766 Toeplitz Matrix

2018年5月4日 16:28

A matrix is *Toeplitz* if every diagonal from top-left to bottom-right has the same element.
Now given an M x N matrix, return True if and only if the matrix is *Toeplitz*.

Example 1:
Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
Output: True
Explanation:
1234
5123
9512
In the above grid, the diagonals are "[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]", and in each diagonal all elements are the same, so the answer is True.
Example 2:
Input: matrix = [[1,2],[2,2]]
Output: False
Explanation:
The diagonal "[1, 2]" has different elements.
Note:

1. matrix will be a 2D array of integers.
2. matrix will have a number of rows and columns in range [1, 20].
3. matrix[i][j] will be integers in range [0, 99].

来自 <https://leetcode.com/problems/toeplitz-matrix/description/>

如果一个矩阵的每一方向由左上到右下的对角线上具有相同元素，那么这个矩阵是*托普利茨矩阵*。
给定一个 M x N 的矩阵，当且仅当它是*托普利茨矩阵*时返回 True。
示例 1:
输入: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
输出: True
解释:
1234
5123
9512
在上面这个矩阵中, 对角线分别是 "[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]", 各条对角线上的所有元素都相同, 因此答案是True。
示例 2:
输入: matrix = [[1,2],[2,2]]
输出: False
解释:
对角线, 比如: "[1, 2]" 上有不同的元素。
注意:

1. matrix (矩阵)是一个包含整数的二维数组。
2. matrix 的行数和列数均在 [1, 20]范围内。
3. matrix[i][j] 包含的整数在 [0, 99]范围内。

Solution for Python3:

```python
class Solution1:
    def isToeplitzMatrix(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: bool
        """
        r, c = len(matrix), len(matrix[0])
        for k in range(r):
           i, j = k + 1, 1
           while i < r and j < c:
               if matrix[i][j] != matrix[k][0]:
                   return False
               i += 1
               j += 1
        for k in range(1, c):
           i, j = 1, k + 1
           while i < r and j < c:
               if matrix[i][j] != matrix[0][k]:
                   return False
               i += 1
               j += 1
        return True

class Solution2:
    def isToeplitzMatrix(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: bool
        """
        groups = {}
        for r, row in enumerate(matrix):
           for c, val in enumerate(row):
               if r - c not in groups:
                   groups[r-c] = val
```

```python
                elif groups[r-c] != val:
                    return False
        return True

class Solution3:
    def isToeplitzMatrix(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: bool
        """
        return all(r == 0 or c == 0 or matrix[r-1][c-1] == val for r, row in enumerate(matrix) for c, val in enumerate(row))

```

Solution for C++:

```cpp
class Solution1 {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        int i = 0, j = 0, r = matrix.size(), c = matrix[0].size();
        for (int k = 0; k < r; k++) {
            i = k + 1, j = 0 + 1;
            while (i < r && j < c) {
                if (matrix[i][j] != matrix[k][0])
                    return false;
                i++;
                j++;
            }
        }
        for (int k = 1; k < c; k++) {
            i = 0 + 1, j = k + 1;
            while (i < r && j < c) {
                if (matrix[i][j] != matrix[k][0])
                    return false;
                i++;
                j++;
            }
        }
        return true;
    }
};

class Solution2 {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        unordered_map<int, int> groups;
        for (int r = 0; r < matrix.size(); r++) {
            for (int c = 0; c < matrix[0].size(); c++) {
                if (groups.find(r-c) == groups.end())
                    groups[r-c] = matrix[r][c];
                else if (groups[r-c] != matrix[r][c])
                    return false;
            }
        }
        return true;
    }
};

class Solution3 {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        for (int r = 0; r < matrix.size(); r++) {
            for (int c = 0; c < matrix[0].size(); c++) {
                if (r > 0 && c > 0 && matrix[r-1][c-1] != matrix[r][c])
                    return false;
            }
        }
        return true;
    }
};
```