Problem 211: Design Add and Search Words Data Structure

Problem Information

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

Problem Description

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the

WordDictionary

class:

WordDictionary()

Initializes the object.

void addWord(word)

Adds

word

to the data structure, it can be matched later.

bool search(word)

Returns

true

if there is any string in the data structure that matches

word

or

false

otherwise.

word

may contain dots

'.'

where dots can be matched with any letter.

Example:

Input

["WordDictionary","addWord","addWord","addWord","search","search","search","search"]
[[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]

Output

[null,null,null,null,false,true,true,true]

Explanation

WordDictionary wordDictionary = new WordDictionary(); wordDictionary.addWord("bad"); wordDictionary.addWord("dad"); wordDictionary.addWord("mad"); wordDictionary.search("pad"); // return False wordDictionary.search("bad"); // return True wordDictionary.search(".ad"); // return True wordDictionary.search("b.."); // return True

Constraints:

1 <= word.length <= 25

word

in

addWord

consists of lowercase English letters.

word

in

search

consist of

'.'

or lowercase English letters.

There will be at most

2

dots in

word

for

search

queries.

At most

10

4

calls will be made to

addWord

and

search

.

Code Snippets

C++:

```
class WordDictionary {
public:
WordDictionary() {

}

void addWord(string word) {

}

bool search(string word) {

}
};

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary* obj = new WordDictionary();
* obj->addWord(word);
* bool param_2 = obj->search(word);
*/
```

Java:

```
class WordDictionary {

public WordDictionary() {

}

public void addWord(String word) {

}

public boolean search(String word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary obj = new WordDictionary();
* obj.addWord(word);
* boolean param_2 = obj.search(word);
*/
```

Python3:

```
class WordDictionary:

def __init__(self):


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


def search(self, word: str) -> bool:



# Your WordDictionary object will be instantiated and called as such:
# obj = WordDictionary()
# obj.addWord(word)
# param_2 = obj.search(word)
```

Python:

```
class WordDictionary(object):

def __init__(self):



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



def search(self, word):
"""
:type word: str
:rtype: bool
"""




# Your WordDictionary object will be instantiated and called as such:
# obj = WordDictionary()
# obj.addWord(word)
# param_2 = obj.search(word)
```

JavaScript:

```


var WordDictionary = function() {

};

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

};

/**
```

```
* @param {string} word
* @return {boolean}
*/
WordDictionary.prototype.search = function(word) {

};

/**
* Your WordDictionary object will be instantiated and called as such:
* var obj = new WordDictionary()
* obj.addWord(word)
* var param_2 = obj.search(word)
*/
```

TypeScript:

```
class WordDictionary {
constructor() {

}

addWord(word: string): void {

}

search(word: string): boolean {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* var obj = new WordDictionary()
* obj.addWord(word)
* var param_2 = obj.search(word)
*/
```

C#:

```
public class WordDictionary {

public WordDictionary() {
}

public void AddWord(string word) {

}

public bool Search(string word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary obj = new WordDictionary();
* obj.AddWord(word);
* bool param_2 = obj.Search(word);
*/
```

C:

```
typedef struct {

} WordDictionary;


WordDictionary* wordDictionaryCreate() {

}

void wordDictionaryAddWord(WordDictionary* obj, char* word) {

}

bool wordDictionarySearch(WordDictionary* obj, char* word) {

}

void wordDictionaryFree(WordDictionary* obj) {
}

/**
* Your WordDictionary struct will be instantiated and called as such:
* WordDictionary* obj = wordDictionaryCreate();
* wordDictionaryAddWord(obj, word);

* bool param_2 = wordDictionarySearch(obj, word);

* wordDictionaryFree(obj);
*/
```

Go:

```
type WordDictionary struct {

}



func Constructor() WordDictionary {

}



func (this *WordDictionary) AddWord(word string) {

}



func (this *WordDictionary) Search(word string) bool {

}



/**
* Your WordDictionary object will be instantiated and called as such:
* obj := Constructor();
* obj.AddWord(word);
* param_2 := obj.Search(word);
*/
```

Kotlin:

```
class WordDictionary() {

fun addWord(word: String) {

}

fun search(word: String): Boolean {

}

}

/**
* Your WordDictionary object will be instantiated and called as such:
* var obj = WordDictionary()
* obj.addWord(word)
* var param_2 = obj.search(word)
*/
```

Swift:

```

class WordDictionary {

init() {

}

func addWord(_ word: String) {

}

func search(_ word: String) -> Bool {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* let obj = WordDictionary()
* obj.addWord(word)
* let ret_2: Bool = obj.search(word)
*/
```

Rust:

```
struct WordDictionary {

}



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

fn new() -> Self {

}

fn add_word(&self, word: String) {

}

fn search(&self, word: String) -> bool {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* let obj = WordDictionary::new();
* obj.add_word(word);
* let ret_2: bool = obj.search(word);
*/
```

Ruby:

```
class WordDictionary
def initialize()
```

```
end



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

end



=begin
:type word: String
:rtype: Boolean
=end
def search(word)

end



end

# Your WordDictionary object will be instantiated and called as such:
# obj = WordDictionary.new()
# obj.add_word(word)
# param_2 = obj.search(word)
```

PHP:

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

}

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

/**
* @param String $word
* @return Boolean
*/
function search($word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* $obj = WordDictionary();
* $obj->addWord($word);
* $ret_2 = $obj->search($word);
*/
```

Dart:

```
class WordDictionary {

WordDictionary() {

}

void addWord(String word) {

}

bool search(String word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary obj = WordDictionary();
* obj.addWord(word);
* bool param2 = obj.search(word);
*/
```

Scala:

```
class WordDictionary() {

def addWord(word: String): Unit = {

}

def search(word: String): Boolean = {

}

}

/**
* Your WordDictionary object will be instantiated and called as such:
* val obj = new WordDictionary()
* obj.addWord(word)
* val param_2 = obj.search(word)
*/
```

Elixir:

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

end

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

end

@spec search(word :: String.t) :: boolean
def search(word) do

end
end

# Your functions will be called as such:
# WordDictionary.init_()
# WordDictionary.add_word(word)
# param_2 = WordDictionary.search(word)

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

Erlang:

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

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

-spec word_dictionary_search(Word :: unicode:unicode_binary()) -> boolean().
word_dictionary_search(Word) ->
.



%% Your functions will be called as such:
%% word_dictionary_init_(),
%% word_dictionary_add_word(Word),
%% Param_2 = word_dictionary_search(Word),

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

Racket:

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

(init-field)

; add-word : string? -> void?
(define/public (add-word word)
)
; search : string? -> boolean?
(define/public (search word)
)))

;; Your word-dictionary% object will be instantiated and called as such:
;; (define obj (new word-dictionary%))
;; (send obj add-word word)
;; (define param_2 (send obj search word))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Add and Search Words Data Structure
* Difficulty: Medium
* Tags: string, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class WordDictionary {
public:
WordDictionary() {

}

void addWord(string word) {

}

bool search(string word) {

}
};

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary* obj = new WordDictionary();
* obj->addWord(word);
* bool param_2 = obj->search(word);
*/
```

Java Solution:

```
/**
* Problem: Design Add and Search Words Data Structure
* Difficulty: Medium
* Tags: string, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class WordDictionary {

public WordDictionary() {

}

public void addWord(String word) {

}

public boolean search(String word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary obj = new WordDictionary();
* obj.addWord(word);
* boolean param_2 = obj.search(word);
*/
```

Python3 Solution:

```
"""
Problem: Design Add and Search Words Data Structure
```

```
Difficulty: Medium
Tags: string, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class WordDictionary:

def __init__(self):



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

Python Solution:

```
class WordDictionary(object):

def __init__(self):



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



def search(self, word):
"""
:type word: str
:rtype: bool
"""




# Your WordDictionary object will be instantiated and called as such:
# obj = WordDictionary()
# obj.addWord(word)
# param_2 = obj.search(word)
```

JavaScript Solution:

```
/**
* Problem: Design Add and Search Words Data Structure
* Difficulty: Medium
* Tags: string, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/


var WordDictionary = function() {

};

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

};

/**
* @param {string} word
* @return {boolean}
*/
WordDictionary.prototype.search = function(word) {

};

/**
* Your WordDictionary object will be instantiated and called as such:
* var obj = new WordDictionary()
* obj.addWord(word)
* var param_2 = obj.search(word)
```

```
*/
```

TypeScript Solution:

```
/**
* Problem: Design Add and Search Words Data Structure
* Difficulty: Medium
* Tags: string, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class WordDictionary {
constructor() {

}

addWord(word: string): void {

}

search(word: string): boolean {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* var obj = new WordDictionary()
* obj.addWord(word)
* var param_2 = obj.search(word)
*/
```

C# Solution:

```
/*
* Problem: Design Add and Search Words Data Structure
* Difficulty: Medium
* Tags: string, search
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class WordDictionary {

public WordDictionary() {

}

public void AddWord(string word) {

}

public bool Search(string word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary obj = new WordDictionary();
* obj.AddWord(word);
* bool param_2 = obj.Search(word);
*/
```

C Solution:

```
/*
* Problem: Design Add and Search Words Data Structure
* Difficulty: Medium
* Tags: string, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
typedef struct {

} WordDictionary;


WordDictionary* wordDictionaryCreate() {

}

void wordDictionaryAddWord(WordDictionary* obj, char* word) {

}

bool wordDictionarySearch(WordDictionary* obj, char* word) {

}

void wordDictionaryFree(WordDictionary* obj) {

}

/**
* Your WordDictionary struct will be instantiated and called as such:
* WordDictionary* obj = wordDictionaryCreate();
* wordDictionaryAddWord(obj, word);

* bool param_2 = wordDictionarySearch(obj, word);

* wordDictionaryFree(obj);
*/
```

Go Solution:

```
// Problem: Design Add and Search Words Data Structure
// Difficulty: Medium
// Tags: string, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type WordDictionary struct {

}


func Constructor() WordDictionary {

}


func (this *WordDictionary) AddWord(word string) {

}


func (this *WordDictionary) Search(word string) bool {

}


/**
* Your WordDictionary object will be instantiated and called as such:
* obj := Constructor();
* obj.AddWord(word);
* param_2 := obj.Search(word);
*/
```

Kotlin Solution:

```
class WordDictionary() {

fun addWord(word: String) {

}

fun search(word: String): Boolean {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* var obj = WordDictionary()
* obj.addWord(word)
* var param_2 = obj.search(word)
*/
```

Swift Solution:

```
class WordDictionary {

init() {

}

func addWord(_ word: String) {

}

func search(_ word: String) -> Bool {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* let obj = WordDictionary()
* obj.addWord(word)
* let ret_2: Bool = obj.search(word)
*/
```

Rust Solution:

```
// Problem: Design Add and Search Words Data Structure
// Difficulty: Medium
// Tags: string, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct WordDictionary {

}



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

fn new() -> Self {

}

fn add_word(&self, word: String) {

}

fn search(&self, word: String) -> bool {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* let obj = WordDictionary::new();
* obj.add_word(word);
* let ret_2: bool = obj.search(word);
*/
```

Ruby Solution:

```
class WordDictionary
def initialize()

end
```

```


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

end


=begin
:type word: String
:rtype: Boolean
=end
def search(word)

end


end

# Your WordDictionary object will be instantiated and called as such:
# obj = WordDictionary.new()
# obj.add_word(word)
# param_2 = obj.search(word)
```

PHP Solution:

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

}

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

/**
* @param String $word
* @return Boolean
*/
function search($word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* $obj = WordDictionary();
* $obj->addWord($word);
* $ret_2 = $obj->search($word);
*/
```

Dart Solution:

```
class WordDictionary {

WordDictionary() {

}

void addWord(String word) {

}

bool search(String word) {

}
}

/**
* Your WordDictionary object will be instantiated and called as such:
* WordDictionary obj = WordDictionary();
* obj.addWord(word);
* bool param2 = obj.search(word);
*/
```

Scala Solution:

```
class WordDictionary() {

def addWord(word: String): Unit = {

}

def search(word: String): Boolean = {

}

}

/**
* Your WordDictionary object will be instantiated and called as such:
* val obj = new WordDictionary()
* obj.addWord(word)
* val param_2 = obj.search(word)
*/
```

Elixir Solution:

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

end

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

end

@spec search(word :: String.t) :: boolean
def search(word) do

end
end

# Your functions will be called as such:
# WordDictionary.init_()
# WordDictionary.add_word(word)
# param_2 = WordDictionary.search(word)

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

Erlang Solution:

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

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

-spec word_dictionary_search(Word :: unicode:unicode_binary()) -> boolean().
word_dictionary_search(Word) ->
.



%% Your functions will be called as such:
%% word_dictionary_init_(),
%% word_dictionary_add_word(Word),
%% Param_2 = word_dictionary_search(Word),

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

Racket Solution:

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

(init-field)

; add-word : string? -> void?
(define/public (add-word word)
)
; search : string? -> boolean?
(define/public (search word)
)))

;; Your word-dictionary% object will be instantiated and called as such:
;; (define obj (new word-dictionary%))
;; (send obj add-word word)
;; (define param_2 (send obj search word))
```