Problem 3078: Match Alphanumerical Pattern in Matrix I

Problem Information

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

Problem Description

You are given a 2D integer matrix

board

and a 2D character matrix

pattern

. Where

0 <= board[r][c] <= 9

and each element of

pattern

is either a digit or a lowercase English letter.

Your task is to find a

submatrix

of

board

that

matches

pattern

.

An integer matrix

part

matches

pattern

if we can replace cells containing letters in

pattern

with some digits (each

distinct

letter with a

unique

digit) in such a way that the resulting matrix becomes identical to the integer matrix

part

. In other words,

The matrices have identical dimensions.

If

pattern[r][c]

is a digit, then

part[r][c]

must be the

same

digit.

If

pattern[r][c]

is a letter

x

:

For every

pattern[i][j] == x

,

part[i][j]

must be the

same

as

part[r][c]

.

For every

pattern[i][j] != x

,

part[i][j]

must be

different

than

part[r][c]

.

Return

an array of length

2

containing the row number and column number of the upper-left corner of a submatrix of

board

which matches

pattern

. If there is more than one such submatrix, return the coordinates of the submatrix with the lowest row index, and in case there is still a tie, return the coordinates of the submatrix with the lowest column index. If there are no suitable answers, return

[-1, -1]

.

Example 1:

1

2

2

2

2

3

2

3

3

a

b

b

b

Input:

board = [[1,2,2],[2,2,3],[2,3,3]], pattern = ["ab","bb"]

Output:

[0,0]

Explanation:

If we consider this mapping:

"a" -> 1

and

"b" -> 2

; the submatrix with the upper-left corner

(0,0)

is a match as outlined in the matrix above.

Note that the submatrix with the upper-left corner (1,1) is also a match but since it comes after the other one, we return

[0,0]

.

Example 2:

1

1

2

3

3

4

6

6

6

a

b

6

6

Input:

board = [[1,1,2],[3,3,4],[6,6,6]], pattern = ["ab","66"]

Output:

[1,1]

Explanation:

If we consider this mapping:

"a" -> 3

and

"b" -> 4

; the submatrix with the upper-left corner

(1,1)

is a match as outlined in the matrix above.

Note that since the corresponding values of

"a"

and

"b"

must differ, the submatrix with the upper-left corner

(1,0)

is not a match. Hence, we return

[1,1]

.

Example 3:

1

2

2

1

x

x

Input:

board = [[1,2],[2,1]], pattern = ["xx"]

Output:

[-1,-1]

Explanation:

Since the values of the matched submatrix must be the same, there is no match. Hence, we return

[-1,-1]

.

Constraints:

1 <= board.length <= 50

1 <= board[i].length <= 50

0 <= board[i][j] <= 9

1 <= pattern.length <= 50

1 <= pattern[i].length <= 50

pattern[i][j]

is either a digit represented as a string or a lowercase English letter.

Code Snippets

C++:

```
class Solution {
public:
vector<int> findPattern(vector<vector<int>>& board, vector<string>& pattern)
{

}
};
```

Java:

```
class Solution {
public int[] findPattern(int[][] board, String[] pattern) {

}
}
```

Python3:

```
class Solution:
def findPattern(self, board: List[List[int]], pattern: List[str]) ->
List[int]:
```

Python:

```
class Solution(object):
def findPattern(self, board, pattern):
"""
:type board: List[List[int]]
:type pattern: List[str]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} board
* @param {string[]} pattern
* @return {number[]}
*/
var findPattern = function(board, pattern) {

};
```

TypeScript:

```
function findPattern(board: number[][], pattern: string[]): number[] {

};
```

C#:

```
public class Solution {
public int[] FindPattern(int[][] board, string[] pattern) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPattern(int** board, int boardSize, int* boardColSize, char**
pattern, int patternSize, int* returnSize) {

}
```

Go:

```
func findPattern(board [][]int, pattern []string) []int {

}
```

Kotlin:

```
class Solution {
fun findPattern(board: Array<IntArray>, pattern: Array<String>): IntArray {

}
}
```

Swift:

```
class Solution {
func findPattern(_ board: [[Int]], _ pattern: [String]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_pattern(board: Vec<Vec<i32>>, pattern: Vec<String>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} board
# @param {String[]} pattern
# @return {Integer[]}
```

```
def find_pattern(board, pattern)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $board
* @param String[] $pattern
* @return Integer[]
*/
function findPattern($board, $pattern) {

}
}
```

Dart:

```
class Solution {
List<int> findPattern(List<List<int>> board, List<String> pattern) {

}
}
```

Scala:

```
object Solution {
def findPattern(board: Array[Array[Int]], pattern: Array[String]): Array[Int]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_pattern(board :: [[integer]], pattern :: [String.t]) :: [integer]
def find_pattern(board, pattern) do

end
```

```
end
```

Erlang:

```
-spec find_pattern(Board :: [[integer()]], Pattern ::
[unicode:unicode_binary()]) -> [integer()].
find_pattern(Board, Pattern) ->
.
```

Racket:

```
(define/contract (find-pattern board pattern)
(-> (listof (listof exact-integer?)) (listof string?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Match Alphanumerical Pattern in Matrix I
* Difficulty: Medium
* Tags: array, string, hash
*
* 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<int> findPattern(vector<vector<int>>& board, vector<string>& pattern)
{

}
};
```

Java Solution:

```
/**
* Problem: Match Alphanumerical Pattern in Matrix I
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 int[] findPattern(int[][] board, String[] pattern) {

}
}
```

Python3 Solution:

```
"""
Problem: Match Alphanumerical Pattern in Matrix I
Difficulty: Medium
Tags: array, string, hash

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 findPattern(self, board: List[List[int]], pattern: List[str]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findPattern(self, board, pattern):
"""
:type board: List[List[int]]
:type pattern: List[str]
:rtype: List[int]
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Match Alphanumerical Pattern in Matrix I
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 {number[][]} board
* @param {string[]} pattern
* @return {number[]}
*/
var findPattern = function(board, pattern) {

};
```

TypeScript Solution:

```
/**
* Problem: Match Alphanumerical Pattern in Matrix I
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 findPattern(board: number[][], pattern: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Match Alphanumerical Pattern in Matrix I
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 int[] FindPattern(int[][] board, string[] pattern) {

}
}
```

C Solution:

```
/*
* Problem: Match Alphanumerical Pattern in Matrix I
* Difficulty: Medium
* Tags: array, string, hash
*
* 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().
*/
int* findPattern(int** board, int boardSize, int* boardColSize, char**
pattern, int patternSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Match Alphanumerical Pattern in Matrix I
// Difficulty: Medium
// Tags: array, string, hash
//
```

```
// 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 findPattern(board [][]int, pattern []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findPattern(board: Array<IntArray>, pattern: Array<String>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func findPattern(_ board: [[Int]], _ pattern: [String]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Match Alphanumerical Pattern in Matrix I
// Difficulty: Medium
// Tags: array, string, hash
//
// 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_pattern(board: Vec<Vec<i32>>, pattern: Vec<String>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} board
# @param {String[]} pattern
# @return {Integer[]}
def find_pattern(board, pattern)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $board
* @param String[] $pattern
* @return Integer[]
*/
function findPattern($board, $pattern) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findPattern(List<List<int>> board, List<String> pattern) {

}
}
```

Scala Solution:

```
object Solution {
def findPattern(board: Array[Array[Int]], pattern: Array[String]): Array[Int]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_pattern(board :: [[integer]], pattern :: [String.t]) :: [integer]
```

```
def find_pattern(board, pattern) do

end
end
```

Erlang Solution:

```
-spec find_pattern(Board :: [[integer()]], Pattern ::
[unicode:unicode_binary()]) -> [integer()].
find_pattern(Board, Pattern) ->
.
```

Racket Solution:

```
(define/contract (find-pattern board pattern)
(-> (listof (listof exact-integer?)) (listof string?) (listof
exact-integer?))
)
```