2512. Reward Top K Students Medium Array Hash Table String) Heap (Priority Queue) Sorting

Leetcode Link

In this problem, you are dealing with student feedback reports that contain words signifying either positive or negative feedback.

Problem Description

IDs of the top k students, where k is a given integer.

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 unique student 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

points. So, we will want to quickly determine whether a word is positive or negative. To expedite this word lookup process, we'll

negative set.

Intuition

membership checking.

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

use a hash set for both positive_feedback and negative_feedback since hash sets can substantially improve the speed of 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

• Once we have calculated the points for each student, we'll need to rank the students. However, because two students can have

• 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

- 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.
- 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

• Step 2: Initialize an empty array arr. This will store tuples, where each tuple consists of a student's total score and their ID (t,

sid).

arr.

Example Walkthrough

• k = 2

Suppose we have the following inputs:

student_id = [102, 101, 104, 103]

• ps = {"excellent", "good", "superb"}

"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

feedback words and the student IDs for tie-breaking.

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

"attention" is not in either set, t remains unchanged

ns = {"poor", "bad", "horrible"}

Solution Approach

• 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 feedback.

• 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.

Specifically, we generate a new list that contains just the student IDs of these top k elements.

if a word is either positive or negative, the operation is fast and efficient.

score of a student based on the positive and negative feedback words.

element (the student ID) in ascending order.

number of students and the length of their feedback reports.

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

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

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

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

o 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

• 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

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

• 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.

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"]

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.

For the second report "bad results but good effort" with student_id 101, the final score t will be 1. Append (1, 101).

Thus, the solution process allows us to rank the students by their feedback scores efficiently, taking into consideration both the

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

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)].

For every word in a student's report, check if it is in

Return a list of the top k student_ids with the highest scores

// Create an array to store scores and corresponding student IDs

// Check if the word is in positive feedback set

if (positiveFeedbackSet.contains(word)) {

int[][] scoresAndStudentIds = new int[numOfReports][0];

// Get the student ID for the current report

// Initialize score for the current report

for (var word : reports[i].split(" ")) {

positive feedback or negative feedback, and update total_score

Append the total_score and student_id as a tuple to the scores list

Sort the scores in descending order by score, and then by student_id for ties

public List<Integer> topStudents(String[] positiveFeedback, String[] negativeFeedback,

String[] reports, int[] studentIds, int k) {

reports: List[str], student_ids: List[int], 10 k: int,) -> List[int]: 11 12 # Convert lists of positive and negative feedback into sets for O(1) look-up time

positive_feedback: List[str],

negative_feedback: List[str],

positive_set = set(positive_feedback)

negative_set = set(negative_feedback)

for word in report.split():

if word in positive_set:

total_score += 3

total_score -= 1

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

return [score[1] for score in scores[:k]]

// Initialize the number of reports

for (int i = 0; i < numOfReports; ++i) {</pre>

int studentId = studentIds[i];

// Split the report into words

int numReports = reports.size();

for (auto& word : words) {

std::vector<int> topStudentsIds;

for (int i = 0; i < k; ++i) {

std::stringstream ss(str);

std::vector<std::string> result;

result.emplace_back(item);

std::string item;

return result;

negativeFeedback: string[],

const numStudents = studentIds.length;

// Iterate over each student's report

const score = reports[i]

return total;

}, 0);

})

which has this complexity.

for (let i = 0; i < numStudents; i++) {</pre>

.reduce((total, word) => {

return total + 3;

return total - 1;

scoresMap.set(studentIds[i], score);

return Array.from(scoresMap.entries())

if (scoreA === scoreB) {

return scoreB - scoreA;

const scoresMap = new Map<number, number>();

const positiveFeedbackSet = new Set(positiveFeedback);

const negativeFeedbackSet = new Set(negativeFeedback);

reports: string[],

k: number,

): number[] {

studentIds: number[],

int totalScore = 0;

for (int i = 0; i < numReports; ++i) {</pre>

int studentId = studentIds[i];

std::vector<std::string> words = split(reports[i], ' ');

totalScore += 3; // Add 3 for positive feedback

totalScore -= 1; // Subtract 1 for negative feedback

// Store the negative of the score for sorting in ascending order later

// Calculate the score for each word in the report

} else if (negativeFeedbackSet.count(word)) {

scoresAndIds.push_back({-totalScore, studentId});

topStudentsIds.emplace_back(scoresAndIds[i].second);

return topStudentsIds; // Return the top k students' IDs

std::vector<std::string> split(std::string& str, char delimiter) {

// Get the number of reports (and students as both should be equal)

// Calculate the score for the current student's report

.split(' ') // Split the report string into words

if (positiveFeedbackSet.has(word)) {

if (negativeFeedbackSet.has(word)) {

// in case of ties, and then extract only the student IDs

return studentIdA - studentIdB;

// Otherwise, sort by score descendingly

.sort(([studentIdA, scoreA], [studentIdB, scoreB]) => {

// If scores are equal, sort by student ID ascendingly

.slice(0, k); // Get the top 'k' student IDs based on their scores

// Initialize a Map to store the student IDs and their calculated scores

// Create Sets from the positive and negative feedback arrays for efficient lookup

// If the word is in the positive feedback set, add 3 to the total score

// If the word is neither positive nor negative, do not change the score

// Update the Map with the current student's id and their calculated score

// Convert the Map into an array, sort it based on scores and then student IDs

// If the word is in the negative feedback set, subtract 1 from the total score

// Sort the vector by score and then by student ID

// Utility function to split a string by a delimiter

while (std::getline(ss, item, delimiter)) {

std::sort(scoresAndIds.begin(), scoresAndIds.end());

if (positiveFeedbackSet.count(word)) {

int numOfReports = reports.length;

// Iterate over each report

elif word in negative_set:

scores.append((total_score, student_id))

Python Solution

class Solution:

13

14

15

22

23

24

26

28

29

30

31

32

33

34

35

36

37

38

10

11

12

13

14

15

16

17

18

19

20

21

22

24

25

26

27

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

57

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

53

52 }

56 };

from typing import List

def topStudents(

self,

16 scores = [] # Initialize an empty list to store tuples of scores and student IDs 17 18 # Iterate over the students' reports and their corresponding student IDs for student_id, report in zip(student_ids, reports): 19 total_score = 0 # Initialize total_score for the student 20 21

// Convert positive and negative feedbacks to sets for quick lookup Set<String> positiveFeedbackSet = new HashSet<>(Arrays.asList(positiveFeedback)); 8 Set<String> negativeFeedbackSet = new HashSet<>(Arrays.asList(negativeFeedback)); 9

Java Solution

import java.util.*;

class Solution {

```
28
                         score += 3; // Add 3 to the score for positive feedback
 29
                     } else if (negativeFeedbackSet.contains(word)) {
                         score -= 1; // Subtract 1 from the score for negative feedback
 30
 31
 32
 33
                 // Assign computed score and student ID to the scores array
                 scoresAndStudentIds[i] = new int[] {score, studentId};
 34
 35
 36
 37
             // Sort the array first by scores in descending order, and then by student IDs in ascending order
 38
             Arrays.sort(scoresAndStudentIds, (a, b) -> {
 39
                 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
 40
             });
 41
 42
 43
             // Initialize the list to store top k student IDs
 44
             List<Integer> topStudents = new ArrayList<>();
 45
 46
             // Extract the top k student IDs
 47
             for (int i = 0; i < k; ++i) {
 48
                 topStudents.add(scoresAndStudentIds[i][1]);
 49
 50
 51
             // Return the list of top k student IDs
 52
             return topStudents;
 53
 54
 55
C++ Solution
  1 #include <vector>
  2 #include <string>
  3 #include <unordered set>
    #include <sstream>
    #include <algorithm>
   class Solution {
    public:
  8
         std::vector<int> topStudents(std::vector<std::string>& positiveFeedback, std::vector<std::string>& negativeFeedback,
 10
                                      std::vector<std::string>& reports, std::vector<int>& studentIds, int k) {
 11
 12
             // 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

Typescript Solution function topStudents(positiveFeedback: string[],

```
Time and Space Complexity
Time Complexity
The time complexity of the provided code is 0(n * log n + (|ps| + |ns| + n) * |s|). To break it down:
  • zip(student_id, report) operation is 0(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 0(n

   * |s|), where n is the number of reports and |s| is the average length of a report since we consider |s| for the average length
   of words in a report.
```

.map(([studentId, _]) => studentId) // Extract only the student IDs from the sorted array

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

stored in these sets. • The arr list, which stores tuples of total score and student IDs for n students, contributes O(n) to space.

The presence checks in ps and ns is 0(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 0(n * log n) since Python's sort function uses TimSort,

Space for the sets ps and ns contributes |ps| * |s| and |ns| * |s|, respectively, because each word of average length |s| is