Problem 1255: Maximum Score Words Formed by Letters

Problem Information

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

Problem Description

Given a list of

words

, list of single

letters

(might be repeating) and

score

of every character.

Return the maximum score of

any

valid set of words formed by using the given letters (

words[i]

cannot be used two or more times).

It is not necessary to use all characters in

letters

and each letter can only be used once. Score of letters

'a'

,

'b'

,

'c'

, ... ,

'z'

is given by

score[0]

,

score[1]

, ... ,

score[25]

respectively.

Example 1:

Input:

words = ["dog","cat","dad","good"], letters = ["a","a","c","d","d","d","g","o","o"], score = [1,0,9,5,0,0,3,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0]

Output:

23

Explanation:

Score a=1, c=9, d=5, g=3, o=2 Given letters, we can form the words "dad" (5+1+5) and "good" (3+2+2+5) with a score of 23. Words "dad" and "dog" only get a score of 21.

Example 2:

Input:

words = ["xxxz","ax","bx","cx"], letters = ["z","a","b","c","x","x","x"], score = [4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,10]

Output:

27

Explanation:

Score a=4, b=4, c=4, x=5, z=10 Given letters, we can form the words "ax" (4+5), "bx" (4+5) and "cx" (4+5) with a score of 27. Word "xxxz" only get a score of 25.

Example 3:

Input:

words = ["leetcode"], letters = ["l","e","t","c","o","d"], score = [0,0,1,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0]

Output:

0

Explanation:

Letter "e" can only be used once.

Constraints:

1 <= words.length <= 14

1 <= words[i].length <= 15

1 <= letters.length <= 100

letters[i].length == 1

score.length == 26

0 <= score[i] <= 10

words[i]

,

letters[i]

contains only lower case English letters.

Code Snippets

C++:

```
class Solution {
public:
int maxScoreWords(vector<string>& words, vector<char>& letters, vector<int>&
score) {

}
};
```

Java:

```
class Solution {
public int maxScoreWords(String[] words, char[] letters, int[] score) {
```

```
}
}
```

Python3:

```
class Solution:
def maxScoreWords(self, words: List[str], letters: List[str], score:
List[int]) -> int:
```

Python:

```
class Solution(object):
def maxScoreWords(self, words, letters, score):
"""
:type words: List[str]
:type letters: List[str]
:type score: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {character[]} letters
* @param {number[]} score
* @return {number}
*/
var maxScoreWords = function(words, letters, score) {

};
```

TypeScript:

```
function maxScoreWords(words: string[], letters: string[], score: number[]):
number {

};
```

C#:

```
public class Solution {
public int MaxScoreWords(string[] words, char[] letters, int[] score) {

}
}
```

C:

```
int maxScoreWords(char** words, int wordsSize, char* letters, int
lettersSize, int* score, int scoreSize) {

}
```

Go:

```
func maxScoreWords(words []string, letters []byte, score []int) int {

}
```

Kotlin:

```
class Solution {
fun maxScoreWords(words: Array<String>, letters: CharArray, score: IntArray):
Int {

}
}
```

Swift:

```
class Solution {
func maxScoreWords(_ words: [String], _ letters: [Character], _ score: [Int])
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_score_words(words: Vec<String>, letters: Vec<char>, score:
Vec<i32>) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {String[]} words
# @param {Character[]} letters
# @param {Integer[]} score
# @return {Integer}
def max_score_words(words, letters, score)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param String[] $letters
* @param Integer[] $score
* @return Integer
*/
function maxScoreWords($words, $letters, $score) {

}
}
```

Dart:

```
class Solution {
int maxScoreWords(List<String> words, List<String> letters, List<int> score)
{

}
}
```

Scala:

```
object Solution {
def maxScoreWords(words: Array[String], letters: Array[Char], score:
Array[Int]): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec max_score_words(words :: [String.t], letters :: [char], score ::
[integer]) :: integer
def max_score_words(words, letters, score) do

end
end
```

Erlang:

```
-spec max_score_words(Words :: [unicode:unicode_binary()], Letters ::
[char()], Score :: [integer()]) -> integer().
max_score_words(Words, Letters, Score) ->
.
```

Racket:

```
(define/contract (max-score-words words letters score)
(-> (listof string?) (listof char?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score Words Formed by Letters
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int maxScoreWords(vector<string>& words, vector<char>& letters, vector<int>&
score) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score Words Formed by Letters
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxScoreWords(String[] words, char[] letters, int[] score) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score Words Formed by Letters
Difficulty: Hard
Tags: array, string, dp, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxScoreWords(self, words: List[str], letters: List[str], score:
```

```
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxScoreWords(self, words, letters, score):
"""
:type words: List[str]
:type letters: List[str]
:type score: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Score Words Formed by Letters
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string[]} words
* @param {character[]} letters
* @param {number[]} score
* @return {number}
*/
var maxScoreWords = function(words, letters, score) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score Words Formed by Letters
* Difficulty: Hard
```

```
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxScoreWords(words: string[], letters: string[], score: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score Words Formed by Letters
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxScoreWords(string[] words, char[] letters, int[] score) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score Words Formed by Letters
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
int maxScoreWords(char** words, int wordsSize, char* letters, int
lettersSize, int* score, int scoreSize) {

}
```

Go Solution:

```
// Problem: Maximum Score Words Formed by Letters
// Difficulty: Hard
// Tags: array, string, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxScoreWords(words []string, letters []byte, score []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxScoreWords(words: Array<String>, letters: CharArray, score: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func maxScoreWords(_ words: [String], _ letters: [Character], _ score: [Int])
-> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Score Words Formed by Letters
// Difficulty: Hard
// Tags: array, string, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_score_words(words: Vec<String>, letters: Vec<char>, score:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {Character[]} letters
# @param {Integer[]} score
# @return {Integer}
def max_score_words(words, letters, score)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String[] $letters
* @param Integer[] $score
* @return Integer
*/
function maxScoreWords($words, $letters, $score) {

}
}
```

Dart Solution:

```
class Solution {
int maxScoreWords(List<String> words, List<String> letters, List<int> score)
{

}
}
```

Scala Solution:

```
object Solution {
def maxScoreWords(words: Array[String], letters: Array[Char], score:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_score_words(words :: [String.t], letters :: [char], score ::
[integer]) :: integer
def max_score_words(words, letters, score) do

end
end
```

Erlang Solution:

```
-spec max_score_words(Words :: [unicode:unicode_binary()], Letters ::
[char()], Score :: [integer()]) -> integer().
max_score_words(Words, Letters, Score) ->
.
```

Racket Solution:

```
(define/contract (max-score-words words letters score)
(-> (listof string?) (listof char?) (listof exact-integer?) exact-integer?)
)
```