# Problem 819: Most Common Word

## Problem Information

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

## Problem Description

Given a string

paragraph

and a string array of the banned words

banned

, return

the most frequent word that is not banned

. It is

guaranteed

there is

at least one word

that is not banned, and that the answer is

unique

.

The words in

paragraph

are

case-insensitive

and the answer should be returned in

lowercase

.

Note

that words can not contain punctuation symbols.

Example 1:

Input:

paragraph = "Bob hit a ball, the hit BALL flew far after it was hit.", banned = ["hit"]

Output:

"ball"

Explanation:

"hit" occurs 3 times, but it is a banned word. "ball" occurs twice (and no other word does), so it is the most frequent non-banned word in the paragraph. Note that words in the paragraph are not case sensitive, that punctuation is ignored (even if adjacent to words, such as "ball,"), and that "hit" isn't the answer even though it occurs more because it is banned.

Example 2:

Input:

paragraph = "a.", banned = []

Output:

"a"

Constraints:

1 <= paragraph.length <= 1000

paragraph consists of English letters, space

' '

, or one of the symbols:

"!?',;."

.

0 <= banned.length <= 100

1 <= banned[i].length <= 10

banned[i]

consists of only lowercase English letters.

## Code Snippets

**C++:**

```
class Solution {
public:
string mostCommonWord(string paragraph, vector<string>& banned) {

}
};
```

**Java:**

```
class Solution {
public String mostCommonWord(String paragraph, String[] banned) {

}
}
```

**Python3:**

```
class Solution:
def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
```

**Python:**

```
class Solution(object):
def mostCommonWord(self, paragraph, banned):
"""
:type paragraph: str
:type banned: List[str]
:rtype: str
"""
```

**JavaScript:**

```
/**
* @param {string} paragraph
* @param {string[]} banned
* @return {string}
*/
var mostCommonWord = function(paragraph, banned) {

};
```

**TypeScript:**

```
function mostCommonWord(paragraph: string, banned: string[]): string {

};
```

**C#:**

```
public class Solution {
public string MostCommonWord(string paragraph, string[] banned) {

}
}
```

**C:**

```
char* mostCommonWord(char* paragraph, char** banned, int bannedSize) {

}
```

**Go:**

```
func mostCommonWord(paragraph string, banned []string) string {

}
```

**Kotlin:**

```
class Solution {
fun mostCommonWord(paragraph: String, banned: Array<String>): String {

}
}
```

**Swift:**

```
class Solution {
func mostCommonWord(_ paragraph: String, _ banned: [String]) -> String {

}
}
```

**Rust:**

```
impl Solution {
pub fn most_common_word(paragraph: String, banned: Vec<String>) -> String {

}
}
```

**Ruby:**

```
# @param {String} paragraph
# @param {String[]} banned
# @return {String}
def most_common_word(paragraph, banned)

end
```

**PHP:**

```
class Solution {

/**
* @param String $paragraph
* @param String[] $banned
* @return String
*/
function mostCommonWord($paragraph, $banned) {

}
}
```

**Dart:**

```
class Solution {
String mostCommonWord(String paragraph, List<String> banned) {

}
}
```

**Scala:**

```
object Solution {
def mostCommonWord(paragraph: String, banned: Array[String]): String = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec most_common_word(paragraph :: String.t, banned :: [String.t]) ::
String.t
def most_common_word(paragraph, banned) do
```

```
end
end
```

**Erlang:**

```
-spec most_common_word(Paragraph :: unicode:unicode_binary(), Banned ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
most_common_word(Paragraph, Banned) ->
.
```

**Racket:**

```
(define/contract (most-common-word paragraph banned)
(-> string? (listof string?) string?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Most Common Word
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* 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 Solution {
public:
string mostCommonWord(string paragraph, vector<string>& banned) {

}
};
```

### Java Solution:

```
/**
* Problem: Most Common Word
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* 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 Solution {
public String mostCommonWord(String paragraph, String[] banned) {

}
}
```

### Python3 Solution:

```
"""
Problem: Most Common Word
Difficulty: Easy
Tags: array, string, graph, hash

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 Solution:
def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def mostCommonWord(self, paragraph, banned):
"""
:type paragraph: str
:type banned: List[str]
:rtype: str
"""
```

### JavaScript Solution:

```
/**
* Problem: Most Common Word
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* 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} paragraph
* @param {string[]} banned
* @return {string}
*/
var mostCommonWord = function(paragraph, banned) {

};
```

### TypeScript Solution:

```
/**
* Problem: Most Common Word
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function mostCommonWord(paragraph: string, banned: string[]): string {

};
```

### C# Solution:

```
/*
* Problem: Most Common Word
* Difficulty: Easy
```

```
* Tags: array, string, graph, hash
*
* 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 Solution {
public string MostCommonWord(string paragraph, string[] banned) {

}
}
```

### C Solution:

```
/*
* Problem: Most Common Word
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* mostCommonWord(char* paragraph, char** banned, int bannedSize) {

}
```

### Go Solution:

```
// Problem: Most Common Word
// Difficulty: Easy
// Tags: array, string, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func mostCommonWord(paragraph string, banned []string) string {
```

```
}
```

### Kotlin Solution:

```
class Solution {
fun mostCommonWord(paragraph: String, banned: Array<String>): String {

}
}
```

### Swift Solution:

```
class Solution {
func mostCommonWord(_ paragraph: String, _ banned: [String]) -> String {

}
}
```

### Rust Solution:

```
// Problem: Most Common Word
// Difficulty: Easy
// Tags: array, string, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn most_common_word(paragraph: String, banned: Vec<String>) -> String {

}
}
```

### Ruby Solution:

```
# @param {String} paragraph
# @param {String[]} banned
# @return {String}
def most_common_word(paragraph, banned)
```

```
end
```

### PHP Solution:

```
class Solution {

/**
* @param String $paragraph
* @param String[] $banned
* @return String
*/
function mostCommonWord($paragraph, $banned) {

}
}
```

### Dart Solution:

```
class Solution {
String mostCommonWord(String paragraph, List<String> banned) {

}
}
```

### Scala Solution:

```
object Solution {
def mostCommonWord(paragraph: String, banned: Array[String]): String = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec most_common_word(paragraph :: String.t, banned :: [String.t]) ::
String.t
def most_common_word(paragraph, banned) do

end
end
```

### Erlang Solution:

```
-spec most_common_word(Paragraph :: unicode:unicode_binary(), Banned ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
most_common_word(Paragraph, Banned) ->
.
```

### Racket Solution:

```
(define/contract (most-common-word paragraph banned)
(-> string? (listof string?) string?)
)
```