### 1604. Alert Using Same Key-Card Three or More Times in a One **Hour Period** Medium Array String Sorting Hash Table Leetcode Link

# The problem provides data on the use of key-cards by LeetCode company workers for unlocking office doors. The security system

Problem Description

we can follow these steps as part of our intuition towards the solution:

more times within any one-hour period during a single day. The task is to write a program that analyzes this data. You're given two lists: keyName, containing the names of the workers, and keyTime, containing the corresponding times when their key-cards were used. The program must output a list of unique names of workers who triggered an alert due to frequent use of their key-card within any given one-hour period. This output list should be sorted in alphabetical order.

logs the name of the worker and the time when the key-card was used. An alert is triggered if any worker uses the key-card three or

Access times in keyTime are in a 24-hour format, denoted as "HH:MM". It is important to note that overlapping hours, such as from "10:00" to "11:00", are considered a one-hour period, but times from "22:51" to "23:52" are not since they span into a new one-hour period. Intuition

To solve this problem, the primary goal is to track the usage frequency of key-cards by each worker within a one-hour period. Here,

1. Parse each time entry in keyTime from "HH:MM" format to minutes only. This makes it easier to calculate the difference between

2. Group all the time entries by worker names using a dictionary where the key is the worker's name and the value is a list of their

## corresponding times in minutes. 3. Iterate over each worker's list of time entries:

times.

 If a worker's list contains less than three time entries, no alert would be triggered for this worker, so they can be skipped. Sort the list of times for each worker, because to find a period of one hour where the card was used at least three times, the times need to be in order.

Look for any three consecutive time entries (after sorting) within a 60-minute window. If there are such entries, add the

- worker's name to the list of those who triggered an alert. 4. Finally, sort the list of workers who triggered an alert in alphabetical order and return it.
- This approach is efficient since it minimizes sorting to only the necessary times (those belonging to the same worker), instead of sorting all time entries globally. The lookup to check for the one-hour frequency is optimized by only comparing between three consecutive time entries due to the sorted nature of the lists.
- Solution Approach

The implementation of the solution given applies several common programming patterns and data structures to efficiently address

1. The defaultdict from Python's collections module is used to create a dictionary that will automatically generate a list for each

This conversion allows for much simpler comparisons since we can now just subtract the times to get the difference in minutes:

new key. This data structure helps group all the key-card use times by worker name. 2. The solution iterates through the combined keyName and keyTime lists using zip, which allows us to process the corresponding elements of these two lists in pairs. 3. Each time in the keyTime list is converted from the "HH:MM" string format to a total number of minutes using integer arithmetic.

the problem. Let's break down how the provided Python solution accomplishes this task:

# Here, t[:2] extracts the hours component and t[3:] extracts the minutes component from the string, which are then converted to an integer total number of minutes.

trio within the sorted list.

within a one-hour period:

names as required for the output.

1. Convert keyTime entries to minutes:

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

1 ["10:00", "10:20", "11:00", "12:00", "10:40"]

3 [600, 620, 660, 720, 640] # after conversion

entries, but Bob's do not, so Bob can be skipped.

5. For Alice, check three consecutive time entries:

name\_times\_dict = defaultdict(list)

Example Walkthrough

1 if ts[i + 2] - ts[i] <= 60:

1 t = int(t[:2]) \* 60 + int(t[3:])

4. For each worker, we sort their list of time entries, necessary to find the sequence of three time entries within a one-hour window (60 minutes). The sorting assures us that no three consecutive time entries can be more than one hour apart if there is no such

Python's Walrus operator := is used for assignment and comparison in a single expression. 6. To check for a one-hour period alert, a window of three consecutive times ts[i], ts[i + 1], and ts[i + 2] is examined. If the

difference between ts[i + 2] and ts[i] is less than or equal to 60 minutes, we confirm that there were at least three uses

5. The algorithm then iterates over each sorted time list for the workers, only if there are at least three time entries (n := len(ts)).

worker. 7. After all the workers have been checked, the list of workers who triggered an alert (ans) is sorted alphabetically to meet the specified output requirement.

8. The sorted list ans is returned, containing all the worker names that triggered an alert, sorted in ascending alphabetical order.

By using these steps, the algorithm is able to efficiently check each worker against the alert condition while maintaining the order of

If this condition is met, the worker's name is added to the ans list, and the loop breaks since one alert is sufficient to flag the

- Suppose keyName and keyTime are as follows: 1 keyName = ["Alice", "Bob", "Alice", "Alice", "Bob"] 2 keyTime = ["10:00", "10:20", "11:00", "12:00", "10:40"]
- 2. Group by worker names using defaultdict: 1 {"Alice": [600, 660, 720], "Bob": [620, 640]}

3. The times associated with Alice are [600, 660, 720], and for Bob, they are [620, 640]. Alice's times contain more than two

Since there are no three consecutive entries within a 60-minute window for Alice, she does not trigger an alert.

6. No alerts are triggered for any worker, so the final step of sorting the ans list is trivial as it is empty.

Hence, the output would be an empty list [], indicating that no worker triggered an alert in this example.

## 1 Between 600, 660 - Not within a one-hour window (difference is 60 minutes). 2 Between 660, 720 - Not within a one-hour window (difference is 60 minutes).

class Solution:

8

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

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

45

46

48

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

};

/\*\*

11

13

14

18

20

21

23

24

25

27

28

32

34

35

36

37

38

39

40

53

54

55

56

57

26 }

/\*\*

\*/

12 }

/\*\*

Typescript Solution

Python Solution 1 from collections import defaultdict

# Iterate through the dictionary to check for each person

if times[i + 2] - times[i] <= 60:</pre>

alert\_names.append(name)

# Sort the list of names alphabetically before returning

public List<String> alertNames(String[] keyNames, String[] keyTimes) {

// Iterate over the input arrays to populate the nameToTimesMap

// List to hold the names of individuals who should receive an alert

// We only care about individuals with more than 2 time logs

if  $(times.get(i + 2) - times.get(i) \le 60)$  {

for (Map.Entry<String, List<Integer>> entry : nameToTimesMap.entrySet()) {

// Go through the times to find a sequence of 3 within 60 minutes

Map<String, List<Integer>> nameToTimesMap = new HashMap<>();

for (int i = 0; i < keyNames.length; ++i) {</pre>

List<String> alerts = new ArrayList<>();

Collections.sort(times);

break;

List<Integer> times = entry.getValue();

for (int i = 0; i < n - 2; ++i) {

// Sort the names alphabetically before returning

// Sort the times to check the sequence

alerts.add(entry.getKey());

// Iterate through the map entries

int n = times.size();

if (n > 2) {

Collections.sort(alerts);

return alerts;

String name = keyNames[i];

String time = keyTimes[i];

int totalMinutes

// Map to track the time (converted to minutes) at which each keyName logs in

// Convert the time string to total minutes for easier comparison later

= Integer.parseInt(time.substring(0, 2)) \* 60 + Integer.parseInt(time.substring(3));

// If such a sequence is found, add the name to the alerts list and break

// If the name is not in the map, put it with a new ArrayList; add the time otherwise

nameToTimesMap.computeIfAbsent(name, k -> new ArrayList<>()).add(totalMinutes);

# Sort the times in ascending order

for i in range(times\_count - 2):

# Only proceed if there are more than two time points

for name, times in name\_times\_dict.items():

times\_count = len(times)

if times\_count > 2:

times.sort()

alert\_names.sort()

return alert\_names

2 import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

class Solution {

def alertNames(self, keyNames: List[str], keyTimes: List[str]) -> List[str]:

# Convert time strings to minutes and store them in the dictionary

# Create a dictionary to store times associated with each name

4. Sort the times for each worker. In this case, Alice's times are already in order.

- for name, time\_str in zip(keyNames, keyTimes): 9 time\_minutes = int(time\_str[:2]) \* 60 + int(time\_str[3:]) 10 name\_times\_dict[name].append(time\_minutes) 12 13 # Initialize a list to hold the names of individuals who should be alerted 14 alert\_names = []
- **Java Solution** 1 import java.util.ArrayList;

# Check for any three consecutive times within a 60 minutes window

break # No need to check further for this person

# If the condition is met, add the person's name to the alert list

```
49
50
  1 #include <vector>
  2 #include <string>
     #include <unordered_map>
     #include <algorithm>
     class Solution {
     public:
         vector<string> alertNames(vector<string>& keyName, vector<string>& keyTime) {
             // Map to store user names and their swipe times in minutes
  9
             unordered_map<string, vector<int>> nameTimeMap;
 10
 11
 12
             // Convert times into minutes and store them in the map
 13
             for (size_t i = 0; i < keyName.size(); ++i) {</pre>
 14
                 string name = keyName[i];
                 string time = keyTime[i];
 15
                 int hours, minutes;
 16
                 // Parse the time string to extract hours and minutes
 17
                 sscanf(time.c_str(), "%d:%d", &hours, &minutes);
 18
 19
                 // Calculate total time in minutes
 20
                 int totalMinutes = hours * 60 + minutes;
                 // Add the total time to the corresponding name in the map
 21
                 nameTimeMap[name].emplace_back(totalMinutes);
 23
 24
 25
             // List to hold the names of users who trigger the alert condition
 26
             vector<string> alertNamesList;
 27
 28
             // Iterate over each entry in the name time map
             for (auto& [name, timeStamps] : nameTimeMap) {
 29
 30
                 // Sort the timestamps for each user
 31
                 sort(timeStamps.begin(), timeStamps.end());
 32
                 // Check if the user has at least three timestamps
 33
                 if (timeStamps.size() > 2) {
 34
                     // Iterate through the timestamps
 35
                     for (size_t i = 0; i < timeStamps.size() - 2; ++i) {</pre>
 36
                         // Check if three consecutive swipes are within 60 minutes
 37
                         if (timeStamps[i + 2] - timeStamps[i] <= 60) {</pre>
```

// Add the name to the alert list and break the loop

alertNamesList.emplace\_back(name);

// Sort the list of names that trigger the alert condition

// Represents a map from user names to an array of their swipe times in minutes

sort(alertNamesList.begin(), alertNamesList.end());

break;

// Return the final list of alert names

const nameTimeMap: { [name: string]: number[] } = {};

\* @returns The number of minutes since midnight.

function convertTimeToMinutes(time: string): number {

\* Checks if the alert condition is met for a user.

\* @param time - A string representing the time in HH:MM format.

\* @param timeStamps - An array of swipe times in minutes for a user.

function processSwipeTimes(keyName: string[], keyTime: string[]): void {

const totalMinutes = convertTimeToMinutes(time);

\* @returns A boolean indicating if the alert condition is met.

function checkAlertCondition(timeStamps: number[]): boolean {

if (timeStamps[i + 2] - timeStamps[i] <= 60) {</pre>

for (let i = 0; i < timeStamps.length - 2; i++) {</pre>

\* @param keyTime - An array of corresponding key times.

const [hours, minutes] = time.split(':').map(Number);

return alertNamesList;

\* Converts a time string to minutes.

return hours \* 60 + minutes;

return true;

\* Adds swipe times to the name-time mapping.

for (let i = 0; i < keyName.length; ++i) {</pre>

nameTimeMap[name] = [];

processSwipeTimes(keyName, keyTime);

const alertNamesList: string[] = [];

for (const name in nameTimeMap) {

nameTimeMap[name].push(totalMinutes);

\* @param keyTime - An array of corresponding key times.

// Iterate over each entry in the name-time map

\* @returns An array of names for those who have triggered the alert.

51 function alertNames(keyName: string[], keyTime: string[]): string[] {

\* @param keyName - An array of key names.

const name = keyName[i];

const time = keyTime[i];

if (!nameTimeMap[name]) {

return false;

#### 42 43 44 /\*\* \* Generates a list of names who triggered the alert condition. \* @param keyName - An array of key names.

N elements, this is also O(N). 3. Sorting the times: The sort() operation on the list of times within the dictionary d can vary in its time complexity depending on the number of time stamps per name, which is less than or equal to N. In the worst case, if every name has N timestamps

where N is the number of entries in the keyName and keyTime lists.

 $O(K \log K)$  for each name and  $O(M * K \log K)$  in total.

**Space Complexity** 

- 4. Checking for 60-minute frames: The for loop runs a maximum of K 2 times per name, which is O(K), regarding the inner
- 5. Sorting the answer: Finally, sorting the list of names ans, which has at most M elements, will take O(M log M). Combining these steps, the dominating factor in time complexity is the sorting of timestamps, because K could be much larger than

1. Zipping and transforming times: The zip(keyName, keyTime) operation goes through all the keys and times once, which is O(N)

2. Time conversion: The conversion of times from strings to minutes is done for each element once and takes constant time, so for

associated with it and we assume M to represent the number of unique names, then the sorting can take O(N log N). However,

since every timestamp can only belong to one name, and assuming K average number of timestamps per name, sorting would be

The space complexity can be analyzed as follows: 1. Storing timestamps: The dictionary d can have at most M unique names with N total timestamps, so in the worst case, space

Hence, the total space complexity for the algorithm is O(N) because the space needed for the dictionary will be larger than or equal to the space for the list with all unique names.

C++ Solution

// Sort the timestamps for each user 58 nameTimeMap[name].sort((a, b) => a - b); // Check if the alert condition is met for the user if (checkAlertCondition(nameTimeMap[name])) { alertNamesList.push(name); 63 64 65 // Sort the list of names that triggered the alert condition alertNamesList.sort(); 67 68 69 return alertNamesList; 70 } 71 Time and Space Complexity Time Complexity The time complexity of the provided code can be analyzed in the following steps:

- operation as constant time. Multiplying by the number of unique names M, this gives us O(M \* K) for all names.
- log K. Therefore, if every person has many timestamps, the time complexity approaches O(N log N). Otherwise, it is more appropriately represented as  $0(M * K \log K + M \log M)$ .
- complexity for d is O(N). 2. Answer List: The ans list will contain at most M elements (as it only stores names), so the space needed is O(M).