Problem 3606: Coupon Code Validator

Problem Information

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

Problem Description

You are given three arrays of length

n

that describe the properties of

n

coupons:

code

,

businessLine

, and

isActive

. The

i

th

coupon has:

code[i]

: a

string

representing the coupon identifier.

businessLine[i]

: a

string

denoting the business category of the coupon.

isActive[i]

: a

boolean

indicating whether the coupon is currently active.

A coupon is considered

valid

if all of the following conditions hold:

code[i]

is non-empty and consists only of alphanumeric characters (a-z, A-Z, 0-9) and underscores (

_

).

businessLine[i]

is one of the following four categories:

"electronics"

,

"grocery"

,

"pharmacy"

,

"restaurant"

.

isActive[i]

is

true

.

Return an array of the

codes

of all valid coupons,

sorted

first by their

businessLine

in the order:

"electronics"

,

"grocery"

,

"pharmacy", "restaurant"

, and then by

code

in lexicographical (ascending) order within each category.

Example 1:

Input:

code = ["SAVE20","","PHARMA5","SAVE@20"], businessLine = ["restaurant","grocery","pharmacy","restaurant"], isActive = [true,true,true,true]

Output:

["PHARMA5","SAVE20"]

Explanation:

First coupon is valid.

Second coupon has empty code (invalid).

Third coupon is valid.

Fourth coupon has special character

@

(invalid).

Example 2:

Input:

code = ["GROCERY15","ELECTRONICS_50","DISCOUNT10"], businessLine = ["grocery","electronics","invalid"], isActive = [false,true,true]

Output:

["ELECTRONICS_50"]

Explanation:

First coupon is inactive (invalid).

Second coupon is valid.

Third coupon has invalid business line (invalid).

Constraints:

n == code.length == businessLine.length == isActive.length

1 <= n <= 100

0 <= code[i].length, businessLine[i].length <= 100

code[i]

and

businessLine[i]

consist of printable ASCII characters.

isActive[i]

is either

true

or

false

.

Code Snippets

C++:

```
class Solution {
public:
vector<string> validateCoupons(vector<string>& code, vector<string>&
businessLine, vector<bool>& isActive) {

}
};
```

Java:

```
class Solution {
public List<String> validateCoupons(String[] code, String[] businessLine,
boolean[] isActive) {

}
}
```

Python3:

```
class Solution:
def validateCoupons(self, code: List[str], businessLine: List[str], isActive:
List[bool]) -> List[str]:
```

Python:

```
class Solution(object):
def validateCoupons(self, code, businessLine, isActive):
"""
:type code: List[str]
:type businessLine: List[str]
:type isActive: List[bool]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} code
* @param {string[]} businessLine
* @param {boolean[]} isActive
* @return {string[]}
*/
var validateCoupons = function(code, businessLine, isActive) {

};
```

TypeScript:

```
function validateCoupons(code: string[], businessLine: string[], isActive:
boolean[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> ValidateCoupons(string[] code, string[] businessLine,
bool[] isActive) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
char** validateCoupons(char** code, int codeSize, char** businessLine, int
businessLineSize, bool* isActive, int isActiveSize, int* returnSize) {

}
```

Go:

```
func validateCoupons(code []string, businessLine []string, isActive []bool)
[]string {

}
```

Kotlin:

```
class Solution {
fun validateCoupons(code: Array<String>, businessLine: Array<String>,
isActive: BooleanArray): List<String> {

}
}
```

Swift:

```
class Solution {
func validateCoupons(_ code: [String], _ businessLine: [String], _ isActive:
[Bool]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn validate_coupons(code: Vec<String>, business_line: Vec<String>,
is_active: Vec<bool>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} code
# @param {String[]} business_line
# @param {Boolean[]} is_active
# @return {String[]}
def validate_coupons(code, business_line, is_active)

end
```

PHP:

```
class Solution {

/**
* @param String[] $code
* @param String[] $businessLine
* @param Boolean[] $isActive
* @return String[]
*/
function validateCoupons($code, $businessLine, $isActive) {

}
}
```

Dart:

```
class Solution {
List<String> validateCoupons(List<String> code, List<String> businessLine,
List<bool> isActive) {

}
}
```

Scala:

```
object Solution {
def validateCoupons(code: Array[String], businessLine: Array[String],
isActive: Array[Boolean]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec validate_coupons(code :: [String.t], business_line :: [String.t],
is_active :: [boolean]) :: [String.t]
def validate_coupons(code, business_line, is_active) do

end
end
```

Erlang:

```
-spec validate_coupons(Code :: [unicode:unicode_binary()], BusinessLine ::
[unicode:unicode_binary()], IsActive :: [boolean()]) ->
[unicode:unicode_binary()].
validate_coupons(Code, BusinessLine, IsActive) ->
.
```

Racket:

```
(define/contract (validate-coupons code businessLine isActive)
(-> (listof string?) (listof string?) (listof boolean?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Coupon Code Validator
* Difficulty: Easy
* Tags: array, string, graph, 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> validateCoupons(vector<string>& code, vector<string>&
businessLine, vector<bool>& isActive) {
}
};
```

Java Solution:

```
/**
* Problem: Coupon Code Validator
* Difficulty: Easy
* Tags: array, string, graph, 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> validateCoupons(String[] code, String[] businessLine,
boolean[] isActive) {

}
}
```

Python3 Solution:

```
"""
Problem: Coupon Code Validator
Difficulty: Easy
Tags: array, string, graph, 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 validateCoupons(self, code: List[str], businessLine: List[str], isActive:
List[bool]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def validateCoupons(self, code, businessLine, isActive):
"""
:type code: List[str]
:type businessLine: List[str]
:type isActive: List[bool]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Coupon Code Validator
* Difficulty: Easy
* Tags: array, string, graph, 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[]} code
* @param {string[]} businessLine
* @param {boolean[]} isActive
* @return {string[]}
*/
var validateCoupons = function(code, businessLine, isActive) {

};
```

TypeScript Solution:

```
/**
* Problem: Coupon Code Validator
* Difficulty: Easy
* Tags: array, string, graph, 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 validateCoupons(code: string[], businessLine: string[], isActive:
boolean[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Coupon Code Validator
* Difficulty: Easy
* Tags: array, string, graph, 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> ValidateCoupons(string[] code, string[] businessLine,
bool[] isActive) {

}
}
```

C Solution:

```
/*
* Problem: Coupon Code Validator
* Difficulty: Easy
* Tags: array, string, graph, 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** validateCoupons(char** code, int codeSize, char** businessLine, int
businessLineSize, bool* isActive, int isActiveSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Coupon Code Validator
// Difficulty: Easy
// Tags: array, string, graph, 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 validateCoupons(code []string, businessLine []string, isActive []bool)
[]string {

}
```

Kotlin Solution:

```
class Solution {
fun validateCoupons(code: Array<String>, businessLine: Array<String>,
isActive: BooleanArray): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func validateCoupons(_ code: [String], _ businessLine: [String], _ isActive:
[Bool]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Coupon Code Validator
// Difficulty: Easy
// Tags: array, string, graph, 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 validate_coupons(code: Vec<String>, business_line: Vec<String>,
is_active: Vec<bool>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} code
# @param {String[]} business_line
# @param {Boolean[]} is_active
# @return {String[]}
def validate_coupons(code, business_line, is_active)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $code
* @param String[] $businessLine
* @param Boolean[] $isActive
* @return String[]
*/
function validateCoupons($code, $businessLine, $isActive) {

}
}
```

Dart Solution:

```
class Solution {
List<String> validateCoupons(List<String> code, List<String> businessLine,
List<bool> isActive) {

}
}
```

Scala Solution:

```
object Solution {
def validateCoupons(code: Array[String], businessLine: Array[String],
isActive: Array[Boolean]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec validate_coupons(code :: [String.t], business_line :: [String.t],
is_active :: [boolean]) :: [String.t]
def validate_coupons(code, business_line, is_active) do

end
end
```

Erlang Solution:

```
-spec validate_coupons(Code :: [unicode:unicode_binary()], BusinessLine ::
[unicode:unicode_binary()], IsActive :: [boolean()]) ->
[unicode:unicode_binary()].
validate_coupons(Code, BusinessLine, IsActive) ->
.
```

Racket Solution:

```
(define/contract (validate-coupons code businessLine isActive)
(-> (listof string?) (listof string?) (listof boolean?) (listof string?))
)
```