# Problem 2456: Most Popular Video Creator

## Problem Information

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

## Problem Description

You are given two string arrays

creators

and

ids

, and an integer array

views

, all of length

n

. The

i

th

video on a platform was created by

creators[i]

, has an id of

ids[i]

, and has

views[i]

views.

The

popularity

of a creator is the

sum

of the number of views on

all

of the creator's videos. Find the creator with the

highest

popularity and the id of their

most

viewed video.

If multiple creators have the highest popularity, find all of them.

If multiple videos have the highest view count for a creator, find the lexicographically

smallest

id.

Note: It is possible for different videos to have the same

id

, meaning that

id

s do not uniquely identify a video. For example, two videos with the same ID are considered as distinct videos with their own viewcount.

Return

a

2D array

of

strings

answer

where

answer[i] = [creators

i

, id

i

]

means that

creators

i

has the

highest

popularity and

id

i

is the

id

of their most

popular

video. The answer can be returned in any order.

Example 1:

Input:

creators = ["alice","bob","alice","chris"], ids = ["one","two","three","four"], views = [5,10,5,4]

Output:

[["alice","one"],["bob","two"]]

Explanation:

The popularity of alice is 5 + 5 = 10.

The popularity of bob is 10.

The popularity of chris is 4.

alice and bob are the most popular creators.

For bob, the video with the highest view count is "two".

For alice, the videos with the highest view count are "one" and "three". Since "one" is lexicographically smaller than "three", it is included in the answer.

Example 2:

Input:

creators = ["alice","alice","alice"], ids = ["a","b","c"], views = [1,2,2]

Output:

[["alice","b"]]

Explanation:

The videos with id "b" and "c" have the highest view count.

Since "b" is lexicographically smaller than "c", it is included in the answer.

Constraints:

n == creators.length == ids.length == views.length

1 <= n <= 10

5

1 <= creators[i].length, ids[i].length <= 5

creators[i]

and

ids[i]

consist only of lowercase English letters.

0 <= views[i] <= 10

5

## Code Snippets

**C++:**

```
class Solution {
public:
vector<vector<string>> mostPopularCreator(vector<string>& creators,
vector<string>& ids, vector<int>& views) {

}
};
```

**Java:**

```
class Solution {
public List<List<String>> mostPopularCreator(String[] creators, String[] ids,
int[] views) {

}
}
```

**Python3:**

```
class Solution:
def mostPopularCreator(self, creators: List[str], ids: List[str], views:
List[int]) -> List[List[str]]:
```

**Python:**

```
class Solution(object):
def mostPopularCreator(self, creators, ids, views):
"""
:type creators: List[str]
```

```
:type ids: List[str]
:type views: List[int]
:rtype: List[List[str]]
"""
```

**JavaScript:**

```
/**
* @param {string[]} creators
* @param {string[]} ids
* @param {number[]} views
* @return {string[][]}
*/
var mostPopularCreator = function(creators, ids, views) {

};
```

**TypeScript:**

```
function mostPopularCreator(creators: string[], ids: string[], views:
number[]): string[][] {

};
```

**C#:**

```
public class Solution {
public IList<IList<string>> MostPopularCreator(string[] creators, string[]
ids, int[] views) {

}
}
```

**C:**

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** mostPopularCreator(char** creators, int creatorsSize, char** ids, int
```

```
idsSize, int* views, int viewsSize, int* returnSize, int** returnColumnSizes)
{

}
```

**Go:**

```
func mostPopularCreator(creators []string, ids []string, views []int)
[][]string {

}
```

**Kotlin:**

```
class Solution {
fun mostPopularCreator(creators: Array<String>, ids: Array<String>, views:
IntArray): List<List<String>> {

}
}
```

**Swift:**

```
class Solution {
func mostPopularCreator(_ creators: [String], _ ids: [String], _ views:
[Int]) -> [[String]] {

}
}
```

**Rust:**

```
impl Solution {
pub fn most_popular_creator(creators: Vec<String>, ids: Vec<String>, views:
Vec<i32>) -> Vec<Vec<String>> {

}
}
```

**Ruby:**

```
# @param {String[]} creators
# @param {String[]} ids
```

```
# @param {Integer[]} views
# @return {String[][]}
def most_popular_creator(creators, ids, views)

end
```

**PHP:**

```
class Solution {

/**
* @param String[] $creators
* @param String[] $ids
* @param Integer[] $views
* @return String[][]
*/
function mostPopularCreator($creators, $ids, $views) {

}
}
```

**Dart:**

```
class Solution {
List<List<String>> mostPopularCreator(List<String> creators, List<String>
ids, List<int> views) {

}
}
```

**Scala:**

```
object Solution {
def mostPopularCreator(creators: Array[String], ids: Array[String], views:
Array[Int]): List[List[String]] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec most_popular_creator(creators :: [String.t], ids :: [String.t], views
:: [integer]) :: [[String.t]]
def most_popular_creator(creators, ids, views) do

end
end
```

**Erlang:**

```
-spec most_popular_creator(Creators :: [unicode:unicode_binary()], Ids ::
[unicode:unicode_binary()], Views :: [integer()]) ->
[[unicode:unicode_binary()]].
most_popular_creator(Creators, Ids, Views) ->
.
```

**Racket:**

```
(define/contract (most-popular-creator creators ids views)
(-> (listof string?) (listof string?) (listof exact-integer?) (listof (listof
string?)))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Most Popular Video Creator
* 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<vector<string>> mostPopularCreator(vector<string>& creators,
vector<string>& ids, vector<int>& views) {
```

```
}
};
```

### Java Solution:

```
/**
* Problem: Most Popular Video Creator
* 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<List<String>> mostPopularCreator(String[] creators, String[] ids,
int[] views) {

}
}
```

### Python3 Solution:

```
"""
Problem: Most Popular Video Creator
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 mostPopularCreator(self, creators: List[str], ids: List[str], views:
List[int]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def mostPopularCreator(self, creators, ids, views):
"""
:type creators: List[str]
:type ids: List[str]
:type views: List[int]
:rtype: List[List[str]]
"""
```

### JavaScript Solution:

```
/**
* Problem: Most Popular Video Creator
* 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[]} creators
* @param {string[]} ids
* @param {number[]} views
* @return {string[][]}
*/
var mostPopularCreator = function(creators, ids, views) {

};
```

### TypeScript Solution:

```
/**
* Problem: Most Popular Video Creator
* 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 mostPopularCreator(creators: string[], ids: string[], views:
number[]): string[][] {

};
```

### C# Solution:

```
/*
* Problem: Most Popular Video Creator
* 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<IList<string>> MostPopularCreator(string[] creators, string[]
ids, int[] views) {

}
}
```

### C Solution:

```
/*
* Problem: Most Popular Video Creator
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** mostPopularCreator(char** creators, int creatorsSize, char** ids, int
idsSize, int* views, int viewsSize, int* returnSize, int** returnColumnSizes)
{

}
```

### Go Solution:

```
// Problem: Most Popular Video Creator
// 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 mostPopularCreator(creators []string, ids []string, views []int)
[][]string {

}
```

### Kotlin Solution:

```
class Solution {
fun mostPopularCreator(creators: Array<String>, ids: Array<String>, views:
IntArray): List<List<String>> {

}
}
```

### Swift Solution:

```
class Solution {
func mostPopularCreator(_ creators: [String], _ ids: [String], _ views:
[Int]) -> [[String]] {

}
}
```

### Rust Solution:

```
// Problem: Most Popular Video Creator
// 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 most_popular_creator(creators: Vec<String>, ids: Vec<String>, views:
Vec<i32>) -> Vec<Vec<String>> {

}
}
```

### Ruby Solution:

```
# @param {String[]} creators
# @param {String[]} ids
# @param {Integer[]} views
# @return {String[][]}
def most_popular_creator(creators, ids, views)

end
```

### PHP Solution:

```
class Solution {

/**
* @param String[] $creators
* @param String[] $ids
* @param Integer[] $views
* @return String[][]
*/
function mostPopularCreator($creators, $ids, $views) {

}
}
```

### Dart Solution:

```
class Solution {
List<List<String>> mostPopularCreator(List<String> creators, List<String>
ids, List<int> views) {

}
}
```

### Scala Solution:

```
object Solution {
def mostPopularCreator(creators: Array[String], ids: Array[String], views:
Array[Int]): List[List[String]] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec most_popular_creator(creators :: [String.t], ids :: [String.t], views
:: [integer]) :: [[String.t]]
def most_popular_creator(creators, ids, views) do

end
end
```

### Erlang Solution:

```
-spec most_popular_creator(Creators :: [unicode:unicode_binary()], Ids ::
[unicode:unicode_binary()], Views :: [integer()]) ->
[[unicode:unicode_binary()]].
most_popular_creator(Creators, Ids, Views) ->
.
```

### Racket Solution:

```
(define/contract (most-popular-creator creators ids views)
(-> (listof string?) (listof string?) (listof exact-integer?) (listof (listof
string?)))
)
```