1333. Filter Restaurants by Vegan-Friendly, Price and Distance

Medium Array Sorting

Leetcode Link

Problem Description

In this problem, you are given an array `restaurants`, where each element in the array represents a restaurant and is itself an array of the form `[id_i, rating_i, veganFriendly_i, price_i, distance_i]`. You are required to filter these restaurants based on three criteria: whether they are vegan-friendly, their price, and their distance from the user.

The `veganFriendly` filter is a boolean that, when true, means you should only include restaurants where `veganFriendly_i` is set to `1`. When `veganFriendly` is false, you can include any restaurant, regardless of its `veganFriendly_i` value.

Additionally, you are given a maximum price `maxPrice` and a maximum distance `maxDistance`. Restaurants exceeding either of these values should not be included in the final list.

Your task is to return an array of restaurant IDs that satisfy all three filter conditions. The IDs should be ordered first by the restaurant's rating in descending order, and in the case of a tie in ratings, by the restaurant's ID in descending order.

Intuition

To solve this problem, we first need to deal with the restaurant ordering. Given that we want the restaurants sorted by rating and ID as secondary criteria, we can achieve this by sorting the `restaurants` array. In Python, we can use the `sort()` method with a custom key function that sorts by rating in descending order first and then by ID in descending order as a tiebreaker.

Once we have the sorted array, we need to filter out the restaurants according to the given criteria. We accomplish this by iterating over each restaurant and checking if it satisfies the conditions of being vegan-friendly (if required), within the maximum price, and within the maximum distance. If a restaurant meets all the conditions, we add its ID to our final list of restaurant IDs.

The reason behind sorting the array first before filtering is to ensure that once the filtering is done, we don't need to sort the resulting list again, as it will already be ordered according to the required criteria.

The key aspects of this solution are understanding how to sort the array according to multiple criteria and filtering the elements of the array based on given thresholds.

Solution Approach

The solution approach can be detailed through the following steps, which encompass the use of sorting and filtering in Python:

1. **Sorting:** Firstly, the given list of `restaurants` is sorted in a descending order based on the `rating_i`, and in the case where two restaurants have the same `rating_i`, they're further sorted by `id_i` in descending order. This is done using the `sort()` method with a lambda function as the `key` parameter. The lambda function returns a tuple `(-x[1], -x[0])`. Here, `x` represents an element (which is a list) in `restaurants`. The minus sign (`-`) is used to sort in descending order since Python's sort functions sort in ascending order by default. The tuple essentially says "sort by the second element (rating) in descending order, and if those are equal, sort by the first element (ID) in descending order".

```
restaurants.sort(key=lambda x: (-x[1], -x[0]))
```

2. **Filtering:** After sorting, the next step is to filter the restaurants based on the criteria:
   - If `veganFriendly` is `1`, then only include restaurants where `veganFriendly_i` is also `1`.
   - Include restaurants where the `price_i` is less than or equal to the `maxPrice`.
   - Include restaurants where the `distance_i` is less than or equal to the `maxDistance`.

   We use a `for` loop to iterate through each restaurant in the sorted `restaurants` list. For each restaurant (represented as a sublist), we check the three conditions mentioned above.

```
for idx, _, vegan, price, dist in restaurants:
    if vegan >= veganFriendly and price <= maxPrice and dist <= maxDistance:
        ans.append(idx)
```

   We declare an empty list to store the IDs of the restaurants that meet all of the filters. For each restaurant, `idx` represents its ID. We only append `idx` to `ans` if all the conditions are met.

3. **Return the Result:** Once the iteration is complete, the `ans` list contains the IDs of the restaurants that have passed through the filters, sorted as required. This list is then returned as the final answer.

```
return ans
```

The algorithms used in this implementation include sorting and simple linear iteration for filtering. The data structure used is a list. No complex patterns or data structures like trees, graphs, or dynamic programming are used; the solution is straightforward and only involves array manipulation.

Example Walkthrough

Let's walkthrough an example to illustrate the solution approach.

Suppose we have the following list of `restaurants` and the filters `veganFriendly`, `maxPrice`, and `maxDistance` given as:

- `restaurants` = [[1, 4, 1, 40, 10], [2, 8, 0, 50, 5], [3, 8, 1, 30, 4], [4, 10, 0, 10, 3], [5, 10, 1, 15, 1]]
- `veganFriendly` = 1
- `maxPrice` = 20
- `maxDistance` = 10

Initial List of Restaurants

[ID, Rating, VeganFriendly, Price, Distance]

1. [1, 4, 1, 40, 10]
2. [2, 8, 0, 50, 5]
3. [3, 8, 1, 30, 4]
4. [4, 10, 0, 10, 3]
5. [5, 10, 1, 15, 1]

Step 1: Sort Restaurants

First, we sort the list by rating, and for those with the same rating, by ID. In descending order by rating and ID:

- Sort result: [4, 10, 0, 10, 3], [5, 10, 1, 15, 1], [2, 8, 0, 50, 5], [3, 8, 1, 30, 4], [1, 4, 1, 40, 10]

Step 2: Filter Based on Vegan Friendly, Max Price, and Max Distance

Next, we filter by the vegan-friendly requirement, price, and distance. Since `veganFriendly` is 1, we only select the restaurants where `veganFriendly_i` is also 1. Then, we ensure `price_i` is less than or equal to `maxPrice` and `distance_i` is less than or equal to `maxDistance`.

- Restaurant 1: Fails (`veganFriendly` = 1, but `Price` = 40 (> 20))
- Restaurant 2: Fails (`veganFriendly` = 0)
- Restaurant 3: Fails (`Price` = 30 (> 20))
- Restaurant 4: Fails (`veganFriendly` = 0)
- Restaurant 5: Passes (`Rating` = 10, `VeganFriendly` = 1, `Price` = 15 (<= 20), `Distance` = 1 (<= 10))

Step 3: Return Result

The final list of restaurant IDs that meet all the criteria is:

- [5]

Thus, after following steps 1 and 2, our filtered and sorted list of restaurant IDs is `[5]`, which is the answer to our problem. This is the array we would return.

Python Solution

```
from typing import List

class Solution:
    def filterRestaurants(
        self,
        restaurants: List[List[int]],
        vegan_friendly: int,
        max_price: int,
        max_distance: int,
    ) -> List[int]:
        # Sort the restaurants list first by rating in descending order
        # then by restaurant_id, also in descending order in case of ties in ratings.
        restaurants.sort(key=lambda restaurant: (-restaurant[1], -restaurant[0]))

        # Initialize an empty list to hold the filtered restaurant ids
        filtered_restaurant_ids = []

        # Iterate over the sorted restaurants list
        for restaurant_id, _, is_vegan, price, distance in restaurants:
            # Apply the filters:
            # 1. Ensure the restaurant's vegan-friendliness meets the requirement (0 or 1)
            # 2. Ensure the price does not exceed the max price
            # 3. Ensure the distance does not exceed the max distance
            if is_vegan >= vegan_friendly and price <= max_price and distance <= max_distance:
                # If the restaurant meets all conditions, add its id to the filtered list.
                filtered_restaurant_ids.append(restaurant_id)

        # Return the finalized list of restaurant ids that meet all the specified criteria
        return filtered_restaurant_ids

# Example usage:
# solution_instance = Solution()
# result = solution_instance.filterRestaurants([[1,4,1,40,10],[2,8,0,50,5],...], 1, 50, 10)
# print(result) would print out the list of restaurant IDs that satisfy the input criteria.
```

Java Solution

```
import java.util.*;

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

        // Sort the array of restaurants based on their ratings in descending order
        // If the ratings are the same, sort based on the restaurant IDs in descending order
        Arrays.sort(restaurants, (restaurant1, restaurant2) -> {
            if (restaurant1[1] == restaurant2[1])
                return restaurant2[0] - restaurant1[0]; // Sort by ID if ratings are equal
            else
                return restaurant2[1] - restaurant1[1]; // Sort by Rating
        });

        // Initialize a list to store the IDs of restaurants that satisfy the conditions
        List<Integer> filteredRestaurants = new ArrayList<>();

        // Iterate through the sorted array of restaurants
        for (int[] restaurant : restaurants) {
            // Check if the restaurant satisfies the conditions for vegan-friendly, max price and max distance
            if (veganFriendly >= restaurant[2] && restaurant[3] <= maxPrice && restaurant[4] <= maxDistance) {
                // Add the restaurant ID to the list
                filteredRestaurants.add(restaurant[0]);
            }
        }

        // Return the list of filtered restaurant IDs
        return filteredRestaurants;
    }
}
```

C++ Solution

```
#include <vector>
#include <algorithm>

class Solution {
public:
    // Rewritten function that filters restaurants by given criteria.
    std::vector<int> filterRestaurants(std::vector<std::vector<int>>& restaurants, int veganFriendly, int maxPrice, int maxDistance) {
        // Sort the restaurants array using a custom lambda function for sorting.
        // Highest rating first, then by ID if ratings are the same.
        std::sort(restaurants.begin(), restaurants.end(), [](const std::vector<int>& a, const std::vector<int>& b) {
            // First, compare based on ratings.
            if (a[1] != b[1]) {
                return a[1] > b[1]; // Return true if 'a' has a higher rating than 'b'.
            }
            // If ratings are the same, compare based on ID.
            return a[0] > b[0]; // Return true if 'a' has a greater ID than 'b'.
        });

        // Initialize an empty vector to store the IDs of the filtered restaurants.
        std::vector<int> filteredRestaurants;

        // Loop through all the restaurants.
        for (auto& restaurant : restaurants) {
            // Check if each restaurant meets the criteria:
            // 1. Vegan-friendly if required.
            // 2. Price does not exceed maxPrice.
            // 3. Distance does not exceed maxDistance.
            if ((veganFriendly == 0 || restaurant[2] == 1) && restaurant[3] <= maxPrice && restaurant[4] <= maxDistance) {
                filteredRestaurants.push_back(restaurant[0]); // Add the restaurant's ID to the filtered list.
            }
        }

        // Return the list of restaurant IDs that meet the criteria.
        return filteredRestaurants;
    }
};
```

Typescript Solution

```
function filterRestaurants(
    restaurants: number[][], // Array of restaurant details; each restaurant is an array of [id, rating, veganFriendly, price, distance]
    veganFriendly: number,   // Filter flag for vegan-friendly restaurants (1 for vegan friendly, 0 otherwise)
    maxPrice: number,        // Maximum acceptable price for filtering
    maxDistance: number      // Maximum acceptable distance for filtering
): number[] {
    // Sort the restaurants first by rating descending, then by id descending in case of a tie on rating
    restaurants.sort((restaurantA, restaurantB) => {
        const ratingA = restaurantA[1];
        const ratingB = restaurantB[1];
        const idA = restaurantA[0];
        const idB = restaurantB[0];

        // If ratings are equal, sort by ID; otherwise sort by rating
        if (ratingA === ratingB) {
            return idB - idA; // for tie on ratings, higher ID comes first
        } else {
            return ratingB - ratingA; // higher rating comes first
        }
    });

    const filteredRestaurantIds: number[] = []; // Array to store the IDs of restaurants that meet the filter criteria

    // Iterate through each restaurant to apply filtration based on the given criteria
    for (const [id, , vegan, price, distance] of restaurants) {
        // Check if each restaurant meets all the filter conditions
        if (vegan >= veganFriendly && price <= maxPrice && distance <= maxDistance) {
            filteredRestaurantIds.push(id); // Add restaurant ID to the result if it meets the criteria
        }
    }

    return filteredRestaurantIds; // Return the array containing IDs of filtered restaurants
}
```

Time and Space Complexity

Time Complexity

The time complexity of the provided solution primarily depends on the sorting operation and the subsequent filtering of the `restaurants` list based on the specified conditions.

- **Sorting**: The `sort()` function is used, which generally has a time complexity of $O(n \log n)$, where n is the number of restaurants.
- **Filtering**: The for loop iterates over each restaurant, performing constant-time checks (`if` conditions) for each. This gives us $O(n)$.

The overall time complexity combines the above operations, where sorting dominates, resulting in $O(n \log n) + O(n)$. Since the $O(n \log n)$ term is the dominant factor, the overall time complexity simplifies to $O(n \log n)$.

Space Complexity

The space complexity of the solution involves the storage required for the sorted list and the answer list.

- **Sorted List**: The in-place `sort()` method is used, so it does not require additional space apart from the input list. Hence, the space required remains $O(1)$ as an auxiliary space.
- **Answer List**: In the worst case, all restaurants might satisfy the given conditions, so the `ans` list could contain all restaurant IDs. Therefore, the space complexity for the `ans` list is $O(n)$.

Taking both into consideration, the overall space complexity of the solution is $O(n)$, where n is the number of restaurants.