Problem 1333: Filter Restaurants by Vegan-Friendly, Price and Distance

Problem Information

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

Problem Description

Given the array

restaurants

where

restaurants[i] = [id

i

, rating

i

, veganFriendly

i

, price

i

, distance

i

]

. You have to filter the restaurants using three filters.

The

veganFriendly

filter will be either

true

(meaning you should only include restaurants with

veganFriendly

i

set to true) or

false

(meaning you can include any restaurant). In addition, you have the filters

maxPrice

and

maxDistance

which are the maximum value for price and distance of restaurants you should consider respectively.

Return the array of restaurant

IDs

after filtering, ordered by

rating

from highest to lowest. For restaurants with the same rating, order them by

id

from highest to lowest. For simplicity

veganFriendly

i

and

veganFriendly

take value

1

when it is

true

, and

0

when it is

false

.

Example 1:

Input:

restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 1, maxPrice = 50, maxDistance = 10

Output:

[3,1,5]

Explanation:

The restaurants are: Restaurant 1 [id=1, rating=4, veganFriendly=1, price=40, distance=10] Restaurant 2 [id=2, rating=8, veganFriendly=0, price=50, distance=5] Restaurant 3 [id=3, rating=8, veganFriendly=1, price=30, distance=4] Restaurant 4 [id=4, rating=10, veganFriendly=0, price=10, distance=3] Restaurant 5 [id=5, rating=1, veganFriendly=1, price=15, distance=1] After filter restaurants with veganFriendly = 1, maxPrice = 50 and maxDistance = 10 we have restaurant 3, restaurant 1 and restaurant 5 (ordered by rating from highest to lowest).

Example 2:

Input:

restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 0, maxPrice = 50, maxDistance = 10

Output:

[4,3,2,1,5]

Explanation:

The restaurants are the same as in example 1, but in this case the filter veganFriendly = 0, therefore all restaurants are considered.

Example 3:

Input:

restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 0, maxPrice = 30, maxDistance = 3

Output:

[4,5]

Constraints:

1 <= restaurants.length <= 10^4

restaurants[i].length == 5

1 <= id

i

, rating

i

, price

i

, distance

i

<= 10^5

1 <= maxPrice, maxDistance <= 10^5

veganFriendly

i

and

veganFriendly

are 0 or 1.

All

id

i

are distinct.

Code Snippets

C++:

```
class Solution {
public:
vector<int> filterRestaurants(vector<vector<int>>& restaurants, int
veganFriendly, int maxPrice, int maxDistance) {

}
};
```

Java:

```
class Solution {
public List<Integer> filterRestaurants(int[][] restaurants, int
veganFriendly, int maxPrice, int maxDistance) {

}
}
```

Python3:

```
class Solution:
def filterRestaurants(self, restaurants: List[List[int]], veganFriendly: int,
maxPrice: int, maxDistance: int) -> List[int]:
```

Python:

```
class Solution(object):
def filterRestaurants(self, restaurants, veganFriendly, maxPrice,
maxDistance):
```

```
"""
:type restaurants: List[List[int]]
:type veganFriendly: int
:type maxPrice: int
:type maxDistance: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} restaurants
* @param {number} veganFriendly
* @param {number} maxPrice
* @param {number} maxDistance
* @return {number[]}
*/
var filterRestaurants = function(restaurants, veganFriendly, maxPrice,
maxDistance) {

};
```

TypeScript:

```
function filterRestaurants(restaurants: number[][], veganFriendly: number,
maxPrice: number, maxDistance: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FilterRestaurants(int[][] restaurants, int veganFriendly,
int maxPrice, int maxDistance) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
int* filterRestaurants(int** restaurants, int restaurantsSize, int*
restaurantsColSize, int veganFriendly, int maxPrice, int maxDistance, int*
returnSize) {

}
```

Go:

```
func filterRestaurants(restaurants [][]int, veganFriendly int, maxPrice int,
maxDistance int) []int {

}
```

Kotlin:

```
class Solution {
fun filterRestaurants(restaurants: Array<IntArray>, veganFriendly: Int,
maxPrice: Int, maxDistance: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func filterRestaurants(_ restaurants: [[Int]], _ veganFriendly: Int, _
maxPrice: Int, _ maxDistance: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn filter_restaurants(restaurants: Vec<Vec<i32>>, vegan_friendly: i32,
max_price: i32, max_distance: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} restaurants
# @param {Integer} vegan_friendly
# @param {Integer} max_price
# @param {Integer} max_distance
# @return {Integer[]}
def filter_restaurants(restaurants, vegan_friendly, max_price, max_distance)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $restaurants
* @param Integer $veganFriendly
* @param Integer $maxPrice
* @param Integer $maxDistance
* @return Integer[]
*/
function filterRestaurants($restaurants, $veganFriendly, $maxPrice,
$maxDistance) {

}
}
```

Dart:

```
class Solution {
List<int> filterRestaurants(List<List<int>> restaurants, int veganFriendly,
int maxPrice, int maxDistance) {

}
}
```

Scala:

```
object Solution {
def filterRestaurants(restaurants: Array[Array[Int]], veganFriendly: Int,
maxPrice: Int, maxDistance: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec filter_restaurants(restaurants :: [[integer]], vegan_friendly ::
integer, max_price :: integer, max_distance :: integer) :: [integer]
def filter_restaurants(restaurants, vegan_friendly, max_price, max_distance)
do

end
end
```

Erlang:

```
-spec filter_restaurants(Restaurants :: [[integer()]], VeganFriendly ::
integer(), MaxPrice :: integer(), MaxDistance :: integer()) -> [integer()].
filter_restaurants(Restaurants, VeganFriendly, MaxPrice, MaxDistance) ->
.
```

Racket:

```
(define/contract (filter-restaurants restaurants veganFriendly maxPrice
maxDistance)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
```

```
public:
vector<int> filterRestaurants(vector<vector<int>>& restaurants, int
veganFriendly, int maxPrice, int maxDistance) {

}
};
```

Java Solution:

```
/**
* Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<Integer> filterRestaurants(int[][] restaurants, int
veganFriendly, int maxPrice, int maxDistance) {

}
}
```

Python3 Solution:

```
"""
Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
Difficulty: Medium
Tags: array, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def filterRestaurants(self, restaurants: List[List[int]], veganFriendly: int,
maxPrice: int, maxDistance: int) -> List[int]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def filterRestaurants(self, restaurants, veganFriendly, maxPrice,
maxDistance):
"""
:type restaurants: List[List[int]]
:type veganFriendly: int
:type maxPrice: int
:type maxDistance: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} restaurants
* @param {number} veganFriendly
* @param {number} maxPrice
* @param {number} maxDistance
* @return {number[]}
*/
var filterRestaurants = function(restaurants, veganFriendly, maxPrice,
maxDistance) {

};
```

TypeScript Solution:

```
/**
* Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function filterRestaurants(restaurants: number[][], veganFriendly: number,
maxPrice: number, maxDistance: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<int> FilterRestaurants(int[][] restaurants, int veganFriendly,
int maxPrice, int maxDistance) {

}
}
```

C Solution:

```
/*
* Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* filterRestaurants(int** restaurants, int restaurantsSize, int*
restaurantsColSize, int veganFriendly, int maxPrice, int maxDistance, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func filterRestaurants(restaurants [][]int, veganFriendly int, maxPrice int,
maxDistance int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun filterRestaurants(restaurants: Array<IntArray>, veganFriendly: Int,
maxPrice: Int, maxDistance: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func filterRestaurants(_ restaurants: [[Int]], _ veganFriendly: Int, _
```

```
maxPrice: Int, _ maxDistance: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Filter Restaurants by Vegan-Friendly, Price and Distance
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn filter_restaurants(restaurants: Vec<Vec<i32>>, vegan_friendly: i32,
max_price: i32, max_distance: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} restaurants
# @param {Integer} vegan_friendly
# @param {Integer} max_price
# @param {Integer} max_distance
# @return {Integer[]}
def filter_restaurants(restaurants, vegan_friendly, max_price, max_distance)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $restaurants
* @param Integer $veganFriendly
* @param Integer $maxPrice
* @param Integer $maxDistance
```

```
* @return Integer[]
*/
function filterRestaurants($restaurants, $veganFriendly, $maxPrice,
$maxDistance) {

}
}
```

Dart Solution:

```
class Solution {
List<int> filterRestaurants(List<List<int>> restaurants, int veganFriendly,
int maxPrice, int maxDistance) {

}
}
```

Scala Solution:

```
object Solution {
def filterRestaurants(restaurants: Array[Array[Int]], veganFriendly: Int,
maxPrice: Int, maxDistance: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec filter_restaurants(restaurants :: [[integer]], vegan_friendly ::
integer, max_price :: integer, max_distance :: integer) :: [integer]
def filter_restaurants(restaurants, vegan_friendly, max_price, max_distance)
do

end
end
```

Erlang Solution:

```
-spec filter_restaurants(Restaurants :: [[integer()]], VeganFriendly ::
integer(), MaxPrice :: integer(), MaxDistance :: integer()) -> [integer()].
```

```
filter_restaurants(Restaurants, VeganFriendly, MaxPrice, MaxDistance) ->
.
```

Racket Solution:

```
(define/contract (filter-restaurants restaurants veganFriendly maxPrice
maxDistance)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer? (listof exact-integer?))
)
```