2512. Reward Top K Students

Medium Array Hash Table String Sorting Heap (Priority Queue)

Problem Description

In this problem, you are dealing with student feedback reports that contain words signifying either positive or negative feedback. Each student starts with zero points. The points a student has are affected by the words in their feedback report: every time a positive word is mentioned, their points are increased by 3, and for each negative word, their points are reduced by 1.

You are given two arrays `positive_feedback` and `negative_feedback` which list the words that count as positive and negative feedback, respectively. Alongside this, there is an array `report` which contains feedback reports for the students, and a corresponding `student_id` array that links each feedback report to a student's ID.

Your task is to calculate the final points for each student after analyzing the feedback reports, and then rank the students according to their points. If there's a tie in points, the student with a lower ID should come first. After ranking them, you will return a list of the IDs of the top `k` students, where `k` is given integer.

Intuition

To find a solution to this problem, let's consider how we would approach this task manually:

- First, we recognize that each word in the feedback is binary; it either has a positive effect or a negative effect on the student's points. So, we will want to quickly determine whether a word is positive or negative. To expedite this word lookup process, we'll use a hash set for both `positive_feedback` and `negative_feedback` since hash sets can substantially improve the speed of membership checking.
- Next, we associate each report with the student that it belongs to. We'll need to calculate the total points for each student by going through each word in their report and adjusting their score according to whether the word is in the positive set or the negative set.
- Once we have calculated the points for each student, we'll need to rank the students. However, because two students can have the same number of points, we'll need a way to break ties. The problem specifies that in such cases, the student with the lower student ID should rank higher.
- The final step is to sort the students based on their points, and in the case of ties, their IDs. After sorting, we will just select the top `k` students as per the sorted order.

To implement this in code, we'll traverse the `report` array and calculate the points for each student. We'll keep the scores and their respective student IDs in an array, then sort the array by the points in descending order, and by student ID in ascending order to break ties. Lastly, we extract the first `k` elements from the sorted array which represent the top `k` students.

Solution Approach

The solution provided uses a combination of data structures (sets for quick word lookup and arrays for storing final scores) and algorithms (iterative score calculation and sorting).

Here's a step-by-step breakdown:

- **Step 1:** Convert the lists of positive and negative feedback words into sets, `ps` and `ns` respectively. This is done to take advantage of the average constant time complexity for lookup operations in a set. The use of sets ensures that when we check if a word is either positive or negative, the operation is fast and efficient.
- **Step 2:** Initialize an empty array `arr`. This will store tuples, where each tuple consists of a student's total score and their ID (`-tid`, `sid`).
- **Step 3:** Loop through each `feedback report` paired with the respective `student_id`. Inside the loop, set a temporary score `t` to zero; this variable will hold the cumulative score for the student as we iterate through their report.
- **Step 4:** Iterate over every word `w` in each feedback report by splitting the report's string. Within this loop, we perform the following checks for each word:
  - If the word is in the positive set `ps`, increase `t` by `3`.
  - If the word is in the negative set `ns`, decrease `t` by `1`. As we iterate through the words, we are effectively calculating the net score for a student based on the positive and negative feedback words.
- **Step 5:** After calculating the total score `t` for a student, we append a tuple containing the score and the student's ID to the array `arr`.
- **Step 6:** Sort the array `arr` first by the score in descending order, and then by student ID in ascending order. The sorting technique used is a custom sort based on a lambda function as the key, which implements the rules for ranking: `(-x[0], x[1])`. This means we are sorting primarily by the first element of the tuple (the score) in descending order and then by the second element (the student ID) in ascending order.
- **Step 7:** Now that we have a sorted array of students by their score and ID, we extract the first `k` elements from the array. Specifically, we generate a new list that contains just the student IDs of these top `k` elements.

The overall complexity is governed by the time it takes to go through the reports and calculate the scores (which is linear with respect to the number of words in all reports), and the time it takes to sort the array of scores (which depends on the sorting algorithm used, typically O(n log n), where n is the number of students). Therefore, the solution is efficient and scales well with the number of students and the length of their feedback reports.

Example Walkthrough

Let's use a small example to illustrate the solution approach:

Suppose we have the following inputs:

- `positive_feedback` = ["excellent", "good", "superb"]
- `negative_feedback` = ["poor", "bad", "horrible"]
- `report` = ["excellent work but poor attention", "bad results but good effort", "superb participation", "good performance"]
- `student_id` = [102, 101, 104, 103]
- `k` = 2

Step 1: Convert the positive and negative feedback arrays into sets for faster lookup:

- `ps` = {"excellent", "good", "superb"}
- `ns` = {"poor", "bad", "horrible"}

Step 2: Initialize an empty array `arr` to store tuples of the form `(score, student_id)`.

Step 3: Begin iterating over each `report` and the respective `student_id`. For example, start with the first report "`excellent work but poor attention`" and `student_id` 102.

Step 4: For the first report, set a temporary score `t` to zero. Then iterate over every word:

- "excellent" is in `ps`, so add 3 to `t`: `t = 0 + 3`
- "work" is not in either set, `t` remains unchanged
- "but" is not in either set, `t` remains unchanged
- "poor" is in `ns`, so subtract 1 from `t`: `t = 3 - 1`
- "attention" is not in either set, `t` remains unchanged

Step 5: After processing the first report, `t` is 2. Append the tuple (2, 102) to `arr`.

Repeat **Steps 3-5** for the remaining reports:

- For the second report "bad results but good effort" with `student_id` 101, the final score `t` will be 1. Append (1, 101).
- For the third report "superb participation" with `student_id` 104, the final score `t` is 3. Append (3, 104).
- For the fourth report "good performance" with `student_id` 103, the final score `t` is 3. Append (3, 103).

Step 6: Now, `arr` = [(2, 102), (1, 101), (3, 104), (3, 103)]. Sort `arr` based on the score in descending order and student ID in ascending order. The sorted `arr` will be `[(3, 103), (3, 104), (2, 102), (1, 101)]`.

Step 7: Extract the top `k` student IDs from the sorted list, which gives `[103, 104]` as the top 2 students.

Thus, the solution process allows us to rank the students by their feedback scores efficiently, taking into consideration both the feedback words and the student IDs for tie-breaking.

Python Solution

```python
from typing import List

class Solution:
    def topStudents(
        self,
        positive_feedback: List[str],
        negative_feedback: List[str],
        reports: List[str],
        student_ids: List[int],
        k: int,
    ) -> List[int]:
        # Convert positive and negative feedbacks to sets for quick lookup
        ps = set(positive_feedback)
        ns = set(negative_feedback)

        # Initialize an empty array to store tuples of scores and student IDs
        arr = []

        # Iterate over each student's report and their corresponding student ID
        for student_id, report in zip(student_ids, reports):
            # Initialize total_score for the student
            t = 0

            # For every word in the student's report, check if it is in
            # positive feedback or negative feedback, and update total_score
            for w in report.split():
                if w in ps:
                    t += 3  # Add 3 to the score for positive feedback
                elif w in ns:
                    t -= 1  # Subtract 1 from the score for negative feedback

            # Append the total_score and student_id as a tuple to the scores list
            arr.append((-t, student_id))

        # Sort the scores in descending order by score, and then by student_id for ties
        arr.sort()

        # Return a list of the top k student_ids with the highest scores
        return [student_id for _, student_id in arr[:k]]
```

Java Solution

```java
import java.util.*;

class Solution {
    public List<Integer> topStudents(String[] positiveFeedback, String[] negativeFeedback,
                                     String[] reports, int[] studentIds, int k) {

        // Convert positive and negative feedbacks to sets for quick lookup
        Set<String> positiveFeedbackSet = new HashSet<>(Arrays.asList(positiveFeedback));
        Set<String> negativeFeedbackSet = new HashSet<>(Arrays.asList(negativeFeedback));

        // Initialize the number of reports
        int numOfReports = reports.length;

        // Create an array to store scores and corresponding student IDs
        int[][] scoresAndStudentIds = new int[numOfReports][2];

        // Iterate over each report
        for (int i = 0; i < numOfReports; ++i) {
            // Split the report into words
            String[] words = reports[i].split(" ");
            // Initialize score for the current report
            int studentId = studentIds[i];
            // Initialize score for the current report
            int score = 0;

            // Split the report into words
            for (var word : words) {
                // Check if the word is in positive feedback set
                if (positiveFeedbackSet.contains(word)) {
                    score += 3; // Add 3 to the score for positive feedback
                } else if (negativeFeedbackSet.contains(word)) {
                    score -= 1; // Subtract 1 from the score for negative feedback
                }
            }
            // Assign computed score and student ID to the scores array
            scoresAndStudentIds[i] = new int[] {score, studentId};
        }

        // Sort the array first by scores in descending order, and then by student IDs in ascending order
        Arrays.sort(scoresAndStudentIds, (a, b) -> {
            if (a[0] == b[0]) return a[1] - b[1]; // Same score, sort by ID
            return b[0] - a[0]; // Different scores, sort by score
        });

        // Initialize the list to store top k student IDs
        List<Integer> topStudents = new ArrayList<>();

        // Extract the top k student IDs
        for (int i = 0; i < k; ++i) {
            topStudents.add(scoresAndStudentIds[i][1]);
        }

        // Return the list of top k student IDs
        return topStudents;
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <string>
#include <unordered_set>
#include <sstream>
#include <algorithm>

class Solution {
public:
    std::vector<int> topStudents(std::vector<std::string>& positiveFeedback, std::vector<std::string>& negativeFeedback,
                                 std::vector<std::string>& reports, std::vector<int>& studentIds, int k) {

        // Convert the positive and negative feedback vectors to sets for efficient lookup
        std::unordered_set<std::string> positiveFeedbackSet(positiveFeedback.begin(), positiveFeedback.end());
        std::unordered_set<std::string> negativeFeedbackSet(negativeFeedback.begin(), negativeFeedback.end());
        std::vector<std::pair<int, int>> scoresAndIds; // Vector to store the score and student ID pairs

        int numReports = reports.size();
        for (int i = 0; i < numReports; ++i) {
            int studentId = studentIds[i];
            std::vector<std::string> words = split(reports[i], ' ');
            int totalScore = 0;

            // Calculate the score for each word in the report
            for (auto& word : words) {
                if (positiveFeedbackSet.count(word)) {
                    totalScore += 3;  // Add 3 for positive feedback
                } else if (negativeFeedbackSet.count(word)) {
                    totalScore -= 1;  // Subtract 1 for negative feedback
                }
            }

            // Store the negative of the score for sorting in ascending order later
            scoresAndIds.push_back({-totalScore, studentId});
        }

        // Sort the vector by score and then by student ID
        std::sort(scoresAndIds.begin(), scoresAndIds.end());

        std::vector<int> topStudentIDs;
        for (int i = 0; i < k; ++i) {
            topStudentIDs.emplace_back(scoresAndIds[i].second);
        }

        return topStudentIDs; // Return the top k students' IDs
    }

    // Utility function to split a string by a delimiter
    std::vector<std::string> split(std::string str, char delimiter) {
        std::vector<std::string> result;
        std::string item;
        std::istringstream iss(str);
        while (std::getline(iss, item, delimiter)) {
            result.emplace_back(item);
        }
        return result;
    }
};
```

Typescript Solution

```typescript
function topStudents(
    positiveFeedback: string[],
    negativeFeedback: string[],
    reports: string[],
    studentIds: number[],
    k: number
): number[] {
    // Get the number of reports (and students as both should be equal)
    const numReports = studentIds.length;
    // Create an array to store the student IDs and their calculated scores
    const scoresAndIds = new Array(numReports);

    // Create Sets from the positive and negative feedback arrays for efficient lookup
    const positiveFeedbackSet = new Set(positiveFeedback);
    const negativeFeedbackSet = new Set(negativeFeedback);

    // Iterate over each student's report
    for (let i = 0; i < numReports; ++i) {
        // Calculate the score for the current student's report
        const words = reports[i].split(' ');
        let totalScore = words.reduce((score, word) => {
            // If the word is in the positive feedback set, add 3 to the total score
            if (positiveFeedbackSet.has(word)) {
                return score + 3;
            }
            // If the word is in the negative feedback set, subtract 1 from the total score
            if (negativeFeedbackSet.has(word)) {
                return score - 1;
            }
            // If the word is neither positive nor negative, do not change the score
            return score;
        }, 0);

        // Update the Map with the current student's id and their calculated score
        scoresMap.set(studentIds[i], score);
    }

    // Convert the Map into an array, sort it based on scores and then student IDs
    // in case of ties, and then extract only the student IDs
    return Array.from(scoresMap.entries())
        .sort(([studentId1, score1], [studentId2, score2]) => {
            // Sort by score in descending order, and by student ID ascendingly
            if (score1 === score2) {
                return studentId1 - studentId2;
            }
            // Otherwise, sort by score descendingly
            return score2 - score1;
        })
        .map(([studentId, _]) => studentId) // Extract only the student IDs from the sorted array
        .slice(0, k); // Get the top 'k' student IDs based on their scores
}
```

Time and Space Complexity

Time Complexity

The time complexity of the provided code is $O(n * log n * (|ps| + |ns| + n) + |s|)$. To break it down:

- `zip(student_id, report)` operation is $O(n)$ because it is iterating through the list of student IDs and their corresponding reports.
- Splitting the `report` and checking if each `word` is in the sets `positive_feedback` (ps) or `negative_feedback` (ns) contributes $O(n * |s|)$, where s is the number of words and $|s|$ is the average length of a report since we consider $|s|$ for the average length of words in a report.
- The presence checks in ps and ns is $O(1)$ on average for hash sets, which is multiplied with $n * |s|$ for all words in all reports.
- Sorting the `arr` list of tuples based on a compound key contributes $(n * log n)$ since Python's sort function uses TimSort, which has this complexity.

Space Complexity

The space complexity of the code is $O(|ps|+|ns|) + |s| + n)$. Here's how this is derived:

- Space for the sets ps and ns contributes $|ps| + |s|$ and $|ns| + |s|$, respectively, because each word of average length $|s|$ is stored in these sets.
- The `arr` list, which stores tuples of total score and student IDs for n students, contributes $O(n)$ to space.