Problem 1169: Invalid Transactions

Problem Information

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

Problem Description

A transaction is possibly invalid if:

the amount exceeds

$1000

, or;

if it occurs within (and including)

60

minutes of another transaction with the

same name

in a

different city

.

You are given an array of strings

transaction

where

transactions[i]

consists of comma-separated values representing the name, time (in minutes), amount, and city of the transaction.

Return a list of

transactions

that are possibly invalid. You may return the answer in

any order

.

Example 1:

Input:

transactions = ["alice,20,800,mtv","alice,50,100,beijing"]

Output:

["alice,20,800,mtv","alice,50,100,beijing"]

Explanation:

The first transaction is invalid because the second transaction occurs within a difference of 60 minutes, have the same name and is in a different city. Similarly the second one is invalid too.

Example 2:

Input:

transactions = ["alice,20,800,mtv","alice,50,1200,mtv"]

Output:

["alice,50,1200,mtv"]

Example 3:

Input:

transactions = ["alice,20,800,mtv","bob,50,1200,mtv"]

Output:

["bob,50,1200,mtv"]

Constraints:

transactions.length <= 1000

Each

transactions[i]

takes the form

"{name},{time},{amount},{city}"

Each

{name}

and

{city}

consist of lowercase English letters, and have lengths between

1

and

10

.

Each

{time}

consist of digits, and represent an integer between

0

and

1000

.

Each

{amount}

consist of digits, and represent an integer between

0

and

2000

.

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<string> invalidTransactions(vector<string>& transactions) {

}
};
```

Java:

```
class Solution {
public List<String> invalidTransactions(String[] transactions) {

}
}
```

Python3:

```
class Solution:
def invalidTransactions(self, transactions: List[str]) -> List[str]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function invalidTransactions(transactions: string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> InvalidTransactions(string[] transactions) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** invalidTransactions(char** transactions, int transactionsSize, int*
returnSize) {

}
```

Go:

```
func invalidTransactions(transactions []string) []string {

}
```

Kotlin:

```
class Solution {
fun invalidTransactions(transactions: Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func invalidTransactions(_ transactions: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn invalid_transactions(transactions: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} transactions
# @return {String[]}
def invalid_transactions(transactions)

end
```

PHP:

```
class Solution {

/**
* @param String[] $transactions
* @return String[]
*/
function invalidTransactions($transactions) {

}
}
```

Dart:

```
class Solution {
List<String> invalidTransactions(List<String> transactions) {

}
}
```

Scala:

```
object Solution {
def invalidTransactions(transactions: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec invalid_transactions(transactions :: [String.t]) :: [String.t]
def invalid_transactions(transactions) do

end
end
```

Erlang:

```
-spec invalid_transactions(Transactions :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
invalid_transactions(Transactions) ->
.
```

Racket:

```
(define/contract (invalid-transactions transactions)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Invalid Transactions
* 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> invalidTransactions(vector<string>& transactions) {

}
};
```

Java Solution:

```
/**
* Problem: Invalid Transactions
* 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> invalidTransactions(String[] transactions) {

}
}
```

Python3 Solution:

```
"""
Problem: Invalid Transactions
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 invalidTransactions(self, transactions: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Invalid Transactions
* 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[]} transactions
* @return {string[]}
*/
var invalidTransactions = function(transactions) {

};
```

TypeScript Solution:

```
/**
* Problem: Invalid Transactions
* 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 invalidTransactions(transactions: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Invalid Transactions
* 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> InvalidTransactions(string[] transactions) {

}
}
```

C Solution:

```
/*
* Problem: Invalid Transactions
* 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** invalidTransactions(char** transactions, int transactionsSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Invalid Transactions
// 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 invalidTransactions(transactions []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun invalidTransactions(transactions: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func invalidTransactions(_ transactions: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Invalid Transactions
// 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 invalid_transactions(transactions: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} transactions
# @return {String[]}
def invalid_transactions(transactions)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $transactions
* @return String[]
*/
function invalidTransactions($transactions) {

}
}
```

Dart Solution:

```
class Solution {
List<String> invalidTransactions(List<String> transactions) {

}
}
```

Scala Solution:

```
object Solution {
def invalidTransactions(transactions: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec invalid_transactions(transactions :: [String.t]) :: [String.t]
def invalid_transactions(transactions) do

end
```

```
end
```

Erlang Solution:

```
-spec invalid_transactions(Transactions :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
invalid_transactions(Transactions) ->
.
```

Racket Solution:

```
(define/contract (invalid-transactions transactions)
(-> (listof string?) (listof string?))
)
```