

BACKTRACKING SOLUTIONS

Solution 1:

Algorithm -

- 1. Create a solution matrix, initially filled with 0's.
- 2. Create a recursive function, which takes the initial matrix, output matrix and position of rat (i, j).
- 3. if the position is out of the matrix or the position is not valid then return.
- 4. Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.
- 5. Recursively call for position (i+1, j) and (i, j+1).
- 6. Unmark position (i, j), i.e output[i][j] = 0.

```
&& y \ge 0 && y < maze.length && maze[x][y] == 1);
public static boolean solveMaze(int maze[][]) {
    int sol[][] = new int[N][N];
```



```
if (x == maze.length - 1 && y == maze.length - 1 && maze[x][y] == 1) {
    sol[x][y] = 1;
    if (sol[x][y] == 1)
    if (solveMazeUtil(maze, x, y + 1, sol))
    sol[x][y] = 0;
int maze[][] = { { 1, 0, 0, 0 },
solveMaze(maze);
```



Solution 2:

```
ublic class Solution {
  final static char[][] L = {{},{},{'a','b','c'},{'d','e','f'},{'g','h','i'},
 public static void letterCombinations(String D) {
     bfs(0, len, new StringBuilder(), D);
      if (pos == len) {
         System.out.println(sb.toString());
          char[] letters = L[Character.getNumericValue(D.charAt(pos))];
              bfs(pos+1, len, new StringBuilder(sb).append(letters[i]), D);
```

Solution 3:

```
public class Solution {
   static int N = 8;
   public static boolean isSafe(int x, int y, int sol[][]){
```



```
return (x >= 0 && x < N && y >= 0 && y < N
            && sol[x][y] == -1);
            System.out.print(sol[x][y] + " ");
public static boolean solveKT() {
    sol[0][0] = 0;
        printSolution(sol);
public static boolean solveKTUtil(int x, int y, int movei, int sol[][],
```



COLLEGE