Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Computer Programming

Laboratory work #3

Two-Dimensional Array Operations and Processing

Author:
Alexandru Rudoi
std. gr. FAF-231

Verified:
Alexandru Furdui

Chișinău 2023

Theory Background

2D Arrays: A two-dimensional array, often referred to as a 2D array, is a fundamental data structure that extends the concept of a one-dimensional array. In contrast to a one-dimensional array, which stores elements in a linear, sequential fashion, a 2D array organizes elements in a grid-like structure with rows and columns, forming a matrix. This grid-like structure is particularly useful for representing and manipulating data in a tabular format, making it suitable for a wide range of applications, including tables, grids, and matrices. Accessing elements within a 2D array involves the use of two indices: one to specify the row and another to specify the column.

Square Matrix Properties: A square matrix is a specific type of 2D array where the number of rows (N) is equal to the number of columns (M). These matrices hold a unique place in mathematics and applications due to their balanced structure. Square matrices are widely used in various mathematical operations, including linear algebra and transformations. Their balanced nature simplifies many calculations.

Matrix Manipulation Algorithms: Matrix manipulation algorithms encompass a set of operations used to transform matrices. These operations include transposition, where rows become columns and vice versa, rotation, reshaping, and more. Transposition, in particular, is a fundamental operation that allows for the reorganization of data within a matrix, and it is often used in mathematical and computational tasks.

Recursive Algorithms: Recursive algorithms are a powerful problem-solving approach that involves breaking down complex problems into smaller, similar subproblems. In the context of working with matrices, recursive algorithms are employed when tasks require rearranging matrix elements recursively, such as creating patterns or performing calculations on submatrices. Recursive approaches leverage the principle of solving a problem by solving smaller instances of the same problem.

Mountain Patterns: In matrix manipulation, creating a mountain pattern means arranging elements within the matrix in a specific way. In this pattern, the largest value is centered, and the values decrease as you move away from the center in all directions. This concept can be visualized as a peak at the center of the matrix, with values gradually diminishing as you move toward the edges. Achieving this pattern often involves a systematic rearrangement of elements.

Boundary Conditions: Boundary conditions refer to the considerations when working with matrices, particularly at the edges. These conditions are crucial to prevent index out-of-bounds errors and ensure the correct handling of elements located on the matrix's perimeter. Decisions must be made on how to manage the manipulation of elements near the edges to maintain the desired patterns or calculations.

Complexity Analysis: Analyzing the time and space complexity of an algorithm is a critical step in understanding its efficiency and performance characteristics. When working with matrices, the algorithm's efficiency can be influenced by the matrix's size (N, M). Striving for efficient algorithms that can handle matrices of significant dimensions without excessive computational demands is essential for practical applications.

The Task

Read n and m variables from console, then a 2D table of size n x m also from the console. You are given a square matrix of size N x M. Your task is to rearrange(recursively) the elements of the matrix in such a way that it forms a mountain pattern with the largest value in the center and decreasing values as you move away from the center in all directions. For this problem, you will have to use N, M ¿ 10(a good idea to implement a matrix generator as well).

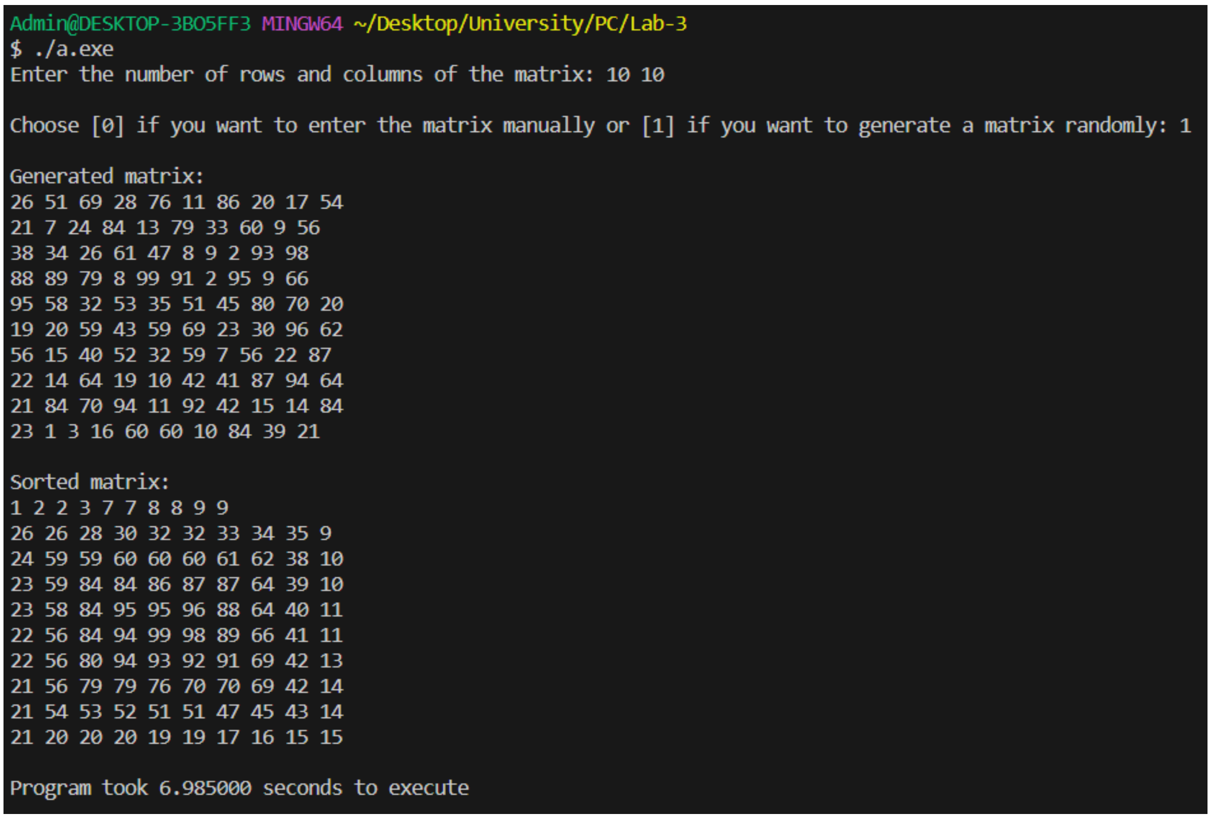
Technical implementation

Listing of the program: Github
Pseudo-code:

```
Function generateMatrix(n, m, arr)
    For i = 0 to n-1
        For j = 0 to m-1
            arr[i][j] = Random(1, 100)

Function createMatrix(n, m, arr)
    Do
        Print "Choose␣[0]␣if␣you␣want␣to␣enter␣the␣matrix␣
           manually␣or␣[1]␣if␣you␣want␣to␣generate␣a␣matrix␣
           randomly:␣"
        Read k
    While k is not 0 and k is not 1

    If k is 1 Then
        Seed Random Number Generator with current time
        Call generateMatrix(n, m, arr)
        Print "Generated␣matrix:"
        Call printMatrix(n, m, arr)
    Else
        Print "Enter␣the␣elements␣of␣the␣matrix:"
        For i = 0 to n-1
            For j = 0 to m-1
                Read arr[i][j]

Function printMatrix(n, m, arr)
    For i = 0 to n-1
        For j = 0 to m-1
            Print arr[i][j], "␣"
        End For
        Print NewLine
    End For

Function bubbleSort(v, n)
    For i = 0 to n-2
        For j = 0 to n-2-i
            If v[j] > v[j+1] Then
                Swap v[j] and v[j+1]
            End If
        End For
    End For

Function MatrixToVector(arr, n, m, v)
    k = 0
    For i = 0 to n-1
        For j = 0 to m-1
            v[k] = arr[i][j]
            k = k + 1

Function ToSpiral(m, n, v, arr, index, row, col)
    If row >= m Or col >= n Then
        Return
    End If

    For i = col to n-1
        arr[row][i] = v[index]
        index = index + 1
    End For
    row = row + 1
    For i = row to m-1
        arr[i][n-1] = v[index]
        index = index + 1
    End For
    n = n - 1

    If row < m Then
        For i = n-1 to col Step -1
            arr[m-1][i] = v[index]
            index = index + 1
        End For
        m = m - 1
    End If

    If col < n Then
        For i = m-1 to row Step -1
            arr[i][col] = v[index]
            index = index + 1
        End For
        col = col + 1

    Call ToSpiral(m, n, v, arr, index, row, col)

Main()
    start = CurrentTime()

    Print "Enter␣the␣number␣of␣rows␣and␣columns␣of␣the␣matrix
       :"
    Read n, m

    Declare arr as a 2D array of size n x m

    Call createMatrix(n, m, arr)

    Declare v as an array of size n * m

    Call MatrixToVector(arr, n, m, v)
    Call bubbleSort(v, n * m)

    Call ToSpiral(n, m, v, arr, 0, 0, 0)

    Print "Sorted␣matrix:"
    Call printMatrix(n, m, arr)

    Free memory for arr

    end = CurrentTime()
    cpu_time_used = end - start
    Print "Program␣took", cpu_time_used, "seconds␣to␣execute"
```

Results

In the next picture, the program displays a matrix that was initially generated with random numbers in a 10x10 grid. It then sorts the numbers in ascending order to create a sorted spiral matrix. The output also includes the time it took for the program to execute, which was approximately 6.985 seconds.



```
Admin@DESKTOP-3BO5FF3 MINGW64 ~/Desktop/University/PC/Lab-3
$ ./a.exe
Enter the number of rows and columns of the matrix: 10 10

Choose [0] if you want to enter the matrix manually or [1] if you want to generate a matrix randomly: 1

Generated matrix:
26 51 69 28 76 11 86 20 17 54
21 7 24 84 13 79 33 60 9 56
38 34 26 61 47 8 9 2 93 98
88 89 79 8 99 91 2 95 9 66
95 58 32 53 35 51 45 80 70 20
19 20 59 43 59 69 23 30 96 62
56 15 40 52 32 59 7 56 22 87
22 14 64 19 10 42 41 87 94 64
21 84 70 94 11 92 42 15 14 84
23 1 3 16 60 60 10 84 39 21

Sorted matrix:
1 2 2 3 7 7 8 8 9 9
26 26 28 30 32 32 33 34 35 9
24 59 59 60 60 60 61 62 38 10
23 59 84 84 86 87 87 64 39 10
23 58 84 95 95 96 88 64 40 11
22 56 84 94 99 98 89 66 41 11
22 56 80 94 93 92 91 69 42 13
21 56 79 79 76 70 70 69 42 14
21 54 53 52 51 51 47 45 43 14
21 20 20 20 19 19 17 16 15 15

Program took 6.985000 seconds to execute
```

Conclusion

In summary, delving into the intricacies of 2D arrays, sorting algorithms, and dynamic memory management in this project has marked a pivotal moment in my journey toward becoming a future software engineer. It has expanded my understanding of efficient data manipulation, sorting techniques, and memory allocation within the context of 2D matrices. This experience has equipped me with essential skills and insights that will undoubtedly benefit my future endeavors in software engineering, especially when dealing with complex data structures and algorithmic challenges.

Bibliography

1. The overview of Computer Programming course lessons for students (lecturer: associate professor M. Kulev). Chisinau, UTM, FCIM, 2023.
2. https://www.geeksforgeeks.org/matrix/
3. https://www.geeksforgeeks.org/bubble-sort/
4. https://www.scaler.com/topics/c/recursion-in-c/
5. https://www.geeksforgeeks.org/sort-the-given-array-in-spiral-manner-starting-from-the-center/