2711. Difference of Number of Distinct Values on Diagonals Medium Array Hash Table Matrix

Problem Description

and n columns, and we want to create a new 2D array answer of the same dimensions. For each cell in the answer matrix at position (r, c), the value is determined as follows:

The task is to compute a matrix answer based on another 2D grid given as input. The grid is a two-dimensional array with m rows

Leetcode Link

• topLeft[r][c]: count the number of distinct values that appear in the cells on the top-left diagonal of cell (r, c) in the original

- grid. This diagonal includes cells (x, y) from the top row or the leftmost column cutting through to just above (r, c) such that
- x < r and y < c. bottomRight[r][c]: count the number of distinct values that appear in the cells on the bottom-right diagonal of cell (r, c) in the
- such that x > r and y > c. The value in answer[r][c] is the absolute difference between these two counts: |topLeft[r][c] - bottomRight[r][c]|.

original grid. This diagonal includes cells (x, y) starting just below (r, c) running down to the bottom right corner of the grid

grid.

The problem requires returning the matrix answer, which is constructed by performing these computations for every cell (r, c) of

The intuition behind the solution is to focus on each cell (r, c) individually and to accumulate the distinct values along its top-left

and bottom-right diagonals. Using sets is helpful because sets naturally maintain unique elements, thereby making the counting of

distinct values straightforward. For the top-left diagonal, we start from the current cell (r, c) and move upwards and to the left (x - 1, y - 1) until we either

Intuition

reach the first row or the first column. For the bottom-right diagonal, we start from the current cell (r, c) and move downwards and to the right (x + 1, y + 1) until we either reach the last row or the last column.

At each step for both diagonals, we add the values of grid[x][y] to their respective sets. Once all unique values are accumulated,

the sizes of the sets represent the count of distinct values in each diagonal. We calculate the absolute difference of these counts and assign it to answer[r][c].

This process is repeated for every cell in the grid, which eventually yields the completed answer matrix. The brute force nature of this solution is acceptable given the problem constraints, and it directly translates the problem's requirements into algorithmic steps to find the solution.

The solution as implemented above employs a straightforward, brute-force approach. To compute the answer[r][c], the algorithm inspects two distinct diagonals for each cell (r, c) in the grid. Here is a step-by-step breakdown of the approach: 1. Initialize a matrix ans with the same dimensions as grid but filled with zeros. This will store the final results.

2. Traverse every cell (r, c) in the grid. Use nested loops, the outer loop running through the rows, and the inner loop going

3. To find topLeft[r][c], initialize an empty set s. Then, iterate from the current cell upwards and leftward diagonally (decreasing both row and column indices). For each cell encountered, add the value of the cell from grid[x][y] to the set s. The loop stops

3 while x and y:

6 tl = len(s)

1 x, y = i, j

6 br = len(s)

2 s = set()

4 x, y = x - 1, y - 1

3 while x + 1 < m and y + 1 < n:

x, y = x + 1, y + 1

6. After filling in all values, return the ans matrix.

duplicates, simplifying the code and ensuring accuracy.

5 s.add(grid[x][y])

5 s.add(grid[x][y])

through the columns.

Solution Approach

unique values in the top-left diagonal. $1 \times , y = i, j$ 2 s = set()

and rightward diagonally (increasing both row and column indices). Add the value from grid[x][y] to set s until reaching the

when reaching the top row or the leftmost column. After iteration, tl is set as the length of the set s, which is the count of

4. Repeat a similar process to find bottomRight[r][c]. Initialize another empty set s, and iterate from the current cell downwards

bottom row or the rightmost column. Assign br as the length of this set.

the ans matrix. 1 ans[i][j] = abs(tl - br)

Using this method, every cell in the grid is visited, and its corresponding diagonals are processed to determine the values of

answer[r][c]. The utilization of sets for counting unique elements is pivotal because it eliminates the need for manual checks for

5. Calculate the absolute difference between tl and br and assign it to ans[i][j]. This step executes for every cell (r, c), filling

Example Walkthrough Let's go through a small example to illustrate the solution approach.

top-left and bottom-right diagonals.

1 grid = [

[1, 2, 3],

[4, 1, 6],

[7, 8, 1]

Step-by-step Process:

[0, 0, 0]

Suppose we have the following grid matrix:

1. Initialize an answer matrix with zeros having the same dimensions as grid: 1 answer = [[0, 0, 0], [0, 0, 0],

5. Continue the process for each cell, calculating the unique counts on both diagonals and computing their absolute difference.

along the top-left and bottom-right diagonals for each cell and calculating the absolute difference between these unique counts for

We want to construct an answer matrix where each element is the absolute difference between the count of distinct numbers in its

3. Move to cell (0, 1):

[0, 0, 0],

[0, 0, 0]

Update the answer matrix:

[1, 2, 0],

[0, 1, 0],

[0, 0, 0]

[1, 1, 0],

[1, 0, 0]

the answer matrix.

8

9

10

11

12

13

14

15

16

17

18

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

40

10

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

37

38

39

40

41

42

43

44

46

47

48

49

50

51

39 }

42 # Notes:

Python Solution

class Solution:

Get the grid dimensions.

for row in range(rows):

return answer_grid

rows, cols = len(grid), len(grid[0])

Iterate over each cell in the grid.

 x_{up} , $y_{up} = row$, col

while x_up > 0 and y_up > 0:

int[][] answerGrid = new int[rowCount][colCount];

// Iterate over each cell in the grid.

for (int j = 0; j < colCount; ++j) {</pre>

for (int i = 0; i < rowCount; ++i) {</pre>

// Return the filled answer grid.

return answerGrid;

row += direction;

col += direction;

return distinctValues.size();

 x_{up} , $y_{up} = x_{up} - 1$, $y_{up} - 1$

while x down + 1 < rows and y down + 1 < cols:

count_downright = len(unique_values_downright)

 x_{down} , $y_{down} = x_{down} + 1$, $y_{down} + 1$

unique_values_downright.add(grid[x_down][y_down])

answer_grid[row][col] = abs(count_upleft - count_downright)

// Compute the distinct count in the top-left diagonal direction.

// Helper method to calculate the distinct count in a diagonal direction.

// Add the current value to the set of distinct values.

// Move in the diagonal direction specified by 'direction'.

// Return the size of the set, i.e., the number of distinct values.

vector<vector<int>> differenceOfDistinctValues(vector<vector<int>>& grid) {

// Traverse the top-left diagonal from the current cell

// Traverse the bottom-right diagonal from the current cell

// Store the absolute difference of the counts in the answer grid

for (int x = i, y = j; $x < rows && y < cols; ++x, ++y) {$

// Count distinct values in the top-left diagonal from the current cell

// Count distinct values in the bottom-right diagonal from the current cell

for (int x = i, y = j; x >= 0 && y >= 0; --x, --y) {

topLeftValues.insert(grid[x][y]);

int topLeftCount = topLeftValues.size();

bottomRightValues.insert(grid[x][y]);

int bottomRightCount = bottomRightValues.size();

// Count distinct values in the bottom-right diagonal

const bottomRightDistinctCount = bottomRightSet.size;

// Return the answer grid with the differences

// and assign it to the corresponding cell in the answer grid

// Calculate the absolute difference of distinct values in both diagonals

answerGrid[i][j] = Math.abs(topLeftDistinctCount - bottomRightDistinctCount);

answer[i][j] = abs(topLeftCount - bottomRightCount);

int rows = grid.size(); // Number of rows in the input grid

vector<vector<int>> answer(rows, vector<int>(cols));

// Iterate over every cell in the grid

for (int j = 0; j < cols; ++j) {

for (int i = 0; i < rows; ++i) {

int cols = grid[0].size(); // Number of columns in the input grid

// Create a set to keep track of the distinct values seen.

Set<Integer> distinctValues = new HashSet<>();

distinctValues.add(grid[row][col]);

private int calculateDistinctCount(int[][] grid, int row, int col, int direction) {

while (row $>= 0 \&\& col >= 0 \&\& row < grid.length \&\& col < grid[0].length) {$

// Continue moving in the diagonal direction until the grid boundaries are reached.

int distinctCountTopLeft = calculateDistinctCount(grid, i, j, -1);

// Compute the distinct count in the bottom-right diagonal direction.

int distinctCountBottomRight = calculateDistinctCount(grid, i, j, 1);

answerGrid[i][j] = Math.abs(distinctCountTopLeft - distinctCountBottomRight);

// Compute the absolute difference of the distinct counts in both diagonal directions.

Count the unique values in the bottom-right diagonal.

Return the answer grid populated with differences of distinct values.

for col in range(cols):

answer_grid = [[0] * cols for _ in range(rows)]

1 answer = [

1 answer = [

[1, 0, 0],

[0, 0, 0],

[0, 0, 0]

1 answer = [

 Update the answer matrix: 1 answer = [[1, 2, 0],

The bottom-right diagonal has 1 and 6, so bottomRight = 2.

2. Start iterating over each cell (r, c) of the grid. First with cell (0, 0):

For the bottom-right diagonal, only 1 is on the diagonal, so bottomRight = 1.

There is no top-left diagonal, so topLeft = 0.

Again, no top-left diagonal, so topLeft = 0.

 \circ answer[0][0] = |0 - 1| = 1

 \circ answer[0][1] = |0 - 2| = 2

 \circ answer[1][1] = |2 - 1| = 1.

Update the answer matrix:

4. Inspect the cell (1, 1): The top-left diagonal contains 4 and 1 (so topLeft[r][c] = 2). The bottom-right diagonal contains only 1, hence bottomRight[r][c] = 1.

After completing this process for all cells, we would obtain the final answer matrix:

def differenceOfDistinctValues(self, grid: List[List[int]]) -> List[List[int]]:

Initialize the answer grid with zeros, matching the input grid dimensions.

Initialize variables for traversing diagonally up-left.

Now we'll do the same for cell (1,1), which has both top-left and bottom-right diagonals:

Each step of the process follows the explanation in the solution approach, utilizing the property of sets to count unique elements

19 unique_values_upleft.add(grid[x_up][y_up]) 20 21 # Count the unique values in the top-left diagonal. 22 count_upleft = len(unique_values_upleft) 23 # Reset the variables for traversing diagonally down-right. 24 25 x_down, y_down = row, col

Traverse diagonally to the bottom-right corner, collecting unique values.

Calculate the absolute difference and update the value in the answer grid.

unique_values_downright = set() # To store unique values seen so far down-right.

unique_values_upleft = set() # To store unique values seen so far up-left.

Traverse diagonally to the top-left corner, collecting unique values.

43 # 1. The variable names have been improved for readability. 44 # 2. Added comments to explain each section of code. 45 # 3. The code logic and method names are unchanged since the request was only to rewrite the code with better variable naming and a 46 # 4. Used zero-indexed while loops for traversing the diagonals to correctly access the grid indices. 47 Java Solution class Solution { public int[][] differenceOfDistinctValues(int[][] grid) { // Get the number of rows and columns in the grid. int rowCount = grid.length, colCount = grid[0].length; // Initialize the answer grid with the same dimensions.

13 14 15 16 17

C++ Solution

1 class Solution {

2 public:

```
34
           return answer; // Return the final grid containing absolute differences
35
36 };
37
Typescript Solution
    function differenceOfDistinctValues(grid: number[][]): number[][] {
         // Get the dimensions of the grid
         const rows = grid.length;
         const columns = grid[0].length;
  5
  6
         // Initialize an answer grid of the same dimensions with zeroes
         const answerGrid: number[][] = Array.from({ length: rows }, () => Array(columns).fill(0));
  8
         // Iterate over each cell of the grid
  9
         for (let i = 0; i < rows; ++i) {
 10
 11
             for (let j = 0; j < columns; ++j) {</pre>
 12
                 // Initialize row and column index for top-left diagonal
 13
                 let rowTopLeft = i;
 14
                 let colTopLeft = j;
 15
                 // Create a set to store unique values in the top-left diagonal
 16
 17
                 const topLeftSet = new Set<number>();
 18
 19
                 // Traverse the top-left diagonal
 20
                 while (rowTopLeft > 0 && colTopLeft > 0) {
                     topLeftSet.add(grid[--rowTopLeft][--colTopLeft]);
 21
 22
 23
 24
                 // Count distinct values in the top-left diagonal
 25
                 const topLeftDistinctCount = topLeftSet.size;
 26
 27
                 // Initialize row and column index for bottom-right diagonal
 28
                 let rowBottomRight = i;
 29
                 let colBottomRight = j;
 30
 31
                 // Create a set to store unique values in the bottom-right diagonal
 32
                 const bottomRightSet = new Set<number>();
 33
                 // Traverse the bottom-right diagonal
 34
 35
                 while (rowBottomRight + 1 < rows && colBottomRight + 1 < columns) {</pre>
 36
                     bottomRightSet.add(grid[++rowBottomRight][++colBottomRight]);
```

unordered_set<int> topLeftValues; // Set to store distinct values in the top-left diagonal

unordered_set<int> bottomRightValues; // Set to store distinct values in the bottom-right diagonal

The time complexity of the code provided is primarily determined by the two nested loops, each iterating over the dimensions of the grid, and the two while loops inside the nested loops. The outer loops run m * n times where m is the number of rows and n is the number of columns in the grid (m and n can be considered separately if not square). The inner while loops iterate at worst from 0 to i

Time Complexity

return answerGrid;

Time and Space Complexity

Space Complexity The space complexity is 0 (m * n) due to the auxiliary space required for the ans list which is the same size as the grid. Additionally, the set s takes up space, but since it is overwritten in each iteration of the loops rather than accumulated, the maximum space it uses at any given time is the size of the largest number of distinct elements on a diagonal, which is at most min(m, n). This does not change the overall space complexity, which remains 0(m * n).

complexity is roughly 0((m * n) * (m + n)), which can also be represented as $0(m^2 * n + m * n^2)$ when m and n are different.

for the top-left diagonal elements, and from i to m for the bottom-right diagonal elements (and similarly for j and n). This gives us

corresponds to the sum of two arithmetic series, which is effectively 0(m + n) for each outer loop iteration. Therefore, the total time

two arithmetic progressions counting down and up respectively. The sum of operations due to these internal while loops