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.

```java
public class Solution {

  public static void printSolution(int sol[][]) {

    for (int i = 0; i<sol.length; i++) {

      for (int j = 0; j<sol.length; j++) {

        System.out.print(" " + sol[i][j] + " ");

      }

      System.out.println();

    }

  }


  public static boolean isSafe(int maze[][], int x, int y) {

    // if (x, y outside maze) return false

    return (x >= 0 && x < maze.length

      && y >= 0 && y < maze.length && maze[x][y] == 1);

  }


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

    int N = maze.length;

    int sol[][] = new int[N][N];

    if (solveMazeUtil(maze, 0, 0, sol) == false) {

      System.out.print("Solution doesn't exist");

      return false;

    }

    printSolution(sol);

    return true;
```

```java
    }

    public static boolean solveMazeUtil(int maze[][], int x, int y, int sol[][]) {
        if (x == maze.length - 1 && y == maze.length - 1 && maze[x][y] == 1) {
            sol[x][y] = 1;
            return true;
        }


        // Check if maze[x][y] is valid
        if (isSafe(maze, x, y) == true) {
            if (sol[x][y] == 1)
                return false;
            sol[x][y] = 1;
            if (solveMazeUtil(maze, x + 1, y, sol))
                return true;
            if (solveMazeUtil(maze, x, y + 1, sol))
                return true;
            sol[x][y] = 0;
            return false;
        }


        return false;
    }

    public static void main(String args[]){
        int maze[][] = { { 1, 0, 0, 0 },
                         { 1, 1, 0, 1 },
                         { 0, 1, 0, 0 },
                         { 1, 1, 1, 1 } };


        solveMaze(maze);
    }
}
```

Solution 2:

```java
public class Solution {
    final static char[][] L = {{},{},{'a','b','c'},{'d','e','f'},{'g','h','i'},
                    {'j','k','l'},{'m','n','o'},{'p','q','r','s'},
                    {'t','u','v'},{'w','x','y','z'}};

    public static void letterCombinations(String D) {
        int len = D.length();
        if (len == 0) {
            System.out.println("");
            return;
        }
        bfs(0, len, new StringBuilder(), D);
    }

    public static void bfs(int pos, int len, StringBuilder sb, String D) {
        if (pos == len){
            System.out.println(sb.toString());
        }
        else {
            char[] letters = L[Character.getNumericValue(D.charAt(pos))];
            for (int i = 0; i < letters.length; i++)
                bfs(pos+1, len, new StringBuilder(sb).append(letters[i]), D);
        }
    }
    public static void main(String args[]){
        letterCombinations("2");
    }
}
```

Solution 3 :

```java
public class Solution {
    static int N = 8;

    public static boolean isSafe(int x, int y, int sol[][]){
```

```java
        return (x >= 0 && x < N && y >= 0 && y < N
                && sol[x][y] == -1);
    }


    public static void printSolution(int sol[][]) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++)
                System.out.print(sol[x][y] + " ");
            System.out.println();

        }

    }


    public static boolean solveKT() {
        int sol[][] = new int[8][8];
        for (int x = 0; x < N; x++)
            for (int y = 0; y < N; y++)
                sol[x][y] = -1;

        int xMove[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
        int yMove[] = { 1, 2, 2, 1, -1, -2, -2, -1 };

        //As the Knight starts from cell(0,0)
        sol[0][0] = 0;

        if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {
            System.out.println("Solution does not exist");
            return false;

        }
        else
            printSolution(sol);

        return true;

    }


    public static boolean solveKTUtil(int x, int y, int movei, int sol[][],
                                int xMove[],int yMove[]) {
        int k, next_x, next_y;
        if (movei == N * N)
            return true;
```

```java
        for (k = 0; k < 8; k++) {

            next_x = x + xMove[k];

            next_y = y + yMove[k];

            if (isSafe(next_x, next_y, sol)) {

                sol[next_x][next_y] = movei;

                if (solveKTUtil(next_x, next_y, movei + 1,
                                sol, xMove, yMove))

                    return true;

                else

                    sol[next_x][next_y]

                        = -1; // backtracking

            }

        }

        return false;

    }


    public static void main(String args[]){

        solveKT();

    }

}
```