Problem 692: Top K Frequent Words

Problem Information

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

Problem Description

Given an array of strings

words

and an integer

k

, return

the

k

most frequent strings

.

Return the answer

sorted

by

the frequency

from highest to lowest. Sort the words with the same frequency by their

lexicographical order

.

Example 1:

Input:

words = ["i","love","leetcode","i","love","coding"], k = 2

Output:

["i","love"]

Explanation:

"i" and "love" are the two most frequent words. Note that "i" comes before "love" due to a lower alphabetical order.

Example 2:

Input:

words = ["the","day","is","sunny","the","the","the","sunny","is","is"], k = 4

Output:

["the","is","sunny","day"]

Explanation:

"the", "is", "sunny" and "day" are the four most frequent words, with the number of occurrence being 4, 3, 2 and 1 respectively.

Constraints:

1 <= words.length <= 500

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

words[i]

consists of lowercase English letters.

k

is in the range

[1, The number of

unique

words[i]]

Follow-up:

Could you solve it in

O(n log(k))

time and

O(n)

extra space?

Code Snippets

C++:

```
class Solution {
public:
vector<string> topKFrequent(vector<string>& words, int k) {

}
};
```

Java:

```
class Solution {
public List<String> topKFrequent(String[] words, int k) {

}
}
```

Python3:

```
class Solution:
def topKFrequent(self, words: List[str], k: int) -> List[str]:
```

Python:

```
class Solution(object):
def topKFrequent(self, words, k):
"""
:type words: List[str]
:type k: int
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {number} k
* @return {string[]}
*/
var topKFrequent = function(words, k) {

};
```

TypeScript:

```
function topKFrequent(words: string[], k: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> TopKFrequent(string[] words, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** topKFrequent(char** words, int wordsSize, int k, int* returnSize) {

}
```

Go:

```
func topKFrequent(words []string, k int) []string {

}
```

Kotlin:

```
class Solution {
fun topKFrequent(words: Array<String>, k: Int): List<String> {

}
}
```

Swift:

```
class Solution {
func topKFrequent(_ words: [String], _ k: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn top_k_frequent(words: Vec<String>, k: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {Integer} k
# @return {String[]}
def top_k_frequent(words, k)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param Integer $k
* @return String[]
*/
function topKFrequent($words, $k) {

}
}
```

Dart:

```
class Solution {
List<String> topKFrequent(List<String> words, int k) {

}
}
```

Scala:

```
object Solution {
def topKFrequent(words: Array[String], k: Int): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec top_k_frequent(words :: [String.t], k :: integer) :: [String.t]
def top_k_frequent(words, k) do

end
end
```

Erlang:

```
-spec top_k_frequent(Words :: [unicode:unicode_binary()], K :: integer()) ->
[unicode:unicode_binary()].
top_k_frequent(Words, K) ->
.
```

Racket:

```
(define/contract (top-k-frequent words k)
(-> (listof string?) exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Top K Frequent Words
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, queue, heap
*
* 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:
vector<string> topKFrequent(vector<string>& words, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Top K Frequent Words
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, queue, heap
*
* 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 List<String> topKFrequent(String[] words, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Top K Frequent Words
Difficulty: Medium
Tags: array, string, graph, hash, sort, queue, heap

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 topKFrequent(self, words: List[str], k: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def topKFrequent(self, words, k):
"""
:type words: List[str]
:type k: int
:rtype: List[str]
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Top K Frequent Words
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, queue, heap
*
* 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[]} words
* @param {number} k
* @return {string[]}
*/
var topKFrequent = function(words, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Top K Frequent Words
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, queue, heap
*
* 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 topKFrequent(words: string[], k: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Top K Frequent Words
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, queue, heap
*
* 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 IList<string> TopKFrequent(string[] words, int k) {

}
}
```

C Solution:

```
/*
* Problem: Top K Frequent Words
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** topKFrequent(char** words, int wordsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Top K Frequent Words
// Difficulty: Medium
// Tags: array, string, graph, hash, sort, queue, heap
//
// 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 topKFrequent(words []string, k int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun topKFrequent(words: Array<String>, k: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func topKFrequent(_ words: [String], _ k: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Top K Frequent Words
// Difficulty: Medium
// Tags: array, string, graph, hash, sort, queue, heap
//
// 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 top_k_frequent(words: Vec<String>, k: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {Integer} k
# @return {String[]}
def top_k_frequent(words, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param Integer $k
* @return String[]
*/
function topKFrequent($words, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<String> topKFrequent(List<String> words, int k) {

}
}
```

Scala Solution:

```
object Solution {
def topKFrequent(words: Array[String], k: Int): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec top_k_frequent(words :: [String.t], k :: integer) :: [String.t]
def top_k_frequent(words, k) do
```

```
end
end
```

Erlang Solution:

```
-spec top_k_frequent(Words :: [unicode:unicode_binary()], K :: integer()) ->
[unicode:unicode_binary()].
top_k_frequent(Words, K) ->
.
```

Racket Solution:

```
(define/contract (top-k-frequent words k)
(-> (listof string?) exact-integer? (listof string?))
)
```