref, $suff) {

}
}

/**
* Your WordFilter object will be instantiated and called as such:
* $obj = WordFilter($words);
* $ret_1 = $obj->f($pref, $suff);
*/
```

Dart Solution:

```
class WordFilter {

WordFilter(List<String> words) {

}

int f(String pref, String suff) {

}
```

```
}

/**
* Your WordFilter object will be instantiated and called as such:
* WordFilter obj = WordFilter(words);
* int param1 = obj.f(pref,suff);
*/
```

Scala Solution:

```
class WordFilter(_words: Array[String]) {

def f(pref: String, suff: String): Int = {

}

}

/**
* Your WordFilter object will be instantiated and called as such:
* val obj = new WordFilter(words)
* val param_1 = obj.f(pref,suff)
*/
```

Elixir Solution:

```
defmodule WordFilter do
@spec init_(words :: [String.t]) :: any
def init_(words) do

end

@spec f(pref :: String.t, suff :: String.t) :: integer
def f(pref, suff) do

end
end

# Your functions will be called as such:
# WordFilter.init_(words)
# param_1 = WordFilter.f(pref, suff)
```

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

Erlang Solution:

```
-spec word_filter_init_(Words :: [unicode:unicode_binary()]) -> any().
word_filter_init_(Words) ->
.

-spec word_filter_f(Pref :: unicode:unicode_binary(), Suff :: unicode:unicode_binary()) -> integer().
word_filter_f(Pref, Suff) ->
.


%% Your functions will be called as such:
%% word_filter_init_(Words),
%% Param_1 = word_filter_f(Pref, Suff),

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

Racket Solution:

```
(define word-filter%
(class object%
(super-new)

; words : (listof string?)
(init-field
words)

; f : string? string? -> exact-integer?
(define/public (f pref suff)
)))

;; Your word-filter% object will be instantiated and called as such:
;; (define obj (new word-filter% [words words]))
;; (define param_1 (send obj f pref suff))
```