Problem 676: Implement Magic Dictionary

Problem Information

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

Problem Description

Design a data structure that is initialized with a list of

different

words. Provided a string, you should determine if you can change exactly one character in this string to match any word in the data structure.

Implement the

MagicDictionary

class:

MagicDictionary()

Initializes the object.

void buildDict(String[] dictionary)

Sets the data structure with an array of distinct strings

dictionary

.

bool search(String searchWord)

Returns

true

if you can change

exactly one character

in

searchWord

to match any string in the data structure, otherwise returns

false

.

Example 1:

Input

["MagicDictionary", "buildDict", "search", "search", "search", "search"] [[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]

Output

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

Explanation

MagicDictionary magicDictionary = new MagicDictionary(); magicDictionary.buildDict(["hello", "leetcode"]); magicDictionary.search("hello"); // return False magicDictionary.search("hhllo"); // We can change the second 'h' to 'e' to match "hello" so we return True magicDictionary.search("hell"); // return False magicDictionary.search("leetcoded"); // return False

Constraints:

1 <= dictionary.length <= 100

1 <= dictionary[i].length <= 100

dictionary[i]

consists of only lower-case English letters.

All the strings in

dictionary

are

distinct

.

1 <= searchWord.length <= 100

searchWord

consists of only lower-case English letters.

buildDict

will be called only once before

search

.

At most

100

calls will be made to

search

.

Code Snippets

C++:

```
class MagicDictionary {
public:
MagicDictionary() {

}

void buildDict(vector<string> dictionary) {

}

bool search(string searchWord) {

}
};

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary* obj = new MagicDictionary();
* obj->buildDict(dictionary);
* bool param_2 = obj->search(searchWord);
*/
```

Java:

```
class MagicDictionary {

public MagicDictionary() {

}

public void buildDict(String[] dictionary) {

}

public boolean search(String searchWord) {
}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary obj = new MagicDictionary();
* obj.buildDict(dictionary);
* boolean param_2 = obj.search(searchWord);
*/
```

Python3:

```
class MagicDictionary:

def __init__(self):



def buildDict(self, dictionary: List[str]) -> None:



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




# Your MagicDictionary object will be instantiated and called as such:
# obj = MagicDictionary()
# obj.buildDict(dictionary)
# param_2 = obj.search(searchWord)
```

Python:

```
class MagicDictionary(object):

def __init__(self):



def buildDict(self, dictionary):
"""
:type dictionary: List[str]
:rtype: None
"""
```

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



# Your MagicDictionary object will be instantiated and called as such:
# obj = MagicDictionary()
# obj.buildDict(dictionary)
# param_2 = obj.search(searchWord)
```

JavaScript:

```
var MagicDictionary = function() {

};

/**
* @param {string[]} dictionary
* @return {void}
*/
MagicDictionary.prototype.buildDict = function(dictionary) {

};

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

};

/**
* Your MagicDictionary object will be instantiated and called as such:
* var obj = new MagicDictionary()
* obj.buildDict(dictionary)
* var param_2 = obj.search(searchWord)
*/
```

TypeScript:

```
class MagicDictionary {
constructor() {

}

buildDict(dictionary: string[]): void {

}

search(searchWord: string): boolean {

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* var obj = new MagicDictionary()
* obj.buildDict(dictionary)
* var param_2 = obj.search(searchWord)
*/
```

C#:

```
public class MagicDictionary {

public MagicDictionary() {

}

public void BuildDict(string[] dictionary) {

}

public bool Search(string searchWord) {

}
}
```

```
/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary obj = new MagicDictionary();
* obj.BuildDict(dictionary);
* bool param_2 = obj.Search(searchWord);
*/
```

C:

```
typedef struct {

} MagicDictionary;


MagicDictionary* magicDictionaryCreate() {

}

void magicDictionaryBuildDict(MagicDictionary* obj, char** dictionary, int
dictionarySize) {

}

bool magicDictionarySearch(MagicDictionary* obj, char* searchWord) {

}

void magicDictionaryFree(MagicDictionary* obj) {

}

/**
* Your MagicDictionary struct will be instantiated and called as such:
* MagicDictionary* obj = magicDictionaryCreate();
* magicDictionaryBuildDict(obj, dictionary, dictionarySize);

* bool param_2 = magicDictionarySearch(obj, searchWord);
```

}

```
* magicDictionaryFree(obj);
*/
```

Go:

```
type MagicDictionary struct {

}


func Constructor() MagicDictionary {

}


func (this *MagicDictionary) BuildDict(dictionary []string) {

}


func (this *MagicDictionary) Search(searchWord string) bool {

}


/**
* Your MagicDictionary object will be instantiated and called as such:
* obj := Constructor();
* obj.BuildDict(dictionary);
* param_2 := obj.Search(searchWord);
*/
```

Kotlin:

```
class MagicDictionary() {

fun buildDict(dictionary: Array<String>) {

}

fun search(searchWord: String): Boolean {
}

}

/**
* Your MagicDictionary object will be instantiated and called as such:
* var obj = MagicDictionary()
* obj.buildDict(dictionary)
* var param_2 = obj.search(searchWord)
*/
```

Swift:

```
class MagicDictionary {

init() {

}

func buildDict(_ dictionary: [String]) {

}

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

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* let obj = MagicDictionary()
* obj.buildDict(dictionary)
* let ret_2: Bool = obj.search(searchWord)
*/
```

Rust:

```
struct MagicDictionary {

}
```

```


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

fn new() -> Self {

}

fn build_dict(&self, dictionary: Vec<String>) {

}

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

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* let obj = MagicDictionary::new();
* obj.build_dict(dictionary);
* let ret_2: bool = obj.search(searchWord);
*/
```

Ruby:

```
class MagicDictionary
def initialize()

end



=begin
:type dictionary: String[]
:rtype: Void
=end
def build_dict(dictionary)

end
```

```



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

end



end

# Your MagicDictionary object will be instantiated and called as such:
# obj = MagicDictionary.new()
# obj.build_dict(dictionary)
# param_2 = obj.search(search_word)
```

PHP:

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

}

/**
* @param String[] $dictionary
* @return NULL
*/
function buildDict($dictionary) {

}

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

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* $obj = MagicDictionary();
* $obj->buildDict($dictionary);
* $ret_2 = $obj->search($searchWord);
*/
```

Dart:

```
class MagicDictionary {

MagicDictionary() {

}

void buildDict(List<String> dictionary) {

}

bool search(String searchWord) {

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary obj = MagicDictionary();
* obj.buildDict(dictionary);
* bool param2 = obj.search(searchWord);
*/
```

Scala:

```
class MagicDictionary() {

def buildDict(dictionary: Array[String]): Unit = {

}

def search(searchWord: String): Boolean = {


}

}

/**
* Your MagicDictionary object will be instantiated and called as such:
* val obj = new MagicDictionary()
* obj.buildDict(dictionary)
* val param_2 = obj.search(searchWord)
*/
```

Elixir:

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

end

@spec build_dict(dictionary :: [String.t]) :: any
def build_dict(dictionary) do

end

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

end
end

# Your functions will be called as such:
# MagicDictionary.init_()
# MagicDictionary.build_dict(dictionary)
# param_2 = MagicDictionary.search(search_word)

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

Erlang:

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

-spec magic_dictionary_build_dict(Dictionary :: [unicode:unicode_binary()])
-> any().
magic_dictionary_build_dict(Dictionary) ->
.

-spec magic_dictionary_search(SearchWord :: unicode:unicode_binary()) ->
boolean().
magic_dictionary_search(SearchWord) ->
.


%% Your functions will be called as such:
%% magic_dictionary_init_(),
%% magic_dictionary_build_dict(Dictionary),
%% Param_2 = magic_dictionary_search(SearchWord),

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

Racket:

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

(init-field)

; build-dict : (listof string?) -> void?
(define/public (build-dict dictionary)
)
; search : string? -> boolean?
(define/public (search search-word)
)))

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

Solutions

C++ Solution:

```
/*
* Problem: Implement Magic Dictionary
* Difficulty: Medium
* 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 MagicDictionary {
public:
MagicDictionary() {

}

void buildDict(vector<string> dictionary) {

}

bool search(string searchWord) {

}
};

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary* obj = new MagicDictionary();
* obj->buildDict(dictionary);
* bool param_2 = obj->search(searchWord);
*/
```

Java Solution:

```
/**
* Problem: Implement Magic Dictionary
* Difficulty: Medium
* 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 MagicDictionary {

public MagicDictionary() {

}

public void buildDict(String[] dictionary) {

}

public boolean search(String searchWord) {

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary obj = new MagicDictionary();
* obj.buildDict(dictionary);
* boolean param_2 = obj.search(searchWord);
*/
```

Python3 Solution:

```
"""
Problem: Implement Magic Dictionary
Difficulty: Medium
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 MagicDictionary:

def __init__(self):



def buildDict(self, dictionary: List[str]) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class MagicDictionary(object):

def __init__(self):



def buildDict(self, dictionary):
"""
:type dictionary: List[str]
:rtype: None
"""



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




# Your MagicDictionary object will be instantiated and called as such:
# obj = MagicDictionary()
# obj.buildDict(dictionary)
# param_2 = obj.search(searchWord)
```

JavaScript Solution:

```
/**
* Problem: Implement Magic Dictionary
* Difficulty: Medium
* 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
*/


var MagicDictionary = function() {

};

/**
* @param {string[]} dictionary
* @return {void}
*/
MagicDictionary.prototype.buildDict = function(dictionary) {

};

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

};

/**
* Your MagicDictionary object will be instantiated and called as such:
* var obj = new MagicDictionary()
* obj.buildDict(dictionary)
* var param_2 = obj.search(searchWord)
*/
```

TypeScript Solution:

```
/**
* Problem: Implement Magic Dictionary
* Difficulty: Medium
* 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 MagicDictionary {
constructor() {

}

buildDict(dictionary: string[]): void {

}

search(searchWord: string): boolean {

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* var obj = new MagicDictionary()
* obj.buildDict(dictionary)
* var param_2 = obj.search(searchWord)
*/
```

C# Solution:

```
/*
* Problem: Implement Magic Dictionary
* Difficulty: Medium
* 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 MagicDictionary {
```

```
public MagicDictionary() {

}

public void BuildDict(string[] dictionary) {

}

public bool Search(string searchWord) {

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary obj = new MagicDictionary();
* obj.BuildDict(dictionary);
* bool param_2 = obj.Search(searchWord);
*/
```

C Solution:

```
/*
* Problem: Implement Magic Dictionary
* Difficulty: Medium
* 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 {

} MagicDictionary;


MagicDictionary* magicDictionaryCreate() {
}

void magicDictionaryBuildDict(MagicDictionary* obj, char** dictionary, int
dictionarySize) {

}

bool magicDictionarySearch(MagicDictionary* obj, char* searchWord) {

}

void magicDictionaryFree(MagicDictionary* obj) {

}

/**
* Your MagicDictionary struct will be instantiated and called as such:
* MagicDictionary* obj = magicDictionaryCreate();
* magicDictionaryBuildDict(obj, dictionary, dictionarySize);

* bool param_2 = magicDictionarySearch(obj, searchWord);

* magicDictionaryFree(obj);
*/
```

Go Solution:

```
// Problem: Implement Magic Dictionary
// Difficulty: Medium
// 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 MagicDictionary struct {

}
```

```
func Constructor() MagicDictionary {

}


func (this *MagicDictionary) BuildDict(dictionary []string) {

}


func (this *MagicDictionary) Search(searchWord string) bool {

}


/**
* Your MagicDictionary object will be instantiated and called as such:
* obj := Constructor();
* obj.BuildDict(dictionary);
* param_2 := obj.Search(searchWord);
*/
```

Kotlin Solution:

```
class MagicDictionary() {

fun buildDict(dictionary: Array<String>) {

}

fun search(searchWord: String): Boolean {

}

}

/**
* Your MagicDictionary object will be instantiated and called as such:
* var obj = MagicDictionary()
* obj.buildDict(dictionary)
* var param_2 = obj.search(searchWord)
```

```
*/
```

Swift Solution:

```
class MagicDictionary {

init() {

}

func buildDict(_ dictionary: [String]) {

}

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

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* let obj = MagicDictionary()
* obj.buildDict(dictionary)
* let ret_2: Bool = obj.search(searchWord)
*/
```

Rust Solution:

```
// Problem: Implement Magic Dictionary
// Difficulty: Medium
// 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 MagicDictionary {

}
```

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

fn new() -> Self {

}

fn build_dict(&self, dictionary: Vec<String>) {

}

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

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* let obj = MagicDictionary::new();
* obj.build_dict(dictionary);
* let ret_2: bool = obj.search(searchWord);
*/
```

Ruby Solution:

```
class MagicDictionary
def initialize()

end



=begin
:type dictionary: String[]
:rtype: Void
=end
def build_dict(dictionary)
```

```
end



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

end



end

# Your MagicDictionary object will be instantiated and called as such:
# obj = MagicDictionary.new()
# obj.build_dict(dictionary)
# param_2 = obj.search(search_word)
```

PHP Solution:

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

}

/**
* @param String[] $dictionary
* @return NULL
*/
function buildDict($dictionary) {

}

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

/**
* Your MagicDictionary object will be instantiated and called as such:
* $obj = MagicDictionary();
* $obj->buildDict($dictionary);
* $ret_2 = $obj->search($searchWord);
*/
```

Dart Solution:

```
class MagicDictionary {

MagicDictionary() {

}

void buildDict(List<String> dictionary) {

}

bool search(String searchWord) {

}
}

/**
* Your MagicDictionary object will be instantiated and called as such:
* MagicDictionary obj = MagicDictionary();
* obj.buildDict(dictionary);
* bool param2 = obj.search(searchWord);
*/
```

Scala Solution:

```
class MagicDictionary() {

def buildDict(dictionary: Array[String]): Unit = {
}

def search(searchWord: String): Boolean = {

}

}

/**
* Your MagicDictionary object will be instantiated and called as such:
* val obj = new MagicDictionary()
* obj.buildDict(dictionary)
* val param_2 = obj.search(searchWord)
*/
```

Elixir Solution:

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

end

@spec build_dict(dictionary :: [String.t]) :: any
def build_dict(dictionary) do

end

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

end
end

# Your functions will be called as such:
# MagicDictionary.init_()
# MagicDictionary.build_dict(dictionary)
# param_2 = MagicDictionary.search(search_word)

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

Erlang Solution:

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

-spec magic_dictionary_build_dict(Dictionary :: [unicode:unicode_binary()])
-> any().
magic_dictionary_build_dict(Dictionary) ->
.

-spec magic_dictionary_search(SearchWord :: unicode:unicode_binary()) ->
boolean().
magic_dictionary_search(SearchWord) ->
.



%% Your functions will be called as such:
%% magic_dictionary_init_(),
%% magic_dictionary_build_dict(Dictionary),
%% Param_2 = magic_dictionary_search(SearchWord),

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

Racket Solution:

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

(init-field)

; build-dict : (listof string?) -> void?
(define/public (build-dict dictionary)
)
; search : string? -> boolean?
(define/public (search search-word)
)))

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