2456. Most Popular Video Creator

Medium Array Hash Table String Sorting Heap (Priority Queue) Leetcode Link

Problem Description

In this problem, we're working with a platform that hosts videos. Each video has a unique creator and ID and has accumulated a certain number of views. We're given three arrays of equal length `n`:

- `creators` which includes the names of the creators of the videos.
- `ids` which contains the unique identifiers for each video.
- `views` representing the number of times each video has been watched.

The goal is to calculate the **popularity** of each creator, defined as the sum of the views of all their videos, and then determine the creator(s) with the highest popularity. For each creator with the maximum popularity, we also need to find the ID of their most viewed video. If there is more than one such video, we select the one with the lexicographically smallest ID. The final output should be a 2D array listing the creators with the highest popularity alongside the ID of their most viewed video.

Special conditions to be aware of:

1. There may be more than one creator tieing for the highest popularity.
2. If a creator has multiple videos with the same highest view count, the one with the smallest ID in lexicographic order should be chosen.

The expectation is to have a comprehensive solution that handles these special cases correctly.

Intuition

To solve this problem, we can tackle it step-by-step:

1. We need to keep track of the total views for each creator. This can be achieved by iterating over the given `creators` and `views` arrays and accumulating the views in a dictionary where the keys are creator names and the values are their total views.
2. We need to find each creator's most viewed video. This also requires iteration over the arrays, but this time we are tracking the maximum number of views for each creator. We use another dictionary to map the creator's name to the index of their most viewed video.
3. If there is a tie in view counts for a creator's videos, we must make sure to select the video ID that is the smallest lexicographically. We can achieve this by updating the dictionary only when we find a video with more views or if the view count is the same but the video ID is smaller lexicographically.
4. After populating the dictionaries with the necessary information, we determine the maximum popularity by looking at the values in the views dictionary.
5. Finally, we compile the list of creators who have matched the highest popularity and pair them with the ID of their most viewed video (referenced by index in the final step) to form the 2D array.

The solution code implements these steps with efficient lookups and comparisons using dictionaries, and it accommodates the possibility of multiple creators sharing the same highest popularity, as well as the need to compare strings lexicographically.

The key aspects of this approach involve the use of hashing (dictionaries) for fast lookups and careful logic to handle ties in both popularity and view counts, ensuring the specified conditions for selecting the most viewed video ID are satisfied.

Solution Approach

The implementation of the solution can be broken down into distinct stages aligned with the problem requirements and utilizing specific data structures for efficiency:

1. **Use of Default Dicts**: The solution uses two `defaultdict`s from Python's `collections` module to manage the accumulation of views and tracking of the most viewed video indices. A `defaultdict(int)` automatically initializes any new key with an integer value of `0`, facilitating easy summation.

2. **Tracking Total Views**: Iterating over the `creators`, `ids`, and `views` arrays simultaneously, the code increments the view count for each creator. This is achieved by the `for` loop and by summing up views into the `cnt` dictionary: `cnt[c] += v`.

3. **Finding the Most Viewed Video ID**: Alongside counting views, for each creator, we track the index of their most viewed video using the `d` dictionary. A comparison is made to determine if the current video either has more views or, on equal views, a lexicographically smaller ID than the previously tracked video.

   The condition `if c not in d or views[d[c]] < v or (views[d[c]] == v and ids[d[c]] > i):` ensures that we are only updating the record for a creator in `d` when a video with more views is found or if we find a video with the same number of views but a smaller ID, ensuring the proper selection per the problem's constraints.

4. **Determining the Maximum Popularity**: Once the iteration is complete and all view counts and most viewed video indices are stored, we determine the maximum popularity using Python's `max` function on the values of the `cnt` dictionary: `mx = max(cnt.values())`.

5. **Building the Answer**: The final step is to construct a 2D array that contains the highest popularity creators and the IDs of their most viewed videos. This is done by building a list comprehension that checks for creators `c` whose popularity `x` is equal to the maximum `mx` and then pairs them with the most viewed video ID found at `ids[d[c]]`. The final list comprehension looks like this: `[[c, ids[d[c]]] for c, x in cnt.items() if x == mx]`.

This solution approach expertly combines efficient iteration, conditional logic, and Python's built-in functions and data structures to fulfill the complex requirements of the problem, leading to an optimal algorithm that is both succinct and highly readable.

Example Walkthrough

To illustrate the solution approach, let's use a small example with the following data:

- `creators = ["Anne", "Ben", "Anne", "Ben", "Cara"]`
- `ids = ["A2", "B1", "A1", "B2", "C1"]`
- `views = [100, 150, 50, 200, 100]`

Following the steps of the solution approach:

1. **Use of Default Dicts**: We create two `defaultdict`s, one (`cnt`) for tracking the total views per creator and another (`d`) for tracing the most viewed video index for each creator.

2. **Tracking Total Views**: As we iterate, we sum the views for each creator in `cnt` like this:

   - `cnt["Anne"] += 100` (first video by Anne),
   - `cnt["Ben"] += 150` (first video by Ben),
   - `cnt["Anne"] += 50` (second video by Anne, now Anne's total is 150),
   - and so on...
3. **Finding the Most Viewed Video ID**: For each creator, we determine the most viewed video. After iterating, we end up with:

   - `d["Anne"]` points to the index of "A2" because "A2" has more views than "A1",
   - `d["Ben"]` points to the index of "B2" because it has more views,
   - for "Cara", we only have one video, so `d["Cara"]` points to "C1".
4. **Determining the Maximum Popularity**: We find the maximum popularity. In this case:

   - `mx` is `200`, the total views of Ben's most popular video.
5. **Building the Answer**: We build the final 2D array that lists creators with the highest popularity and their most viewed videos. Only Ben has the maximum popularity of 200, so the answer is:

   - `[['Ben', 'B2']]` since Ben's most viewed video is "B2" with 200 views.

In this case, our output indicates that Ben is the most popular creator with the most viewed video "B2". If there were another creator with a total view count of 200, they would also appear in the final array with their most viewed video.

Python Solution

```python
from collections import defaultdict

class Solution:
    def mostPopularCreator(self, creators: List[str], ids: List[str], views: List[int]) -> List[List[str]]:
        # Initialize two dictionaries to keep track of view counts and most viewed video indices for each creator.
        creator_view_count = defaultdict(int)
        creator_best_video_index = defaultdict(int)

        # Loop through each video and its associated creator and views.
        for index, (creator, video_id, view_count) in enumerate(zip(creators, ids, views)):
            # Increment the view count for the creator.
            creator_view_count[creator] += view_count

            # If this is the first time we see the creator or if this video has more views than the currently
            # recorded best one (or same views but smaller id), then update the most viewed video index.
            if (creator not in creator_best_video_index or
                views[creator_best_video_index[creator]] < view_count or
                (views[creator_best_video_index[creator]] == view_count and ids[creator_best_video_index[creator]] > video_id)):
                creator_best_video_index[creator] = index

        # Find the maximum view count across all creators.
        max_view_count = max(creator_view_count.values())

        # Create a list of [creator, video_id] for those creators whose total view count equals the max view count.
        most_popular_creators = [
            [creator, ids[creator_best_video_index[creator]]]
            for creator, total_views in creator_view_count.items() if total_views == max_view_count
        ]

        # Return the list of most popular creators and their most popular video ids.
        return most_popular_creators
```

In addition, the required `List` type hint should be imported from the `typing` module, which is not shown in the code snippet. To include it, add the following line at the beginning of your script:

```python
from typing import List

```

Java Solution

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

class Solution {
    public List<List<String>> mostPopularCreator(String[] creators, String[] ids, int[] views) {
        // Total number of entries
        int numberOfEntries = ids.length;
        // Map to store the total views for each creator
        Map<String, Long> creatorViewsCount = new HashMap<>(numberOfEntries);
        // Map to store the index of the most viewed id per creator
        Map<String, Integer> mostViewedIdIndex = new HashMap<>(numberOfEntries);

        // Iterate over all entries
        for (int index = 0; index < numberOfEntries; ++index) {
            String creator = creators[index], id = ids[index];
            long viewCount = views[index];

            // Sum up views for each creator
            creatorViewsCount.merge(creator, viewCount, Long::sum);

            // Update the most viewed id index for the creator if this entry has more views
            // or if the view count is the same but the id is lexicographically smaller
            if (!mostViewedIdIndex.containsKey(creator) || views[mostViewedIdIndex.get(creator)] < viewCount
                || (views[mostViewedIdIndex.get(creator)] == viewCount && ids[mostViewedIdIndex.get(creator)].compareTo(id) > 0)) {
                mostViewedIdIndex.put(creator, index);
            }
        }

        // Find the maximum views across all creators
        long maxViews = 0;
        for (long viewCount : creatorViewsCount.values()) {
            maxViews = Math.max(maxViews, viewCount);
        }

        // List to store the result
        List<List<String>> answer = new ArrayList<>();

        // Iterate through the view counts and find creators with view counts equal to maxViews
        for (var entry : creatorViewsCount.entrySet()) {
            if (entry.getValue() == maxViews) {
                String mostPopularCreator = entry.getKey();
                answer.add(List.of(mostPopularCreator, ids[mostViewedIdIndex.get(mostPopularCreator)]));
            }
        }

        // Return the list of creators with the most viewed contents and their respective most viewed content ids
        return answer;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
using namespace std;

class Solution {
public:
    // Function to find the creators with the most views and their most viewed content.
    vector<vector<string>> mostPopularCreator(vector<string>& creators, vector<string>& contentIds, vector<int>& views) {
        unordered_map<string, long long> creatorViewsSum; // Map to store the sum of views per creator
        unordered_map<string, int> creatorHighestViewIndex; // Map to store the index of each creator's content with the highest view

        int contentCount = contentIds.size(); // Total number of contents

        // Aggregate views for each creator and identify the content with highest views
        for (int i = 0; i < contentCount; ++i) {
            string creator = creators[i];
            string contentId = contentIds[i];
            int viewCount = views[i];

            creatorViewsSum[creator] += viewCount; // Summing up the views for each creator
            // Check if the current content has more views than the stored one, or if it is not stored yet
            if (!creatorHighestViewIndex.count(creator) || views[creatorHighestViewIndex[creator]] < viewCount ||
                (views[creatorHighestViewIndex[creator]] == viewCount && contentIds[creatorHighestViewIndex[creator]] > contentId)) {
                creatorHighestViewIndex[creator] = i;
            }
        }

        long long maximumViews = 0;
        // Find the maximum number of views across all creators
        for (auto& pair : creatorViewsSum) {
            maximumViews = max(maximumViews, pair.second);
        }

        vector<vector<string>> result; // Final result to store the creators with the most views along with their most popular conter
        // Iterate through the creators to find those with the highest views and add them along with their popular content id to the
        for (auto& pair : creatorViewsSum) {
            if (pair.second == maximumViews) {
                result.push_back({pair.first, contentIds[creatorHighestViewIndex[pair.first]]});
            }
        }
        return result;
    }
};

```

Typescript Solution

```typescript
function mostPopularCreator(creators: string[], ids: string[], views: number[]): string[][] {
    // Create a map to store the total views per creator
    const viewCounts: Map<string, number> = new Map();
    // Create a map to store the index of the most viewed content for each creator
    const mostViewedIndex: Map<string, number> = new Map();
    // Get the number of elements in the arrays
    const numElements = ids.length;

    // Iterate through each content piece
    for (let index = 0; index < numElements; ++index) {
        const creator = creators[index];
        const contentId = ids[index];
        const viewCount = views[index];

        // Update the total views for the creator
        viewCounts.set(creator, (viewCounts.get(creator) ?? 0) + viewCount);

        // Determine if the current content has more views or a lower id (in case of a tie) than the stored one
        if (!mostViewedIndex.has(creator) ||
            views[mostViewedIndex.get(creator)!] < viewCount ||
            (views[mostViewedIndex.get(creator)!] === viewCount && ids[mostViewedIndex.get(creator)!] > contentId)) {
            mostViewedIndex.set(creator, index);
        }
    }

    // Find the maximum view count across all creators
    const maxViewCount = Math.max(...viewCounts.values());
    // Prepare the result array
    const result: string[][] = [];

    // Find all creators who have the maximum view count
    for (const [creator, totalViews] of viewCounts) {
        if (totalViews === maxViewCount) {
            // Add the creator and their most viewed content id to the result array
            result.push([creator, ids[mostViewedIndex.get(creator)!]]);
        }
    }

    // Return the final results
    return result;
}

```

Time and Space Complexity

The time complexity of the given code is `O(N)`, where `N` is the total number of videos in the `views` list. This time complexity arises from a single loop over the list of creators, ids, and views, processing each element once. Within the loop, operations of constant time complexity such as dictionary access and comparison are performed. The subsequent loop to generate the result list also runs in `O(N)` in the worst case, where every creator has the maximum views, thus keeping the overall time complexity at `O(N)`.

The space complexity of the code is also `O(N)`. Two dictionaries, `cnt` and `d`, store information for each creator, where `cnt` stores the sum of views and `d` stores the index of their most viewed video under specific conditions. The size of these dictionaries scales with the number of creators, which can be up to `N` in the case where all creators have a unique video. The space for the input lists `creators`, `ids`, and `views` is not counted towards this complexity as they are typically considered as input space.