Print a given matrix in spiral form - GeeksforGeeks

Source: https://www.geeksforgeeks.org/print-a-given-matrix-in-spiral-form/

Print a given matrix in spiral form Last Updated : 17 Jul, 2025 Given a matrix mat[][] of size m x n , the task is to print all elements of the matrix in spiral form. Examples: Input: mat[][] = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]] Output: [1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10] Explanation: Input: mat[][]= [[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12], [13, 14, 15, 16, 17, 18]] Output: [1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11] Try it on GfG Practice Table of Content [Approach] Using Simulation by Visited Matrix - O(m*n) Time and O(m*n) Space [Expected Approach] Using Boundary Traversal - O(m*n) Time and O(1) Space [Approach] Using Simulation by Visited Matrix - O(m*n) Time and O(m*n) Space The idea is to simulate the spiral traversal by marking the cells that have already been visited. We can use a direction array to keep track of the movement (right, down, left, up) and change direction when we hit the boundary or a visited cell. Step By Step Implementations: Initialize a 2D array vis[][] to keep track of visited cells. Use direction arrays dr and dc to represent right, down, left, and up movements. Start from the top-left cell and follow the direction arrays to visit each cell. Change direction when encountering a boundary or a visited cell. Continue until all cells are visited. C++ #include <iostream> #include <vector> using namespace std ; vector < int > spirallyTraverse (vector < vector < int >>& mat) { int m = mat . size (); int n = mat [0]. size (); vector < int > res ; vector < vector < bool >> vis (m , vector < bool > (n , false)); // Arrays to represent the changes // in row and column indices: // move right(0, +1), move down(+1, 0), // move left(0, -1), move up(-1, 0) // Change in row index for each direction vector < int > dr = { 0 , 1 , 0 , -1 }; // Change in column index for each direction vector < int > dc = { 1 , 0 , -1 , 0 }; // Initial position in the matrix int r = 0 , c = 0 ; // Initial direction index (0 corresponds to 'right') int idx = 0 ; for (int i = 0 ; i < m * n ; ++ i) { res . push_back (mat [r][c]); vis [r][c] = true ; // Calculate the next cell coordinates based on // current direction int newR = r + dr [idx]; int newC = c + dc [idx]; // Check if the next cell is within bounds and not // visited if (0 <= newR && newR < m && 0 <= newC && newC < n && ! vis [newR][newC]) { // Move to the next row r = newR ; // Move to the next column c = newC ; } else { // Change direction (turn clockwise) idx = (idx + 1) % 4 ; // Move to the next row according to new // direction r += dr [idx]; // Move to the next column according to new // direction c += dc [idx]; } } return res ; } int main () { vector < vector < int >> mat = { { 1 , 2 , 3 , 4 }, { 5 , 6 , 7 , 8 }, { 9 , 10 , 11 , 12 }, { 13 , 14 , 15 , 16 } }; vector < int > res = spirallyTraverse (mat); for (int num : res) { cout << num << " " ; } return 0 ; } Java import java.util.ArrayList ; import java.util.List ; class GfG { static ArrayList < Integer > spirallyTraverse (int [][] mat) { int m = mat . length ; int n = mat [0] . length ; ArrayList < Integer > res = new ArrayList <> (); boolean [][] vis = new boolean [m][n] ; // Change in row index for each direction int [] dr = { 0 , 1 , 0 , - 1 }; // Change in column index for each direction int [] dc = { 1 , 0 , - 1 , 0 }; // Initial position in the matrix int r = 0 , c = 0 ; // Initial direction index (0 corresponds to 'right') int idx = 0 ; for (int i = 0 ; i < m * n ; ++ i) { // Add current element to result list res . add (mat [r][c]); // Mark current cell as visited vis [r][c] = true ; // Calculate the next cell coordinates based on // current direction int newR = r + dr [idx] ; int newC = c + dc [idx] ; // Check if the next cell is within bounds and not // visited if (0 <= newR && newR < m && 0 <= newC && newC < n && ! vis [newR][newC]) { // Move to the next row r = newR ; // Move to the next column c = newC ; } else { // Change direction (turn clockwise) idx = (idx + 1) % 4 ; // Move to the next row according to new // direction r += dr [idx] ; // Move to the next column according to new // direction c += dc [idx] ; } } return res ; } public static void main (String [] args) { int [][] mat = { { 1 , 2 , 3 , 4 }, { 5 , 6 , 7 , 8 }, { 9 , 10 , 11 , 12 }, { 13 , 14 , 15 , 16 } }; ArrayList < Integer > res = spirallyTraverse

(mat); for (int num : res) { System . out . print (num + " "); } } } Python def spirallyTraverse (mat): m = len (mat) n = len (mat [0]) res = [] vis = [[False] * n for _ in range (m)] # Change in row index for each direction dr = [0 , 1 , 0 , - 1] # Change in column index for each direction dc = [1 , 0 , - 1 , 0] # Initial position in the matrix r , c = 0 , 0 # Initial direction index (0 corresponds to 'right') idx = 0 for _ in range (m * n): res . append (mat [r][c]) vis [r][c] = True # Calculate the next cell coordinates based on # current direction newR , newC = r + dr [idx], c + dc [idx] # Check if the next cell is within bounds and not # visited if 0 <= newR < m and 0 <= newC < n and not vis [newR][newC]: # Move to the next row r , c = newR , newC else : # Change direction (turn clockwise) idx = (idx + 1) % 4 # Move to the next row according to new # direction r += dr [idx] # Move to the next column according to new # direction c += dc [idx] return res if __name__ == "__main__" : mat = [[1 , 2 , 3 , 4], [5 , 6 , 7 , 8], [9 , 10 , 11 , 12], [13 , 14 , 15 , 16]] res = spirallyTraverse (mat) print (" " . join (map (str , res))) C# using System ; using System.Collections.Generic ; class GfG { static List < int > spirallyTraverse (int [][] mat) { int m = mat . Length ; int n = mat [0]. Length ; List < int > res = new List < int > (); bool [,] vis = new bool [m , n]; // Arrays to represent the changes in row and column // indices: turn right(0, +1), turn down(+1, 0), // turn left(0, -1), turn up(-1, 0) int [] dr = { 0 , 1 , 0 , - 1 }; int [] dc = { 1 , 0 , - 1 , 0 }; // Initial position in the matrix int r = 0 , c = 0 ; // Initial direction index (0 corresponds to // 'right') int idx = 0 ; for (int i = 0 ; i < m * n ; ++ i) { res . Add (mat [r][c]); vis [r , c] = true ; // Calculate the next cell coordinates based on // current direction int newR = r + dr [idx]; int newC = c + dc [idx]; // Check if the next cell is within bounds and // not visited if (newR >= 0 && newR < m && newC >= 0 && newC < n && ! vis [newR , newC]) { r = newR ; c = newC ; } else { // Change direction (turn clockwise) idx = (idx + 1) % 4 ; r += dr [idx]; c += dc [idx]; } } return res ; } static void Main () { int [][] mat = { new int [] { 1 , 2 , 3 , 4 }, new int [] { 5 , 6 , 7 , 8 }, new int [] { 9 , 10 , 11 , 12 }, new int [] { 13 , 14 , 15 , 16 } }; List < int > res = spirallyTraverse (mat); foreach (int num in res) Console . Write (num + " "); } } JavaScript function spirallyTraverse (mat) { const m = mat . length ; const n = mat [0]. length ; const res = []; const vis = Array . from ({ length : m }, () => Array (n). fill (false)); // Arrays to represent the changes in row and column // indices: turn right(0, +1), turn down(+1, 0), turn // left(0, -1), turn up(-1, 0) const dr = [0 , 1 , 0 , - 1]; const dc = [1 , 0 , - 1 , 0]; // Initial position in the matrix let r = 0 , c = 0 ; // Initial direction index (0 corresponds to 'right') let idx = 0 ; for (let i = 0 ; i < m * n ; ++ i) { // Add current element to result array res . push (mat [r][c]); // Mark current cell as visited vis [r][c] = true ; // Calculate the next cell coordinates based on // current direction const newR = r + dr [idx]; const newC = c + dc [idx]; // Check if the next cell is within bounds and not // visited if (newR >= 0 && newR < m && newC >= 0 && newC < n && ! vis [newR][newC]) { r = newR ; c = newC ; } else { // Change direction (turn clockwise) idx = (idx + 1) % 4 ; r += dr [idx]; c += dc [idx]; } } return res ; } // Driver Code const mat = [[1 , 2 , 3 , 4], [5 , 6 , 7 , 8], [9 , 10 , 11 , 12], [13 , 14 , 15 , 16]]; const res = spirallyTraverse (mat); console . log (res . join (" ")); Output 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 [Expected Approach] Using Boundary Traversal - O(m*n) Time and O(1) Space We can print the matrix in a spiral order by dividing it into loops or boundaries. We print the elements of the outer boundary first, then move inward to print the elements of the inner boundaries. Algorithm: Define the boundaries of the matrix with variables top, bottom, left, and right. Print the top row from left to right and increment top . Print the right column from top to bottom and decrement right . Check if boundaries have crossed; if not, print the bottom row from right to left and decrement bottom . Print the left column from bottom to top and increment left . Repeat until all boundaries are crossed. Illustration: C++ #include <iostream> #include <vector> using namespace std ; vector < int > spirallyTraverse (vector < vector < int >> & mat) { int m = mat . size (); int n = mat [0]. size (); vector < int > res ; // Initialize boundaries int top = 0 , bottom = m - 1 , left = 0 , right = n - 1 ; // Iterate until all elements are stored while (top <= bottom && left <= right) { // store top row from left to right for (int i = left ; i <= right ; ++ i) { res . push_back (mat [top][i]); } top ++ ; // store right column from top to bottom for (int i = top ; i <= bottom ; ++ i) { res . push_back (mat [i][right]); } right -- ; // store bottom row from right to left (if exists) if (top <= bottom) { for (int i = right ; i >= left ; -- i) { res . push_back (mat [bottom][i]); } bottom -- ; } // store left column from bottom to top (if exists) if (left <= right) { for (int i = bottom ; i >= top ; -- i) { res . push_back (mat [i][left]); } left ++ ; } } return res ; } int main () { vector < vector < int >> mat = {{ 1 , 2 , 3 , 4 }, { 5 , 6 , 7 , 8 }, { 9 , 10 , 11 , 12 }, { 13 , 14 , 15 , 16 } }; vector < int > res = spirallyTraverse (mat); for (int ele : res) cout << ele << " " ; return 0 ; } Java import java.util.ArrayList ; class GfG { static ArrayList < Integer > spirallyTraverse (int [][] mat) { int m = mat . length ; int n = mat [0] . length ; ArrayList < Integer > res = new ArrayList <> (); // Initialize boundaries int top = 0 , bottom = m - 1 , left = 0 , right = n - 1 ; // Iterate until all elements are printed while (top <= bottom && left <= right) { // Print top row from left to right for (int i = left ; i <= right ; ++ i) { res . add (mat [top][i]); } top ++ ; // Print right column from top to bottom for (int i = top ; i <= bottom ; ++ i) { res . add (mat [i][right]); } right -- ; // Print bottom row from right to left (if exists) if (top <= bottom) { for (int i = right ; i >= left ; -- i) { res . add (mat [bottom][i]); } bottom -- ; } // Print left column from bottom to top (if exists) if (left <= right) { for (int i = bottom ; i >= top ; -- i) { res . add (mat [i][left]); } left ++ ; } } return res ; } // Driver Code public static void main (String [] args) { int [][] mat = { { 1 , 2 , 3 , 4 }, { 5 , 6 , 7 , 8 }, { 9 , 10 , 11 , 12 }, { 13 , 14 , 15 , 16 } }; ArrayList < Integer > res = spirallyTraverse (mat); for (int ele : res) { System . out . print (ele + " "); } } } Python def spirallyTraverse (mat): m , n = len (mat), len (mat [0]) res = [] # Initialize boundaries top , bottom , left , right = 0 , m - 1 , 0 , n - 1 # Iterate until all elements are printed while top <= bottom and left <= right : # Print top row from left to right for i in range (left , right + 1): res . append (mat [top][i]) top += 1 # Print right column from top to bottom for i in range (top , bottom + 1): res . append (mat [i][right]) right -= 1 # Print bottom row from right to left (if exists) if top <= bottom : for i in range (right , left - 1 , - 1): res . append (mat [bottom][i]) bottom -= 1 # Print left column from bottom to top (if exists) if left <= right : for i in range (bottom , top - 1 , - 1): res . append (mat [i][left]) left += 1 return res if __name__ == "__main__" : mat = [[1 , 2 , 3 , 4], [5 , 6 , 7 , 8], [9 , 10 , 11 , 12], [13 , 14 , 15 , 16]] res = spirallyTraverse (mat) print (" " . join (map (str , res))) C# using System ; using System.Collections.Generic ; class GfG { static List < int > spirallyTraverse (int [,] mat) { int m = mat . GetLength (0); int n = mat . GetLength (1); List < int > res = new List < int > (); // Initialize boundaries int top = 0 , bottom = m - 1 , left = 0 , right = n - 1 ; // Iterate until all elements are printed while (top <= bottom && left <= right) { // Print top row from left to right for (int i = left ; i <= right ; ++ i) { res . Add (mat [top , i]); } top ++ ; // Print right column from top to bottom for (int i = top ; i <= bottom ; ++ i) { res . Add (mat [i , right]); } right -- ; // Print bottom row from right to left (if exists) if (top <= bottom) { for (int i = right ; i >= left ; -- i) { res . Add (mat [bottom , i]); } bottom -- ; } // Print left column from bottom to top (if exists) if (left <= right) { for (int i = bottom ; i >= top ; -- i) { res . Add (mat [i , left]); } left ++ ; } } return res ; } static void Main (string [] args) { int [,] mat = {{ 1 , 2 , 3 , 4 }, { 5 , 6 , 7 , 8 }, { 9 , 10 , 11 , 12 }, { 13 , 14 , 15 , 16 }}; List < int > res = spirallyTraverse (mat); Console . WriteLine (string . Join (" " , res)); } } JavaScript function spirallyTraverse (mat) { const m = mat . length ; const n = mat [0]. length ; const res = []; // Initialize boundaries let top = 0 , bottom = m - 1 , left = 0 , right = n - 1 ; // Iterate until all elements are printed while (top <= bottom && left <= right) { // Print top row from left to right for (let i = left ; i <= right ; ++ i) { res . push (mat [top][i]); } top ++ ; // Print right column from top to bottom for (let i = top ; i <= bottom ; ++ i) { res . push (mat [i][right]); } right -- ; // Print bottom row from right to left (if exists) if (top <= bottom) { for (let i = right ; i >= left ; -- i) { res . push (mat [bottom][i]); } bottom -- ; } // Print left column from bottom to top (if exists) if (left <= right) { for (let i = bottom ; i >= top ; -- i) { res . push (mat [i][left]); } left ++ ; } } return res ; } // Driver Code const mat = [[1 , 2 , 3 , 4], [5 , 6 , 7 , 8], [9 , 10 , 11 , 12], [13 , 14 , 15 , 16]]; const res = spirallyTraverse (mat); console . log (res . join (" ")); Output 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 Comment Article Tags: Article Tags: Matrix DSA Arrays Basic Coding Problems Microsoft Amazon Morgan Stanley Oracle D-E-Shaw Snapdeal Paytm Accolite Zoho Nagarro MAQ Software BrowserStack MakeMyTrip nearbuy Amazon-Question Snapdeal-Question spiral pattern-printing Arrays + 19 More