# Problem 1023: Camelcase Matching

## Problem Information

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

## Problem Description

Given an array of strings

queries

and a string

pattern

, return a boolean array

answer

where

answer[i]

is

true

if

queries[i]

matches

pattern

, and

false

otherwise.

A query word

queries[i]

matches

pattern

if you can insert lowercase English letters into the pattern so that it equals the query. You may insert a character at any position in pattern or you may choose not to insert any characters

at all

.

Example 1:

Input:

queries = ["FooBar","FooBarTest","FootBall","FrameBuffer","ForceFeedBack"], pattern = "FB"

Output:

[true,false,true,true,false]

Explanation:

"FooBar" can be generated like this "F" + "oo" + "B" + "ar". "FootBall" can be generated like this "F" + "oot" + "B" + "all". "FrameBuffer" can be generated like this "F" + "rame" + "B" + "uffer".

Example 2:

Input:

queries = ["FooBar","FooBarTest","FootBall","FrameBuffer","ForceFeedBack"], pattern = "FoBa"

Output:

[true,false,true,false,false]

Explanation:

"FooBar" can be generated like this "Fo" + "o" + "Ba" + "r". "FootBall" can be generated like this "Fo" + "ot" + "Ba" + "ll".

Example 3:

Input:

queries = ["FooBar","FooBarTest","FootBall","FrameBuffer","ForceFeedBack"], pattern = "FoBaT"

Output:

[false,true,false,false,false]

Explanation:

"FooBarTest" can be generated like this "Fo" + "o" + "Ba" + "r" + "T" + "est".

Constraints:

1 <= pattern.length, queries.length <= 100

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

queries[i]

and

pattern

consist of English letters.

## Code Snippets

**C++:**

```
class Solution {
public:
vector<bool> camelMatch(vector<string>& queries, string pattern) {

}
};
```

**Java:**

```
class Solution {
public List<Boolean> camelMatch(String[] queries, String pattern) {

}
}
```

**Python3:**

```
class Solution:
def camelMatch(self, queries: List[str], pattern: str) -> List[bool]:
```

**Python:**

```
class Solution(object):
def camelMatch(self, queries, pattern):
"""
:type queries: List[str]
:type pattern: str
:rtype: List[bool]
"""
```

**JavaScript:**

```
/**
* @param {string[]} queries
* @param {string} pattern
* @return {boolean[]}
*/
var camelMatch = function(queries, pattern) {

};
```

**TypeScript:**

```
function camelMatch(queries: string[], pattern: string): boolean[] {

};
```

**C#:**

```
public class Solution {
public IList<bool> CamelMatch(string[] queries, string pattern) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* camelMatch(char** queries, int queriesSize, char* pattern, int*
returnSize) {

}
```

**Go:**

```
func camelMatch(queries []string, pattern string) []bool {

}
```

**Kotlin:**

```
class Solution {
fun camelMatch(queries: Array<String>, pattern: String): List<Boolean> {
```

```
}
}
```

**Swift:**

```
class Solution {
func camelMatch(_ queries: [String], _ pattern: String) -> [Bool] {

}
}
```

**Rust:**

```
impl Solution {
pub fn camel_match(queries: Vec<String>, pattern: String) -> Vec<bool> {

}
}
```

**Ruby:**

```
# @param {String[]} queries
# @param {String} pattern
# @return {Boolean[]}
def camel_match(queries, pattern)

end
```

**PHP:**

```
class Solution {

/**
* @param String[] $queries
* @param String $pattern
* @return Boolean[]
*/
function camelMatch($queries, $pattern) {

}
}
```

**Dart:**

```
class Solution {
List<bool> camelMatch(List<String> queries, String pattern) {

}
}
```

**Scala:**

```
object Solution {
def camelMatch(queries: Array[String], pattern: String): List[Boolean] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec camel_match(queries :: [String.t], pattern :: String.t) :: [boolean]
def camel_match(queries, pattern) do

end
end
```

**Erlang:**

```
-spec camel_match(Queries :: [unicode:unicode_binary()], Pattern ::
unicode:unicode_binary()) -> [boolean()].
camel_match(Queries, Pattern) ->
.
```

**Racket:**

```
(define/contract (camel-match queries pattern)
(-> (listof string?) string? (listof boolean?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Camelcase Matching
* Difficulty: Medium
* 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 Solution {
public:
vector<bool> camelMatch(vector<string>& queries, string pattern) {

}
};
```

### Java Solution:

```
/**
* Problem: Camelcase Matching
* Difficulty: Medium
* 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 Solution {
public List<Boolean> camelMatch(String[] queries, String pattern) {

}
}
```

### Python3 Solution:

```
"""
Problem: Camelcase Matching
Difficulty: Medium
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 Solution:
def camelMatch(self, queries: List[str], pattern: str) -> List[bool]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def camelMatch(self, queries, pattern):
"""
:type queries: List[str]
:type pattern: str
:rtype: List[bool]
"""
```

### JavaScript Solution:

```
/**
* Problem: Camelcase Matching
* Difficulty: Medium
* 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[]} queries
* @param {string} pattern
* @return {boolean[]}
*/
var camelMatch = function(queries, pattern) {

};
```

### TypeScript Solution:

```
/**
* Problem: Camelcase Matching
* Difficulty: Medium
* 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
*/

function camelMatch(queries: string[], pattern: string): boolean[] {

};
```

### C# Solution:

```
/*
* Problem: Camelcase Matching
* Difficulty: Medium
* 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 Solution {
public IList<bool> CamelMatch(string[] queries, string pattern) {

}
}
```

### C Solution:

```
/*
* Problem: Camelcase Matching
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* camelMatch(char** queries, int queriesSize, char* pattern, int*
returnSize) {

}
```

### Go Solution:

```
// Problem: Camelcase Matching
// Difficulty: Medium
// 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

func camelMatch(queries []string, pattern string) []bool {

}
```

### Kotlin Solution:

```
class Solution {
fun camelMatch(queries: Array<String>, pattern: String): List<Boolean> {

}
}
```

### Swift Solution:

```
class Solution {
func camelMatch(_ queries: [String], _ pattern: String) -> [Bool] {

}
}
```

### Rust Solution:

```
// Problem: Camelcase Matching
// Difficulty: Medium
// 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

impl Solution {
pub fn camel_match(queries: Vec<String>, pattern: String) -> Vec<bool> {

}
}
```

### Ruby Solution:

```
# @param {String[]} queries
# @param {String} pattern
# @return {Boolean[]}
def camel_match(queries, pattern)

end
```

### PHP Solution:

```
class Solution {

/**
* @param String[] $queries
* @param String $pattern
* @return Boolean[]
*/
function camelMatch($queries, $pattern) {

}
}
```

### Dart Solution:

```
class Solution {
List<bool> camelMatch(List<String> queries, String pattern) {

}
}
```

### Scala Solution:

```
object Solution {
def camelMatch(queries: Array[String], pattern: String): List[Boolean] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec camel_match(queries :: [String.t], pattern :: String.t) :: [boolean]
def camel_match(queries, pattern) do

end
end
```

### Erlang Solution:

```
-spec camel_match(Queries :: [unicode:unicode_binary()], Pattern ::
unicode:unicode_binary()) -> [boolean()].
camel_match(Queries, Pattern) ->
.
```

### Racket Solution:

```
(define/contract (camel-match queries pattern)
(-> (listof string?) string? (listof boolean?))
)
```