Rat in a Maze - GeeksforGeeks

Source: https://www.geeksforgeeks.org/rat-in-a-maze/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Rat in a Maze Last Updated : 27 Sep, 2025 Given a square binary matrix mat[][] representing a maze. A rat starts at the top-left corner (0,0) and needs to reach the bottom-right corner (n-1, n-1). The rat can move in four directions: Up (U), Down (D), Left (L), Right (R). Find all possible paths from (0, 0) to (n-1, n-1). If multiple paths exist, return them in lexicographically sorted order otherwise If no path exists, return empty list. Note: A rat cannot visit the same cell more than once in a path. 1 represents an open cell (rat can visit), and 0 represents a blocked cell (rat cannot visit). Example: Input: mat[][] = [[1, 0, 0, 0], [1, 1, 0, 1], [1, 1, 0, 0], [0, 1, 1, 1]] Output: ["DDRDRR", "DRDDRR"] Explanation: The possible paths are: DDRDRR, DRDDRR [Approach] Using Recursion and Backtracking The idea is to explore all possible paths from the source to the destination in the maze. We recursively build each path, store it when the destination is reached, and backtrack to explore alternative routes. To avoid revisiting cells in the same path, we mark a cell as visited during exploration and unmark it during backtracking. C++ #include <iostream> #include <vector> #include <string> #include <algorithm> using namespace std ; // Directions: Down, Left, Right, Up string dir = "DLRU" ; int dr [4] = { 1 , 0 , 0 , -1 }; int dc [4] = { 0 , -1 , 1 , 0 }; // Check if a cell is valid (inside the maze and open) bool isValid (int r , int c , int n , vector < vector < int >>& maze) { return r >= 0 && c >= 0 && r < n && c < n && maze [r][c]; } // Function to find all valid paths void findPath (int r , int c , vector < vector < int >>& maze , string & path , vector < string >& res) { int n = maze . size (); // If destination is reached, store the path if (r == n - 1 && c == n - 1) { res . push_back (path); return ; } // Mark current cell as visited maze [r][c] = 0 ; for (int i = 0 ; i < 4 ; i ++) { int nr = r + dr [i], nc = c + dc [i]; if (isValid (nr , nc , n , maze)) { path . push_back (dir [i]); // Move to the next cell recursively findPath (nr , nc , maze , path , res); // Backtrack path . pop_back (); } } // Unmark current cell maze [r][c] = 1 ; } // Function to find all paths and return them vector < string > ratInMaze (vector < vector < int >>& maze) { vector < string > result ; int n = maze . size (); string path = "" ; if (maze [0][0] == 1 && maze [n - 1][n - 1] == 1) { // Start from (0,0) findPath (0 , 0 , maze , path , result); } // Sort results lexicographically sort (result . begin (), result . end ()); return result ; } int main () { vector < vector < int >> maze = { { 1 , 0 , 0 , 0 }, { 1 , 1 , 0 , 1 }, { 1 , 1 , 0 , 0 }, { 0 , 1 , 1 , 1 } }; vector < string > result = ratInMaze (maze); for (auto p : result) { cout << p << " " ; } return 0 ; } Java import java.util.ArrayList ; import java.util.Collections ; class GFG { // Directions: Down, Left, Right, Up static String dir = "DLRU" ; static int [] dr = { 1 , 0 , 0 , - 1 }; static int [] dc = { 0 , - 1 , 1 , 0 }; // Check if a cell is valid (inside the maze and open) static boolean isValid (int r , int c , int n , int [][] maze) { return r >= 0 && c >= 0 && r < n && c < n && maze [r][c] == 1 ; } // Function to find all valid paths static void findPath (int r , int c , int [][] maze , StringBuilder path , ArrayList < String > res) { int n = maze . length ; // If destination is reached, store the path if (r == n - 1 && c == n - 1) { res . add (path . toString ()); return ; } // Mark current cell as visited maze [r][c] = 0 ; for (int i = 0 ; i < 4 ; i ++) { int nr = r + dr [i] , nc = c + dc [i] ; if (isValid (nr , nc , n , maze)) { path . append (dir . charAt (i)); // Move to the next cell recursively findPath (nr , nc , maze , path , res); // Backtrack path . deleteCharAt (path . length () - 1); } } // Unmark current cell maze [r][c] = 1 ; } // Function to find all paths and return them static ArrayList < String > ratInMaze (int [][] maze) { ArrayList < String > result = new ArrayList <> (); int n = maze . length ; StringBuilder path = new StringBuilder (); if (maze [0][0] == 1 && maze [n - 1][n - 1] == 1) { // Start from (0,0) findPath (0 , 0 , maze , path , result); } // Sort results lexicographically Collections . sort (result); return result ; } public static void main (String [] args) { int [][] maze = { { 1 , 0 , 0 , 0 }, { 1 , 1 , 0 , 1 }, { 1 , 1 , 0 , 0 }, { 0 , 1 , 1 , 1 } }; ArrayList < String > result = ratInMaze (maze);

for (String p : result) { System . out . print (p + " "); } } } Python # Directions: Down, Left, Right, Up dir = "DLRU" dr = [1 , 0 , 0 , - 1] dc = [0 , - 1 , 1 , 0] # Check if a cell is valid (inside the maze and open) def isValid (r , c , n , maze): return r >= 0 and c >= 0 and r < n and c < n and maze [r][c] == 1 # Function to find all valid paths def findPath (r , c , maze , path , res): n = len (maze) # If destination is reached, store the path if r == n - 1 and c == n - 1 : res . append ("" . join (path)) return # Mark current cell as visited maze [r][c] = 0 for i in range (4): nr , nc = r + dr [i], c + dc [i] if isValid (nr , nc , n , maze): path . append (dir [i]) # Move to the next cell recursively findPath (nr , nc , maze , path , res) # Backtrack path . pop () # Unmark current cell maze [r][c] = 1 # Function to find all paths and return them def ratInMaze (maze): result = [] n = len (maze) path = [] if maze [0][0] == 1 and maze [n - 1][n - 1] == 1 : # Start from (0,0) findPath (0 , 0 , maze , path , result) # Sort results lexicographically result . sort () return result if __name__ == "__main__" : maze = [[1 , 0 , 0 , 0], [1 , 1 , 0 , 1], [1 , 1 , 0 , 0], [0 , 1 , 1 , 1]] result = ratInMaze (maze) for p in result : print (p , end = " ") C# using System ; using System.Collections.Generic ; class GFG { // Directions: Down, Left, Right, Up static string dir = "DLRU" ; static int [] dr = { 1 , 0 , 0 , - 1 }; static int [] dc = { 0 , - 1 , 1 , 0 }; // Check if a cell is valid (inside the maze and open) static bool isValid (int r , int c , int n , int [,] maze) { return r >= 0 && c >= 0 && r < n && c < n && maze [r , c] == 1 ; } // Function to find all valid paths static void findPath (int r , int c , int [,] maze , List < char > path , List < string > res) { int n = maze . GetLength (0); // If destination is reached, store the path if (r == n - 1 && c == n - 1) { res . Add (new string (path . ToArray ())); return ; } // Mark current cell as visited maze [r , c] = 0 ; for (int i = 0 ; i < 4 ; i ++) { int nr = r + dr [i], nc = c + dc [i]; if (isValid (nr , nc , n , maze)) { path . Add (dir [i]); // Move to the next cell recursively findPath (nr , nc , maze , path , res); // Backtrack path . RemoveAt (path . Count - 1); } } // Unmark current cell maze [r , c] = 1 ; } // Function to find all paths and return them static List < string > ratInMaze (int [,] maze) { List < string > result = new List < string > (); int n = maze . GetLength (0); List < char > path = new List < char > (); if (maze [0 , 0] == 1 && maze [n - 1 , n - 1] == 1) { // Start from (0,0) findPath (0 , 0 , maze , path , result); } // Sort results lexicographically result . Sort (); return result ; } public static void Main (string [] args) { int [,] maze = { { 1 , 0 , 0 , 0 }, { 1 , 1 , 0 , 1 }, { 1 , 1 , 0 , 0 }, { 0 , 1 , 1 , 1 } }; List < string > result = ratInMaze (maze); foreach (string p in result) { Console . Write (p + " "); } } } JavaScript // Directions: Down, Left, Right, Up const dir = "DLRU" ; const dr = [1 , 0 , 0 , - 1]; const dc = [0 , - 1 , 1 , 0]; // Check if a cell is valid (inside the maze and open) function isValid (r , c , n , maze) { return r >= 0 && c >= 0 && r < n && c < n && maze [r][c] === 1 ; } // Function to find all valid paths function findPath (r , c , maze , path , res) { const n = maze . length ; // If destination is reached, store the path if (r === n - 1 && c === n - 1) { res . push (path); return ; } // Mark current cell as visited maze [r][c] = 0 ; for (let i = 0 ; i < 4 ; i ++) { const nr = r + dr [i], nc = c + dc [i]; if (isValid (nr , nc , n , maze)) { path += dir [i]; // Move to the next cell recursively findPath (nr , nc , maze , path , res); // Backtrack path = path . slice (0 , - 1); } } // Unmark current cell maze [r][c] = 1 ; } // Function to find all paths and return them function ratInMaze (maze) { let result = []; const n = maze . length ; let path = "" ; if (maze [0][0] === 1 && maze [n - 1][n - 1] === 1) { // Start from (0,0) findPath (0 , 0 , maze , path , result); } // Sort results lexicographically result . sort (); return result ; } // Driver Code const maze = [[1 , 0 , 0 , 0], [1 , 1 , 0 , 1], [1 , 1 , 0 , 0], [0 , 1 , 1 , 1]]; const result = ratInMaze (maze); console . log (result . join (" ")); Output DDRDRR DRDDRR Time Complexity: O(4 n*n), because on every cell we have to try 4 different directions. Auxiliary Space: O(n 2), Maximum Depth of the recursion tree. Comment Article Tags: Article Tags: Backtracking Matrix DSA Amazon Flipkart Drishti-Soft Yatra.com Expedia Paytm MakeMyTrip Visa Grofers Numerify Zycus + 10 More