Problem 1090: Largest Values From Labels

Problem Information

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

Problem Description

You are given

n

item's value and label as two integer arrays

values

and

labels

. You are also given two integers

numWanted

and

useLimit

.

Your task is to find a subset of items with the

maximum sum

of their values such that:

The number of items is

at most

numWanted

.

The number of items with the same label is

at most

useLimit

.

Return the maximum sum.

Example 1:

Input:

values = [5,4,3,2,1], labels = [1,1,2,2,3], numWanted = 3, useLimit = 1

Output:

9

Explanation:

The subset chosen is the first, third, and fifth items with the sum of values 5 + 3 + 1.

Example 2:

Input:

values = [5,4,3,2,1], labels = [1,3,3,3,2], numWanted = 3, useLimit = 2

Output:

12

Explanation:

The subset chosen is the first, second, and third items with the sum of values 5 + 4 + 3.

Example 3:

Input:

values = [9,8,8,7,6], labels = [0,0,0,1,1], numWanted = 3, useLimit = 1

Output:

16

Explanation:

The subset chosen is the first and fourth items with the sum of values 9 + 7.

Constraints:

n == values.length == labels.length

1 <= n <= 2 * 10

4

0 <= values[i], labels[i] <= 2 * 10

4

1 <= numWanted, useLimit <= n

Code Snippets

C++:

```
class Solution {
public:
int largestValsFromLabels(vector<int>& values, vector<int>& labels, int
numWanted, int useLimit) {

}
};
```

Java:

```
class Solution {
public int largestValsFromLabels(int[] values, int[] labels, int numWanted,
int useLimit) {

}
}
```

Python3:

```
class Solution:
def largestValsFromLabels(self, values: List[int], labels: List[int],
numWanted: int, useLimit: int) -> int:
```

Python:

```
class Solution(object):
def largestValsFromLabels(self, values, labels, numWanted, useLimit):
"""
:type values: List[int]
:type labels: List[int]
:type numWanted: int
:type useLimit: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} values
* @param {number[]} labels
* @param {number} numWanted
```

```
* @param {number} useLimit
* @return {number}
*/
var largestValsFromLabels = function(values, labels, numWanted, useLimit) {

};
```

TypeScript:

```
function largestValsFromLabels(values: number[], labels: number[], numWanted:
number, useLimit: number): number {

};
```

C#:

```
public class Solution {
public int LargestValsFromLabels(int[] values, int[] labels, int numWanted,
int useLimit) {

}
}
```

C:

```
int largestValsFromLabels(int* values, int valuesSize, int* labels, int
labelsSize, int numWanted, int useLimit) {

}
```

Go:

```
func largestValsFromLabels(values []int, labels []int, numWanted int,
useLimit int) int {

}
```

Kotlin:

```
class Solution {
fun largestValsFromLabels(values: IntArray, labels: IntArray, numWanted: Int,
useLimit: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func largestValsFromLabels(_ values: [Int], _ labels: [Int], _ numWanted:
Int, _ useLimit: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn largest_vals_from_labels(values: Vec<i32>, labels: Vec<i32>,
num_wanted: i32, use_limit: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} values
# @param {Integer[]} labels
# @param {Integer} num_wanted
# @param {Integer} use_limit
# @return {Integer}
def largest_vals_from_labels(values, labels, num_wanted, use_limit)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $values
* @param Integer[] $labels
* @param Integer $numWanted
* @param Integer $useLimit
```

```
* @return Integer
*/
function largestValsFromLabels($values, $labels, $numWanted, $useLimit) {

}
}
```

Dart:

```
class Solution {
int largestValsFromLabels(List<int> values, List<int> labels, int numWanted,
int useLimit) {

}
}
```

Scala:

```
object Solution {
def largestValsFromLabels(values: Array[Int], labels: Array[Int], numWanted:
Int, useLimit: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_vals_from_labels(values :: [integer], labels :: [integer],
num_wanted :: integer, use_limit :: integer) :: integer
def largest_vals_from_labels(values, labels, num_wanted, use_limit) do

end
end
```

Erlang:

```
-spec largest_vals_from_labels(Values :: [integer()], Labels :: [integer()],
NumWanted :: integer(), UseLimit :: integer()) -> integer().
largest_vals_from_labels(Values, Labels, NumWanted, UseLimit) ->
.
```

Racket:

```
(define/contract (largest-vals-from-labels values labels numWanted useLimit)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Values From Labels
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* 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:
int largestValsFromLabels(vector<int>& values, vector<int>& labels, int
numWanted, int useLimit) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Values From Labels
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* 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 int largestValsFromLabels(int[] values, int[] labels, int numWanted,
int useLimit) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Values From Labels
Difficulty: Medium
Tags: array, greedy, hash, sort

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 largestValsFromLabels(self, values: List[int], labels: List[int],
numWanted: int, useLimit: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def largestValsFromLabels(self, values, labels, numWanted, useLimit):
"""
:type values: List[int]
:type labels: List[int]
:type numWanted: int
:type useLimit: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Largest Values From Labels
```

```
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* 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 {number[]} values
* @param {number[]} labels
* @param {number} numWanted
* @param {number} useLimit
* @return {number}
*/
var largestValsFromLabels = function(values, labels, numWanted, useLimit) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Values From Labels
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* 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 largestValsFromLabels(values: number[], labels: number[], numWanted:
number, useLimit: number): number {

};
```

C# Solution:

```
/*
* Problem: Largest Values From Labels
* Difficulty: Medium
```

```
* Tags: array, greedy, hash, sort
*
* 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 int LargestValsFromLabels(int[] values, int[] labels, int numWanted,
int useLimit) {

}
}
```

C Solution:

```
/*
* Problem: Largest Values From Labels
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int largestValsFromLabels(int* values, int valuesSize, int* labels, int
labelsSize, int numWanted, int useLimit) {

}
```

Go Solution:

```
// Problem: Largest Values From Labels
// Difficulty: Medium
// Tags: array, greedy, hash, sort
//
// 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 largestValsFromLabels(values []int, labels []int, numWanted int,
useLimit int) int {

}
```

Kotlin Solution:

```
class Solution {
fun largestValsFromLabels(values: IntArray, labels: IntArray, numWanted: Int,
useLimit: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func largestValsFromLabels(_ values: [Int], _ labels: [Int], _ numWanted:
Int, _ useLimit: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Largest Values From Labels
// Difficulty: Medium
// Tags: array, greedy, hash, sort
//
// 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 largest_vals_from_labels(values: Vec<i32>, labels: Vec<i32>,
num_wanted: i32, use_limit: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} values
# @param {Integer[]} labels
# @param {Integer} num_wanted
# @param {Integer} use_limit
# @return {Integer}
def largest_vals_from_labels(values, labels, num_wanted, use_limit)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $values
* @param Integer[] $labels
* @param Integer $numWanted
* @param Integer $useLimit
* @return Integer
*/
function largestValsFromLabels($values, $labels, $numWanted, $useLimit) {

}
}
```

Dart Solution:

```
class Solution {
int largestValsFromLabels(List<int> values, List<int> labels, int numWanted,
int useLimit) {

}
}
```

Scala Solution:

```
object Solution {
def largestValsFromLabels(values: Array[Int], labels: Array[Int], numWanted:
Int, useLimit: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec largest_vals_from_labels(values :: [integer], labels :: [integer],
num_wanted :: integer, use_limit :: integer) :: integer
def largest_vals_from_labels(values, labels, num_wanted, use_limit) do

end
end
```

Erlang Solution:

```
-spec largest_vals_from_labels(Values :: [integer()], Labels :: [integer()],
NumWanted :: integer(), UseLimit :: integer()) -> integer().
largest_vals_from_labels(Values, Labels, NumWanted, UseLimit) ->
.
```

Racket Solution:

```
(define/contract (largest-vals-from-labels values labels numWanted useLimit)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer? exact-integer?)
)
```