# 766. Toeplitz Matrix

Easy Array Matrix

## Problem Description

Given an `m x n` matrix, the task is to determine if the matrix is a Toeplitz matrix. A matrix is considered Toeplitz if every diagonal from the top-left to the bottom-right corners has the same elements. This means that if you pick any element and look at the elements diagonally down and to the right, all those elements should be identical. This must hold true for all the elements in the matrix, except for those on the last row and the rightmost column since they do not have any diagonal elements.

To conceptualize this, imagine a 3×3 matrix as an example:

```
a, b, c
d, a, b
e, d, a
```

This is a Toeplitz matrix because the diagonals (`(a, a, a)`, `(b, b)`, and `(c)`, along with `(d, d)` and `(e)`) from the top-left to the bottom right consist of the same elements.

The problem requires a function that takes such a matrix as input and returns `true` if the matrix is Toeplitz, and `false` otherwise.

## Intuition

To check if a matrix is Toeplitz, we only need to verify that each element is equal to the one directly diagonally up and to the left of it. This is because if each element has this property, then by transitivity, all elements on the diagonal will be equivalent.

We start checking from the second row and the second column since the elements in the first row and the first column don't have any diagonal predecessors.

The idea is to iterate through the matrix while comparing each element `(i, j)` with the element `(i - 1, j - 1)`. If all such pairs of elements are equal, we can conclude that the matrix is Toeplitz. Otherwise, if we find any pair of elements that do not match, we immediately know the matrix is not Toeplitz, and we can return `false`.

## Solution Approach

The solution uses a simple iteration pattern to walk through the elements of the matrix and a logical test to validate the Toeplitz condition.

### Implementation Details:

1. **Data Structures**: The primary data structure used is the input matrix itself, which is a 2D list, denoted as `matrix`.
2. **Variables**: Two variables `m` and `n` are used to store the dimensions of the matrix.
3. **For Loops**: Two nested `for` loops are constructed, where `i` ranges from `1` to `m - 1` and `j` ranges from `1` to `n - 1`. This makes sure we're starting from the second row and column.
4. **Comparison**: Inside the loops, we compare each element `matrix[i][j]` with the element `matrix[i - 1][j - 1]`. This is done with a simple equality check: `matrix[i][j] == matrix[i - 1][j - 1]`.
5. **Generator Expression**: This comparison is encapsulated within a generator expression which efficiently creates an iterator for each element's comparison and `for` loop iteration.
6. **`all()` Function**: The generator expression is fed into Python's built-in `all()` function. The `all()` function is perfect for this scenario because it requires all values from the iterator to be `True` for the entire expression to evaluate as `True`. If there is even one `False`, it returns `False`. This aligns exactly with our requirement: all comparisons must be `True` for the matrix to be Toeplitz.

The `return all(...)` line in the code is where the entire logic is applied. It checks all the pairs (continuing until either end of the matrix is reached) and confirms if every required element satisfies the Toeplitz condition. This one-liner leverages Python's readability and built-in functions to implement an elegant and scalable solution.

The code elegantly handles the problem with a high level of efficiency, as it stops checking as soon as one pair of elements fails the test, due to the lazy evaluation property of the generator expression used in the `all()` function. This means that the function will have early termination and will not perform unnecessary checks once a non-Toeplitz pair is found.

### Example Walkthrough

To illustrate the solution approach, let's consider a small 3×3 matrix:

```
1, 2, 3
4, 1, 2
5, 4, 1
```

We are tasked with determining if this is a Toeplitz matrix.

1. **Initialization**: According to our solution approach, we need the dimensions of the matrix. For this 3×3 matrix, `m` is 3 (number of rows) and `n` is 3 (number of columns).
2. **Iteration**: We set up two nested `for` loops. The outer loop iterates over `i` from 1 to 2 (`m - 1`), which corresponds to the second and third rows. The inner loop iterates over `j` from 1 to 2 (`n - 1`), covering the second and third columns.
3. **Comparison and Early Termination**: We compare the elements starting at `matrix[1][1]`:
   - For `i = 1, j = 1`: We check if `matrix[1][1]` is equal to `matrix[0][0]`. For our matrix, it checks if 1 equals 1, which is true.
   - For `i = 1, j = 2`: We check if `matrix[1][2]` is equal to `matrix[0][1]`. It checks if 2 equals 2, which is true.
   - For `i = 2, j = 1`: We check if `matrix[2][1]` is equal to `matrix[1][0]`. It checks if 4 equals 4, which is true.
   - For `i = 2, j = 2`: We check if `matrix[2][2]` is equal to `matrix[1][1]`. It checks if 1 equals 1, which is true.

   All comparisons are true, which means each pair of compared elements satisfies the Toeplitz condition.
4. **Generator Expression and `all()` Function**: The generator expression is as follows:

   ```
   (matrix[i][j] == matrix[i - 1][j - 1] for i in range(1, m) for j in range(1, n))
   ```

   In our case, this generator will create an iterator `(True, True, True, True)` from the above comparisons.

   The `all()` function takes this iterator as an input and returns `True` because all elements are `True`. If there was any `False` in the iterator, `all()` would return `False`, indicating the matrix is not Toeplitz.

Therefore, for this given 3×3 matrix, the function concludes with `True`, confirming that it is indeed a Toeplitz matrix.

## Solution Implementation

### Python

```python
from typing import List

class Solution:
    def is_toeplitz_matrix(self, matrix: List[List[int]]) -> bool:
        """
        Check if a matrix is a Toeplitz matrix.
        A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.

        :param matrix: 2D List[int], the input matrix to check
        :return: bool, True if the matrix is Toeplitz, False otherwise
        """

        # Number of rows in the matrix
        row_count = len(matrix)
        # Number of columns in the matrix
        col_count = len(matrix[0])

        # Iterate over each element in the matrix starting from the second row and second column
        # because the comparison starts with the element just above and to the left (i-1, j-1).
        for i in range(1, row_count):
            for j in range(1, col_count):
                # Compare the current element with the element diagonally above and to the left.
                # If any such comparison fails, the matrix is not Toeplitz.
                if matrix[i][j] != matrix[i - 1][j - 1]:
                    return False

        # If all diagonal comparisons hold, the matrix is Toeplitz.
        return True
```

### Java

```java
class Solution {
    /**
     * Checks if a given matrix is a Toeplitz matrix.
     * A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.
     *
     * @param matrix The input matrix to check.
     * @return true if the matrix is Toeplitz; otherwise, false.
     */
    public boolean isToeplitzMatrix(int[][] matrix) {
        // m is the number of rows in the matrix
        int numRows = matrix.length;
        // n is the number of columns in the matrix
        int numCols = matrix[0].length;

        // Start from the second row and column because we will be comparing with the element above and to the left
        for (int i = 1; i < numRows; ++i) {
            for (int j = 1; j < numCols; ++j) {
                // If the current element is not the same as the one above and to the left, return false
                if (matrix[i][j] != matrix[i - 1][j - 1]) {
                    return false;
                }
            }
        }

        // If all diagonals from top-left to bottom-right have the same elements, return true
        return true;
    }
}
```

### C++

```cpp
#include <vector>

// Class to define the solution.
class Solution {
public:
    // Function to check whether a given matrix is a Toeplitz matrix.
    // A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.
    // @param matrix: 2D vector of ints representing the input matrix.
    // @return: boolean value indicating whether the matrix is Toeplitz.
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        // Get the number of rows in the matrix.
        int numRows = matrix.size();
        // Get the number of columns in the matrix.
        int numCols = matrix[0].size();

        // Start from the second row and second column as the first row and column
        // do not have a top-left element to be compared with.
        for (int row = 1; row < numRows; ++row) {
            for (int col = 1; col < numCols; ++col) {
                // Compare the current element with the top-left neighbor.
                // If they are not the same, the matrix is not Toeplitz, return false.
                if (matrix[row][col] != matrix[row - 1][col - 1]) {
                    return false;
                }
            }
        }

        // If no discrepancies are found, return true indicating it is a Toeplitz matrix.
        return true;
    }
};
```

### TypeScript

```typescript
/**
 * Function to check if a matrix is a Toeplitz matrix.
 * A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.
 * @param matrix A 2D array of numbers representing the matrix.
 * @returns true if the matrix is Toeplitz, otherwise false.
 */
function isToeplitzMatrix(matrix: number[][]): boolean {
    // Get the number of rows in the matrix
    const rowCount: number = matrix.length;
    // Get the number of columns in the matrix by checking the first row
    const columnCount: number = matrix[0].length;

    // Start from the first row (skipping the first element) and check all diagonals
    for (let rowIndex = 1; rowIndex < rowCount; ++rowIndex) {
        for (let columnIndex = 1; columnIndex < columnCount; ++columnIndex) {
            // If the current element is not equal to the element in the previous row and previous column,
            // it's not a Toeplitz matrix
            if (matrix[rowIndex][columnIndex] !== matrix[rowIndex - 1][columnIndex - 1]) {
                return false;
            }
        }
    }

    // If all diagonals have the same elements, return true
    return true;
}

// Example usage:
// const matrix: number[][] = [
//     [1, 2, 3, 4],
//     [5, 1, 2, 3],
//     [9, 5, 1, 2]
// ];
// console.log(isToeplitzMatrix(matrix)); // Should log true
```

```python
from typing import List

class Solution:
    def is_toeplitz_matrix(self, matrix: List[List[int]]) -> bool:
        """
        Check if a matrix is a Toeplitz matrix.
        A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.

        :param matrix: 2D List[int], the input matrix to check
        :return: bool, True if the matrix is Toeplitz, False otherwise
        """

        # Number of rows in the matrix
        row_count = len(matrix)
        # Number of columns in the matrix
        col_count = len(matrix[0])

        # Iterate over each element in the matrix starting from the second row and second column
        # because the comparison starts with the element just above and to the left (i-1, j-1).
        for i in range(1, row_count):
            for j in range(1, col_count):
                # Compare the current element with the element diagonally above and to the left.
                # If any such comparison fails, the matrix is not Toeplitz.
                if matrix[i][j] != matrix[i - 1][j - 1]:
                    return False

        # If all diagonal comparisons hold, the matrix is Toeplitz.
        return True
```

## Time and Space Complexity

The time complexity of the given code is `O(m * n)` where `m` is the number of rows in the matrix and `n` is the number of columns. This is because the code iterates through each element in the matrix, starting from the second row and the second column, checking the top-left diagonal element for each cell.

The space complexity of the code is `O(1)` meaning it does not allocate any additional space that scales with the input size. The use of variables `m` and `n` and the iterative checks on the matrix elements do not require additional space beyond the input itself.