Code: Rat In A Maze

Send Feedback

You are given a N*N maze with a rat placed at maze[0][0]. Find whether any path exist that rat can follow to reach its destination i.e. maze[N-1][N-1]. Rat can move in any direction (left, right, up and down).

Value of every cell in the maze can either be 0 or 1. Cells with value 0 are blocked means rat can-not enter into those cells and those with value 1 are open.

Input Format

```
Line 1: Integer N
Next N Lines: Each line will contain ith row elements (separated by space)
```

Output Format :

```
The output line contains true if any path exists for the rat to reach its destination otherwise print false.
```

Sample Input 1 :

```
3
1 0 1
1 0 1
1 1 1
```

Sample Output 1 :

```
true
```

Sample Input 2 :

```
3
1 0 1
1 0 1
0 1 1
```

Sample Output 2 :

```
 false
```

```
public class Solution {



	public static boolean ratInAMaze(int maze[][]){



		/*Your class should be named Solution.

		*Don't write main().

		*Don't take input, it is passed as function argument.

		*Don't print output.

		*Taking input and printing output is handled automatically.

		*/

    int path[][] = new int[maze.length][maze.length];

		return solveMaze(maze,0,0,path);

	}
```

```
public static boolean solveMaze(int[][] maze, int i, int j, int[][] path)

    {

            //Check if i,j are valid pair of indices => i,j>=0

            int n=maze.length;

            if (i<0 || j<0 || i>=n || j>=n)

                    return false;


            //If cell is already part of the path

            if (path[i][j]==1)

                    return false;


            //If cell is blocked in maze (cell value=0)

            if (maze[i][j]==0)

                    return false;


            //If all previous conditions fail, then the cell is a possible path

            //Include the cell in current path

            path[i][j]=1;


            //If we have reached ending point

            if (i==n-1 && j==n-1)

                    return true;


            //Now, explore in all directions

            // Direction 1 - move towards cell above (top direction)

            if (solveMaze(maze,i-1,j,path))

                    return true;
            //Direction 2 - move towards cell to the right (right direction)

            if (solveMaze(maze,i,j+1,path))

                return true;



            //Direction 3 - move towards cell below (bottom direction)

            if (solveMaze(maze,i+1,j,path))

                return true;



            //Direction 3 - move towards cell to the left (left direction)

            if (solveMaze(maze,i,j-1,path))

                return true;



            //If none of the conditions are satisfied, then the path is not working out

            return false;

        }

}
```

Code: N Queens

Send Feedback

You are given N, and for a given N x N chessboard, find a way to place N queens such that no queen can attack any other queen on the chess board. A queen can be killed when it lies in the same row, or same column, or the same diagonal of any of the other queens. You have to print all such configurations.

Input Format :

```
Line 1 : Integer N
```

Output Format :

```
One Line for every board configuration.
Every line will have N*N board elements printed row wise and are separated by space
```

Note : Don't print anything if there isn't any valid configuration.

Constraints :

1<=N<=10

Sample Input 1:

```
4
```

Sample Output 1 :

```
0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0
0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0
```

```
public class Solution {
```

```
public static void placeNQueens(int n){


            /* Your class should be named Solution.
             * Don't write main() function.
             * Don't read input, it is passed as function argument.
             * Print output as specified in the question
             */


   int[][] board = new int[n][n];
   solveNQueens(board, 0,n);


            }


   static void solveNQueens(int board[][], int row, int N)
            {
   /* base case: If all queens are placed
   then return true */
   if (row == N)
   {
      printSolution(board,N);
      return;
   }


   /* Consider this column and try placing
   this queen in all rows one by one */
   for (int i = 0; i < N; i++)
```

```
{
   /* Check if queen can be placed on
   board[row][i] */
   if ( isSafe(board, row, i, N) )
   {
      /* Place this queen in board[row][i] */
      board[row][i] = 1;

      // Make result true if any placement
      // is possible
      solveNQueens(board, row + 1, N);

      /* If placing queen in board[row][i]
      doesn't lead to a solution, then backtrack and
      remove queen from board[row][i] */
      board[row][i] = 0;
   }
}

        }

static boolean isSafe(int board[][], int row, int col, int N)
        {
int i, j;

//Check if all values in the given column and rows from 0 to row-1 are 0
for (i=0;i<row;i++)
{
        if (board[i][col]==1)
            return false;
    }

            // Check upper diagonal on left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j] == 1)
            return false;

            //Check upper right diagonal
    for (i=row,j=col;i>=0 && j<N;i--,j++)
        if (board[i][j] == 1)
            return false;

    return true;
            }

    static void printSolution(int board[][], int N)
            {
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            System.out.print(board[i][j]+" ");
    }
    System.out.println();
    }

}
```

Code: Rat In a Maze All Paths

Send Feedback

You are given a N*N maze with a rat placed at maze[0][0]. Find and print all paths that rat can follow to reach its destination i.e. maze[N-1][N-1]. Rat can move in any direction (left, right, up and down). Value of every cell in the maze can either be 0 or 1. Cells with value 0 are blocked means rat can-not enter into those cells and those with value 1 are open.

Input Format

The first line of input contains an integer 'N' representing
the dimension of the maze.
The next N lines of input contain N space-separated
integers representing the type of the cell.

Output Format :

For each test case, print the path from start position to destination position and only cells that are part of the solution path should be 1, rest all cells should be 0.

Output for every test case will be printed in a separate line.

Constraints:

0 < N < 11 0 <= Maze[i][j] <=1

Time Limit: 1sec

Sample Input 1 :

```
3
1 0 1
1 0 1
1 1 1
```

Sample Output 1 :

```
1 0 0 1 0 0 1 1 1
```

Sample Output 1 Explanation :

Only 1 path is possible

```
1 0 0
1 0 0
1 1 1
```

Which is printed from left to right and then top to bottom in one line.

Sample Input 2 :

```
3
1 0 1
1 1 1
1 1 1
```

Sample Output 2 :

```
1 0 0 1 1 1 1 1 1
1 0 0 1 0 0 1 1 1 s
1 0 0 1 1 0 0 1 1
1 0 0 1 1 1 0 0 1
```

Sample Output 2 Explanation :

4 paths are possible which are printed in the required format.

//Java code

```
public class Solution {
    static void printAllPaths(int [][] paths,int n){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                System.out.print(paths[i][j]+" ");
            }
        }
    }
static void solveMaze(int maze[][],int paths[][], int x,int y,int n){
    if(x==n-1 && y==n-1){
        paths[x][y]=1;
        printAllPaths(paths,n);
        System.out.println();
        return;
    }
    if(x>n-1 || x<0 || y>n-1 || y<0){
        return;
    }
    if(x>n-1 || x<0 || y>n-1 || y<0 || maze[x][y]==0 || paths[x][y]==1){
        return;
    }
    paths[x][y]=1;
    solveMaze(maze,paths,x-1,y,n);
    solveMaze(maze,paths,x+1,y,n);
    solveMaze(maze,paths,x,y-1,n);
    solveMaze(maze,paths,x,y+1,n);
    paths[x][y]=0;
}

            static void ratInAMaze(int maze[][], int n) {
                        /*
                                    * Your class should be named Solution.
                                    * Write your code here
                         */
        int [][] paths = new int[20][20];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                paths[i][j]=0;
            }
        }
        solveMaze(maze,paths,0 , 0,n);

            }

}
```

//C++ code

```
void ratMaze(int maze[][20], int n, int row, int col){

   if(row == n-1 && col == n-1){

      for(int i = 0; i< n; i++){
          for(int j = 0; j<n; j++){
              cout<<board[i][j]<<" ";
          }
      }
      cout<<endl;
      return;
   }
   if(row < n-1 && col < n){
      if(board[row + 1][col] == 0 && maze[row+1][col] == 1 ){
          board[row+1][col] = 1;
          ratMaze(maze, n, row+1, col);
          board[row+1][col] = 0;
      }
   }
   if(row < n && col < n+1){
      if(board[row][col+1] == 0 && maze[row][col+1] == 1 ){
          board[row][col+1] = 1;
          ratMaze(maze, n, row, col+1);
          board[row][col+1] = 0;
      }
   }
   if(row >= 0 && col >= 1 && row<n && col<n){
      if(board[row][col-1] == 0 && maze[row][col-1] == 1 ){
          board[row][col-1] = 1;
          ratMaze(maze, n, row, col-1);
          board[row][col-1] = 0;
      }
   }
```

```
    if(row >=1 && col >=0 && row<n && col<n){

        if(board[row - 1][col] == 0 && maze[row-1][col] == 1 ){

            board[row-1][col] = 1;

            ratMaze(maze, n, row-1, col);

            board[row-1][col] = 0;

        }

    }

}


void ratInAMaze(int maze[][20], int n){


  /* Don't write main().

   *  Don't read input, it is passed as function argument.

   *  Print output as specified in the question

  */

 memset(board,0,15*15*sizeof(int));

  board[0][0] = 1;

  ratMaze(maze,n,0,0);

  return;


}
```