Problem 1604: Alert Using Same Key-Card Three or More Times in a One Hour Period

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

LeetCode company workers use key-cards to unlock office doors. Each time a worker uses their key-card, the security system saves the worker's name and the time when it was used. The system emits an

alert

if any worker uses the key-card

three or more times

in a one-hour period.

You are given a list of strings

keyName

and

keyTime

where

[keyName[i], keyTime[i]]

corresponds to a person's name and the time when their key-card was used

in a

single day

.

Access times are given in the

24-hour time format "HH:MM"

, such as

"23:51"

and

"09:49"

.

Return a

list of unique worker names who received an alert for frequent keycard use

. Sort the names in

ascending order alphabetically

.

Notice that

"10:00"

-

"11:00"

is considered to be within a one-hour period, while

"22:51"

-

"23:52"

is not considered to be within a one-hour period.

Example 1:

Input:

keyName = ["daniel","daniel","daniel","luis","luis","luis","luis"], keyTime = ["10:00","10:40","11:00","09:00","11:00","13:00","15:00"]

Output:

["daniel"]

Explanation:

"daniel" used the keycard 3 times in a one-hour period ("10:00","10:40", "11:00").

Example 2:

Input:

keyName = ["alice","alice","alice","bob","bob","bob","bob"], keyTime = ["12:01","12:00","18:00","21:00","21:20","21:30","23:00"]

Output:

["bob"]

Explanation:

"bob" used the keycard 3 times in a one-hour period ("21:00","21:20", "21:30").

Constraints:

1 <= keyName.length, keyTime.length <= 10

5

keyName.length == keyTime.length

keyTime[i]

is in the format

"HH:MM"

.

[keyName[i], keyTime[i]]

is

unique

.

1 <= keyName[i].length <= 10

keyName[i] contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> alertNames(vector<string>& keyName, vector<string>& keyTime) {

}
};
```

Java:

```
class Solution {
public List<String> alertNames(String[] keyName, String[] keyTime) {

}
}
```

Python3:

```
class Solution:
def alertNames(self, keyName: List[str], keyTime: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
def alertNames(self, keyName, keyTime):
"""
:type keyName: List[str]
:type keyTime: List[str]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} keyName
* @param {string[]} keyTime
* @return {string[]}
*/
var alertNames = function(keyName, keyTime) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<string> AlertNames(string[] keyName, string[] keyTime) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** alertNames(char** keyName, int keyNameSize, char** keyTime, int
keyTimeSize, int* returnSize) {

}
```

Go:

```
func alertNames(keyName []string, keyTime []string) []string {

}
```

Kotlin:

```
class Solution {
fun alertNames(keyName: Array<String>, keyTime: Array<String>): List<String>
{

}
}
```

Swift:

```
class Solution {
func alertNames(_ keyName: [String], _ keyTime: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn alert_names(key_name: Vec<String>, key_time: Vec<String>) ->
```

```
Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} key_name
# @param {String[]} key_time
# @return {String[]}
def alert_names(key_name, key_time)

end
```

PHP:

```
class Solution {

/**
* @param String[] $keyName
* @param String[] $keyTime
* @return String[]
*/
function alertNames($keyName, $keyTime) {

}
}
```

Dart:

```
class Solution {
List<String> alertNames(List<String> keyName, List<String> keyTime) {

}
}
```

Scala:

```
object Solution {
def alertNames(keyName: Array[String], keyTime: Array[String]): List[String]
= {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec alert_names(key_name :: [String.t], key_time :: [String.t]) ::
[String.t]
def alert_names(key_name, key_time) do

end
end
```

Erlang:

```
-spec alert_names(KeyName :: [unicode:unicode_binary()], KeyTime ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
alert_names(KeyName, KeyTime) ->
.
```

Racket:

```
(define/contract (alert-names keyName keyTime)
(-> (listof string?) (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
```

```
public:
vector<string> alertNames(vector<string>& keyName, vector<string>& keyTime) {

}
};
```

Java Solution:

```
/**
* Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<String> alertNames(String[] keyName, String[] keyTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
Difficulty: Medium
Tags: array, string, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def alertNames(self, keyName: List[str], keyTime: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def alertNames(self, keyName, keyTime):
"""
:type keyName: List[str]
:type keyTime: List[str]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} keyName
* @param {string[]} keyTime
* @return {string[]}
*/
var alertNames = function(keyName, keyTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

};
```

C# Solution:

```
/*
* Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public IList<string> AlertNames(string[] keyName, string[] keyTime) {

}
}
```

C Solution:

```
/*
* Problem: Alert Using Same Key-Card Three or More Times in a One Hour Period
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** alertNames(char** keyName, int keyNameSize, char** keyTime, int
keyTimeSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Alert Using Same Key-Card Three or More Times in a One Hour
Period
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func alertNames(keyName []string, keyTime []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun alertNames(keyName: Array<String>, keyTime: Array<String>): List<String>
{

}
}
```

Swift Solution:

```
class Solution {
func alertNames(_ keyName: [String], _ keyTime: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Alert Using Same Key-Card Three or More Times in a One Hour
Period
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
impl Solution {
pub fn alert_names(key_name: Vec<String>, key_time: Vec<String>) ->
Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} key_name
# @param {String[]} key_time
# @return {String[]}
def alert_names(key_name, key_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $keyName
* @param String[] $keyTime
* @return String[]
*/
function alertNames($keyName, $keyTime) {

}
}
```

Dart Solution:

```
class Solution {
List<String> alertNames(List<String> keyName, List<String> keyTime) {

}
}
```

Scala Solution:

```
object Solution {
def alertNames(keyName: Array[String], keyTime: Array[String]): List[String]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec alert_names(key_name :: [String.t], key_time :: [String.t]) ::
[String.t]
def alert_names(key_name, key_time) do

end
end
```

Erlang Solution:

```
-spec alert_names(KeyName :: [unicode:unicode_binary()], KeyTime ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
alert_names(KeyName, KeyTime) ->
.
```

Racket Solution:

```
(define/contract (alert-names keyName keyTime)
(-> (listof string?) (listof string?) (listof string?))
)
```