Problem 1032: Stream of Characters

Problem Information

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

Problem Description

Design an algorithm that accepts a stream of characters and checks if a suffix of these characters is a string of a given array of strings

words

.

For example, if

words = ["abc", "xyz"]

and the stream added the four characters (one by one)

'a'

,

'x'

,

'y'

, and

'z'

, your algorithm should detect that the suffix

"xyz"

of the characters

"axyz"

matches

"xyz"

from

words

.

Implement the

StreamChecker

class:

StreamChecker(String[] words)

Initializes the object with the strings array

words

.

boolean query(char letter)

Accepts a new character from the stream and returns

true

if any non-empty suffix from the stream forms a word that is in

words

.

Example 1:

Input

["StreamChecker", "query", "query", "query", "query", "query", "query", "query", "query", "query", "query", "query", "query"] [[["cd", "f", "kl"]], ["a"], ["b"], ["c"], ["d"], ["e"], ["f"], ["g"], ["h"], ["i"], ["j"], ["k"], ["l"]]

Output

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

Explanation

StreamChecker streamChecker = new StreamChecker(["cd", "f", "kl"]); streamChecker.query("a"); // return False streamChecker.query("b"); // return False streamChecker.query("c"); // return False streamChecker.query("d"); // return True, because 'cd' is in the wordlist streamChecker.query("e"); // return False streamChecker.query("f"); // return True, because 'f' is in the wordlist streamChecker.query("g"); // return False streamChecker.query("h"); // return False streamChecker.query("i"); // return False streamChecker.query("j"); // return False streamChecker.query("k"); // return False streamChecker.query("l"); // return True, because 'kl' is in the wordlist

Constraints:

1 <= words.length <= 2000

1 <= words[i].length <= 200

words[i]

consists of lowercase English letters.

letter

is a lowercase English letter.

At most

4 * 10

4

calls will be made to query.

Code Snippets

C++:

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

}

bool query(char letter) {

}
};

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker* obj = new StreamChecker(words);
* bool param_1 = obj->query(letter);
*/
```

Java:

```
class StreamChecker {

public StreamChecker(String[] words) {

}

public boolean query(char letter) {
}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker obj = new StreamChecker(words);
* boolean param_1 = obj.query(letter);
*/
```

Python3:

```
class StreamChecker:

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


def query(self, letter: str) -> bool:



# Your StreamChecker object will be instantiated and called as such:
# obj = StreamChecker(words)
# param_1 = obj.query(letter)
```

Python:

```
class StreamChecker(object):

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


def query(self, letter):
"""
:type letter: str
:rtype: bool
"""
```

```
# Your StreamChecker object will be instantiated and called as such:
# obj = StreamChecker(words)
# param_1 = obj.query(letter)
```

JavaScript:

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

};

/**
* @param {character} letter
* @return {boolean}
*/
StreamChecker.prototype.query = function(letter) {

};

/**
* Your StreamChecker object will be instantiated and called as such:
* var obj = new StreamChecker(words)
* var param_1 = obj.query(letter)
*/
```

TypeScript:

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

}

query(letter: string): boolean {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
```

```
* var obj = new StreamChecker(words)
* var param_1 = obj.query(letter)
*/
```

C#:

```
public class StreamChecker {

public StreamChecker(string[] words) {

}

public bool Query(char letter) {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker obj = new StreamChecker(words);
* bool param_1 = obj.Query(letter);
*/
```

C:

```



typedef struct {

} StreamChecker;


StreamChecker* streamCheckerCreate(char** words, int wordsSize) {

}

bool streamCheckerQuery(StreamChecker* obj, char letter) {

}

void streamCheckerFree(StreamChecker* obj) {


}

/**
* Your StreamChecker struct will be instantiated and called as such:
* StreamChecker* obj = streamCheckerCreate(words, wordsSize);
* bool param_1 = streamCheckerQuery(obj, letter);

* streamCheckerFree(obj);
*/
```

Go:

```
type StreamChecker struct {

}


func Constructor(words []string) StreamChecker {

}


func (this *StreamChecker) Query(letter byte) bool {

}


/**
* Your StreamChecker object will be instantiated and called as such:
* obj := Constructor(words);
* param_1 := obj.Query(letter);
*/
```

Kotlin:

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

fun query(letter: Char): Boolean {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* var obj = StreamChecker(words)
* var param_1 = obj.query(letter)
*/
```

Swift:

```
class StreamChecker {

init(_ words: [String]) {

}

func query(_ letter: Character) -> Bool {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* let obj = StreamChecker(words)
* let ret_1: Bool = obj.query(letter)
*/
```

Rust:

```
struct StreamChecker {

}



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

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

fn query(&self, letter: char) -> bool {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* let obj = StreamChecker::new(words);
* let ret_1: bool = obj.query(letter);
*/
```

Ruby:

```
class StreamChecker

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

end


=begin
:type letter: Character
:rtype: Boolean
=end
def query(letter)

end


end

# Your StreamChecker object will be instantiated and called as such:
# obj = StreamChecker.new(words)
# param_1 = obj.query(letter)
```

PHP:

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

}

/**
* @param String $letter
* @return Boolean
*/
function query($letter) {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* $obj = StreamChecker($words);
* $ret_1 = $obj->query($letter);
*/
```

Dart:

```
class StreamChecker {

StreamChecker(List<String> words) {

}

bool query(String letter) {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker obj = StreamChecker(words);
* bool param1 = obj.query(letter);
*/
```

Scala:

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

def query(letter: Char): Boolean = {

}

}

/**
* Your StreamChecker object will be instantiated and called as such:
* val obj = new StreamChecker(words)
* val param_1 = obj.query(letter)
*/
```

Elixir:

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

end

@spec query(letter :: char) :: boolean
def query(letter) do

end
end

# Your functions will be called as such:
# StreamChecker.init_(words)
# param_1 = StreamChecker.query(letter)

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

Erlang:

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

```
-spec stream_checker_query(Letter :: char()) -> boolean().
stream_checker_query(Letter) ->
.



%% Your functions will be called as such:
%% stream_checker_init_(Words),
%% Param_1 = stream_checker_query(Letter),

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

Racket:

```
(define stream-checker%
(class object%
(super-new)

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

; query : char? -> boolean?
(define/public (query letter)
)))

;; Your stream-checker% object will be instantiated and called as such:
;; (define obj (new stream-checker% [words words]))
;; (define param_1 (send obj query letter))
```

Solutions

C++ Solution:

```
/*
* Problem: Stream of Characters
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}

bool query(char letter) {

}
};

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker* obj = new StreamChecker(words);
* bool param_1 = obj->query(letter);
*/
```

Java Solution:

```
/**
* Problem: Stream of Characters
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class StreamChecker {

public StreamChecker(String[] words) {

}

public boolean query(char letter) {
}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker obj = new StreamChecker(words);
* boolean param_1 = obj.query(letter);
*/
```

Python3 Solution:

```
"""
Problem: Stream of Characters
Difficulty: Hard
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class StreamChecker:

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



def query(self, letter: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class StreamChecker(object):

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



def query(self, letter):
```

```
"""
:type letter: str
:rtype: bool
"""



# Your StreamChecker object will be instantiated and called as such:
# obj = StreamChecker(words)
# param_1 = obj.query(letter)
```

JavaScript Solution:

```
/**
* Problem: Stream of Characters
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};

/**
* @param {character} letter
* @return {boolean}
*/
StreamChecker.prototype.query = function(letter) {

};

/**
* Your StreamChecker object will be instantiated and called as such:
* var obj = new StreamChecker(words)
* var param_1 = obj.query(letter)
*/
```

TypeScript Solution:

```
/**
* Problem: Stream of Characters
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}

query(letter: string): boolean {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* var obj = new StreamChecker(words)
* var param_1 = obj.query(letter)
*/
```

C# Solution:

```
/*
* Problem: Stream of Characters
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class StreamChecker {

public StreamChecker(string[] words) {

}

public bool Query(char letter) {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker obj = new StreamChecker(words);
* bool param_1 = obj.Query(letter);
*/
```

C Solution:

```
/*
* Problem: Stream of Characters
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} StreamChecker;


StreamChecker* streamCheckerCreate(char** words, int wordsSize) {
}

bool streamCheckerQuery(StreamChecker* obj, char letter) {

}

void streamCheckerFree(StreamChecker* obj) {

}

/**
* Your StreamChecker struct will be instantiated and called as such:
* StreamChecker* obj = streamCheckerCreate(words, wordsSize);
* bool param_1 = streamCheckerQuery(obj, letter);

* streamCheckerFree(obj);
*/
```

Go Solution:

```
// Problem: Stream of Characters
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type StreamChecker struct {

}



func Constructor(words []string) StreamChecker {

}



func (this *StreamChecker) Query(letter byte) bool {

}
```

```
/**
* Your StreamChecker object will be instantiated and called as such:
* obj := Constructor(words);
* param_1 := obj.Query(letter);
*/
```

Kotlin Solution:

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

fun query(letter: Char): Boolean {

}

}

/**
* Your StreamChecker object will be instantiated and called as such:
* var obj = StreamChecker(words)
* var param_1 = obj.query(letter)
*/
```

Swift Solution:

```
class StreamChecker {

init(_ words: [String]) {

}

func query(_ letter: Character) -> Bool {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* let obj = StreamChecker(words)
* let ret_1: Bool = obj.query(letter)
*/
```

Rust Solution:

```
// Problem: Stream of Characters
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct StreamChecker {

}


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

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

}

fn query(&self, letter: char) -> bool {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* let obj = StreamChecker::new(words);
* let ret_1: bool = obj.query(letter);
*/
```

Ruby Solution:

```
class StreamChecker

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

end


=begin
:type letter: Character
:rtype: Boolean
=end
def query(letter)

end


end

# Your StreamChecker object will be instantiated and called as such:
# obj = StreamChecker.new(words)
# param_1 = obj.query(letter)
```

PHP Solution:

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

}

/**
* @param String $letter
* @return Boolean
*/
function query($letter) {

}

}

/**
* Your StreamChecker object will be instantiated and called as such:
* $obj = StreamChecker($words);
* $ret_1 = $obj->query($letter);
*/
```

Dart Solution:

```
class StreamChecker {

StreamChecker(List<String> words) {

}

bool query(String letter) {

}
}

/**
* Your StreamChecker object will be instantiated and called as such:
* StreamChecker obj = StreamChecker(words);
* bool param1 = obj.query(letter);
*/
```

Scala Solution:

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

def query(letter: Char): Boolean = {

}

}

/**
* Your StreamChecker object will be instantiated and called as such:
* val obj = new StreamChecker(words)
* val param_1 = obj.query(letter)
```

```
*/
```

Elixir Solution:

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

end

@spec query(letter :: char) :: boolean
def query(letter) do

end
end

# Your functions will be called as such:
# StreamChecker.init_(words)
# param_1 = StreamChecker.query(letter)

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

Erlang Solution:

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

-spec stream_checker_query(Letter :: char()) -> boolean().
stream_checker_query(Letter) ->
.



%% Your functions will be called as such:
%% stream_checker_init_(Words),
%% Param_1 = stream_checker_query(Letter),

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

Racket Solution:

```
(define stream-checker%
(class object%
(super-new)

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

; query : char? -> boolean?
(define/public (query letter)
)))

;; Your stream-checker% object will be instantiated and called as such:
;; (define obj (new stream-checker% [words words]))
;; (define param_1 (send obj query letter))
```