**A Project Report**

**On**

**THE SUDOKU SOLVER**

***Submitted in partial fulfillment of the***

***requirement for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**DEGREE**

**Session 2023-24**

**In**

**Name of discipline**

**NISHANT BHAIRAV (21SCSE1010476)**

**Under the guidance of**

**Dr. Vimal Kumar**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GALGOTIAS UNIVERSITY, GREATER NOIDA**

**INDIA**

**Jan, 2024**

**INTRODUCTION**

The Java code presented implements a maze pathfinding algorithm using recursive backtracking. This algorithm explores various paths within a maze, avoiding invalid routes and backtracking when necessary. Users are prompted to input the maze dimensions, allowing for customization and observation of the algorithm's behavior. The code aims to find and print all possible paths from the top-left to the bottom-right corner of the maze. Through recursive exploration, the algorithm provides a comprehensive understanding of backtracking strategies in solving maze-related problems, albeit with potential limitations in scalability for larger maze sizes.

**Problem Statement**

The task is to devise a Java program that employs recursive backtracking to find and display all feasible paths within a maze. The maze is represented as a grid, and the algorithm must navigate from the top-left to the bottom-right corner, exploring four possible directions at each step: right, down, left, and up. The code should efficiently handle user-defined maze dimensions and print both the count of paths and the individual paths themselves. The challenge lies in implementing a backtracking strategy to explore paths while avoiding invalid cells, ultimately providing a versatile solution for various maze configurations.

**Code Structure:**

The code is organized into a class named mazepathBacktrack\_no\_03, containing a main method and the findPossibleMazePath method responsible for the core logic. The code accepts user input for the number of rows and columns in the maze through the console. The isValid boolean array is used to keep track of visited cells.

**Algorithm Overview:**

The algorithm explores four possible directions at each step: right, down, left, and up. It uses a recursive approach to explore each direction and backtracks if the path leads to an invalid cell or reaches the destination. The base case is when the current cell is the destination, at which point the current path is printed, and the count of paths is incremented.

**Time Complexity Analysis:**

The time complexity of the code is determined by the number of recursive calls made during the exploration of paths. In the worst case, the algorithm explores all possible paths in the maze. Each cell has four possible directions, and for each direction, a recursive call is made. Therefore, the time complexity is O(4^(n\*m)), where "n" is the number of rows and "m" is the number of columns in the maze.

It's important to note that backtracking is employed in the algorithm, which prunes paths that lead to invalid or already visited cells. This, in practice, can significantly reduce the effective number of recursive calls, leading to a runtime that is often much better than the worst-case analysis suggests.

**User Interaction:**

The code prompts the user to input the number of rows and columns for the maze through the console. This allows users to experiment with different maze sizes and observe the algorithm's behavior. The user is then presented with the count of all possible paths and the individual paths themselves.

**Scalability and Limitations:**

The algorithm is suitable for small to moderately sized mazes. However, due to its exponential time complexity, it may become impractical for larger mazes as the number of possible paths grows exponentially with the maze size.

**Conclusion:**

In conclusion, the provided code offers an implementation of maze pathfinding using a backtracking algorithm. It efficiently explores different paths in the maze and provides valuable insights into the nature of maze-solving algorithms. The recursive backtracking approach, while conceptually simple, may face limitations in scalability for larger maze sizes. Users can experiment with different maze dimensions to observe the algorithm's behavior and gain a deeper understanding of backtracking strategies in pathfinding algorithms.

**Code :**

import *java.util.\**;

*public* *class* mazepathBacktrack\_no\_03 {

*private* *static* int count = 0;

*public* *static* void findPossibleMazePath(int sr, int sc, int er, int ec, String s, boolean [][]isValid){

        if(sr< 0 || sc < 0) return;

        if(sr > er || sc > ec) return;

        if(isValid[sr][sc] == true) return;

        if(sr == er && sc == ec) {

            System.out.println(s);

            count++;

            return;

        }

        isValid[sr][sc] = true;

*// right*

        findPossibleMazePath(sr, sc+1, er, ec,  s + "R", isValid);

*// down*

        findPossibleMazePath(sr+1, sc, er, ec,  s+"D", isValid);

*//left*

        findPossibleMazePath(sr, sc-1, er, ec,  s+"L", isValid);

*//up*

        findPossibleMazePath(sr-1, sc, er, ec,  s + "U", isValid);

*// Backtracking*

        isValid[sr][sc] = false;

    }

*public* *static* void main(String[] args) {

*// int row = 3;*

*// int column = 3;*

*// boolean [][] isValid = new boolean[row][column];*

        Scanner sc = new Scanner(System.in);

        int row;

        int column;

        System.out.println("Enter the value of the row");

        row = sc.nextInt();

        System.out.println("Enter the value of column");

        column = sc.nextInt();

        boolean [][] isValid = new boolean[row][column];

        findPossibleMazePath(0, 0, row-1, column-1, "", isValid);

        System.out.println("Count of the all path is " + count);

    }

}

**References**

1) Wikipedia [cited 2013 February 21], Web site: <http://en.wikipedia.org/wiki/Sudoku>

2) Home Of Logic Puzzles [cited 2013 February 22], Web Page: <http://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/classic>

3) J.F. Crook, A pencil and paper algorithm for solving Sudoku Puzzles, [Cited 2013 February 24], Winthrop University, Webpage: <http://www.ams.org/notices/200904/tx090400460p.pdf>

4) A.S. Showdhury, S. Skher Solving Sudoku with Boolean Algebra [Cited 2013 February 24], International Journal of Computer Applications, Peer-reviewed Research, Webpage: <http://research.ijcaonline.org/volume52/number21/pxc3879024.pdf>

5) N. Pillay, Finding Solutions to Sudoku Puzzles Using Human Intuitive Heuristics, South African Research Articles, Webpage: <http://sacj.cs.uct.ac.za/index.php/sacj/article/viewArticle/111>

6) Ch. Xu, W. Xu, The model and Algorithm to Estimate the Difficulty Levels of Sudoku Puzzles, Journal of Mathematics Research, 2009 Webpage: <http://journal.ccsenet.org/index.php/jmr/article/viewFile/3732/3336>

7) T. Davis, The Mathematics of Sudoku (http://www.geometer.org/index.html), Research Article, Webpage: http://share.dschola.it/castigliano/ips/Documentazione%20Progetto/Materiale%20Didattico/M atematica/1E/sudoku.pdf