Problem 2933: High-Access Employees

Problem Information

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

Problem Description

You are given a 2D

0-indexed

array of strings,

access_times

, with size

n

. For each

i

where

0 <= i <= n - 1

,

access_times[i][0]

represents the name of an employee, and

access_times[i][1]

represents the access time of that employee. All entries in

access_times

are within the same day.

The access time is represented as

four digits

using a

24-hour

time format, for example,

"0800"

or

"2250"

.

An employee is said to be

high-access

if he has accessed the system

three or more

times within a

one-hour period

.

Times with exactly one hour of difference are

not

considered part of the same one-hour period. For example,

"0815"

and

"0915"

are not part of the same one-hour period.

Access times at the start and end of the day are

not

counted within the same one-hour period. For example,

"0005"

and

"2350"

are not part of the same one-hour period.

Return

a list that contains the names of

high-access

employees with any order you want.

Example 1:

Input:

access_times = [["a","0549"],["b","0457"],["a","0532"],["a","0621"],["b","0540"]]

Output:

["a"]

Explanation:

"a" has three access times in the one-hour period of [05:32, 06:31] which are 05:32, 05:49, and 06:21. But "b" does not have more than two access times at all. So the answer is ["a"].

Example 2:

Input:

access_times = [["d","0002"],["c","0808"],["c","0829"],["e","0215"],["d","1508"],["d","1444"],["d","1410"],["c","0809"]]

Output:

["c","d"]

Explanation:

"c" has three access times in the one-hour period of [08:08, 09:07] which are 08:08, 08:09, and 08:29. "d" has also three access times in the one-hour period of [14:10, 15:09] which are 14:10, 14:44, and 15:08. However, "e" has just one access time, so it can not be in the answer and the final answer is ["c","d"].

Example 3:

Input:

access_times = [["cd","1025"],["ab","1025"],["cd","1046"],["cd","1055"],["ab","1124"],["ab","1120"]]

Output:

["ab","cd"]

Explanation:

"ab" has three access times in the one-hour period of [10:25, 11:24] which are 10:25, 11:20, and 11:24. "cd" has also three access times in the one-hour period of [10:25, 11:24] which are 10:25, 10:46, and 10:55. So the answer is ["ab","cd"].

Constraints:

1 <= access_times.length <= 100

access_times[i].length == 2

1 <= access_times[i][0].length <= 10

access_times[i][0]

consists only of English small letters.

access_times[i][1].length == 4

access_times[i][1]

is in 24-hour time format.

access_times[i][1]

consists only of

'0'

to

'9'

.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findHighAccessEmployees(vector<vector<string>>& access_times)
{

}
};
```

Java:

```
class Solution {
public List<String> findHighAccessEmployees(List<List<String>> access_times)
{

}
}
```

Python3:

```
class Solution:
def findHighAccessEmployees(self, access_times: List[List[str]]) ->
List[str]:
```

Python:

```
class Solution(object):
def findHighAccessEmployees(self, access_times):
"""
:type access_times: List[List[str]]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[][]} access_times
* @return {string[]}
*/
var findHighAccessEmployees = function(access_times) {
```

```
};
```

TypeScript:

```
function findHighAccessEmployees(access_times: string[][]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> FindHighAccessEmployees(IList<IList<string>>
access_times) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findHighAccessEmployees(char*** access_times, int access_timesSize,
int* access_timesColSize, int* returnSize) {

}
```

Go:

```
func findHighAccessEmployees(access_times [][]string) []string {

}
```

Kotlin:

```
class Solution {
fun findHighAccessEmployees(access_times: List<List<String>>): List<String> {

}
}
```

Swift:

```
class Solution {
func findHighAccessEmployees(_ access_times: [[String]]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_high_access_employees(access_times: Vec<Vec<String>>) ->
Vec<String> {

}
}
```

Ruby:

```
# @param {String[][]} access_times
# @return {String[]}
def find_high_access_employees(access_times)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $access_times
* @return String[]
*/
function findHighAccessEmployees($access_times) {

}
}
```

Dart:

```
class Solution {
List<String> findHighAccessEmployees(List<List<String>> access_times) {
```

```
}
}
```

Scala:

```
object Solution {
def findHighAccessEmployees(access_times: List[List[String]]): List[String] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec find_high_access_employees(access_times :: [[String.t]]) :: [String.t]
def find_high_access_employees(access_times) do

end
end
```

Erlang:

```
-spec find_high_access_employees(Access_times ::
[[unicode:unicode_binary()]]) -> [unicode:unicode_binary()].
find_high_access_employees(Access_times) ->
.
```

Racket:

```
(define/contract (find-high-access-employees access_times)
(-> (listof (listof string?)) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: High-Access Employees
```

```
* 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> findHighAccessEmployees(vector<vector<string>>& access_times)
{

}
};
```

Java Solution:

```
/**
* Problem: High-Access Employees
* 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> findHighAccessEmployees(List<List<String>> access_times)
{

}
}
```

Python3 Solution:

```
"""
Problem: High-Access Employees
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 findHighAccessEmployees(self, access_times: List[List[str]]) ->
List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findHighAccessEmployees(self, access_times):
"""
:type access_times: List[List[str]]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: High-Access Employees
* 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[][]} access_times
* @return {string[]}
*/
var findHighAccessEmployees = function(access_times) {

};
```

TypeScript Solution:

```
/**
* Problem: High-Access Employees
* 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 findHighAccessEmployees(access_times: string[][]): string[] {

};
```

C# Solution:

```
/*
* Problem: High-Access Employees
* 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> FindHighAccessEmployees(IList<IList<string>>
access_times) {

}
}
```

C Solution:

```
/*
* Problem: High-Access Employees
* 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** findHighAccessEmployees(char*** access_times, int access_timesSize,
int* access_timesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: High-Access Employees
// 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 findHighAccessEmployees(access_times [][]string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findHighAccessEmployees(access_times: List<List<String>>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func findHighAccessEmployees(_ access_times: [[String]]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: High-Access Employees
// 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 find_high_access_employees(access_times: Vec<Vec<String>>) ->
Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[][]} access_times
# @return {String[]}
def find_high_access_employees(access_times)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $access_times
* @return String[]
*/
function findHighAccessEmployees($access_times) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findHighAccessEmployees(List<List<String>> access_times) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def findHighAccessEmployees(access_times: List[List[String]]): List[String] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_high_access_employees(access_times :: [[String.t]]) :: [String.t]
def find_high_access_employees(access_times) do

end
end
```

Erlang Solution:

```
-spec find_high_access_employees(Access_times ::
[[unicode:unicode_binary()]]) -> [unicode:unicode_binary()].
find_high_access_employees(Access_times) ->
.
```

Racket Solution:

```
(define/contract (find-high-access-employees access_times)
(-> (listof (listof string?)) (listof string?))
)
```