You're working with backtracking, a fundamental technique in recursive problem-solving. Let's build a strong theoretical foundation and then analyze your code thoroughly.

# 🧠 Backtracking – Theoretical Foundation

## 📌 Definition:

Backtracking is a general algorithmic technique that considers searching every possible combination in a solution space and prunes paths that do not lead to a solution.

It is used when:

- You want to explore all possible solutions (like paths in a maze, permutations, combinations, etc.)
- You can discard a solution halfway if it doesn't meet the criteria (pruning).
- Solutions are built incrementally.

## ⚙️ Key Steps in Backtracking:

1. Make a choice (move to a cell or add a character).
2. Explore (recursively call the function).
3. Unmake the choice (undo the move — i.e., backtrack).

This is often known as "Try – Explore – Undo".

## 📚 Applications of Backtracking:

- Maze solving
- Sudoku solver
- N-Queens problem
- Permutations & combinations
- Knight's tour
- Word search

## 🔍 Code Analysis & Explanation

## 🔁 mazeGrid(String p, boolean[][] maze, int r, int c)

## 📜 Function Purpose:

To print all valid paths from the top-left corner (0,0) to the bottom-right corner (n-1,m-1) avoiding obstacles and using backtracking.

## 📌 Code with Comments:

```
static void mazeGrid(String p , boolean [][] maze, int r,int c)

{

   // Base case: if we reach bottom-right, print the path
```

```
    if(r == maze.length - 1 && c == maze[0].length - 1) {
        System.out.println(p);
        return;
    }

    // Invalid cell or already visited
    if(!maze[r][c]) {
        return;
    }

    // Mark current cell as visited
    maze[r][c] = false;

    // Move in 4 directions
    if(c < maze[0].length - 1)   {   mazeGrid(p + 'R', maze, r, c + 1); } // Right
    if(r < maze.length - 1)   {    mazeGrid(p + 'D', maze, r + 1, c);     }// Down
    if(r > 0)              {mazeGrid(p + 'U', maze, r - 1, c);      }    // Up
    if(c > 0)              {         mazeGrid(p + 'L', maze, r, c - 1);        } // Left

    // Backtrack: unmark the cell
    maze[r][c] = true;
}
```

## 📊 Time & Space Complexity (for mazeGrid)

- Time Complexity: O(4^(N*M)) in the worst case (if all paths are open).
- Space Complexity: O(N*M) due to the recursion stack in the worst case (deepest path).

## 📍 mazeGridPrint(...) – With Path Printing

## 📜 Function Purpose:

**Prints each path from start to end and also prints the step count matrix showing which step visited which cell.**

## 🧠 Backtracking with Additional Info:

```
static void mazeGridPrint(String p ,boolean [][] maze, int r, int c , int [][] path,int count)

{

  // Base case: reached destination

  if(r == maze.length-1 && c == maze[0].length-1)

  {

    path[r][c] = count;



    // Print path matrix

    for(int[] list : path) {

      System.out.println(Arrays.toString(list));

    }

    System.out.println(p); // Print path string

    System.out.println();

    return;

  }



  // Invalid move or blocked cell

  if(!maze[r][c]) {

    return;

  }



  // Mark the step count and cell as visited

  path[r][c] = count;

  maze[r][c] = false;
```

```
    // Recursive calls in all 4 directions
    if(c < maze[0].length - 1) mazeGridPrint(p + 'R', maze, r, c + 1, path, count + 1);
    if(r < maze.length - 1)    mazeGridPrint(p + 'D', maze, r + 1, c, path, count + 1);
    if(r > 0)                  mazeGridPrint(p + 'U', maze, r - 1, c, path, count + 1);
    if(c > 0)                  mazeGridPrint(p + 'L', maze, r, c - 1, path, count + 1);

    // Backtrack
    maze[r][c] = true;
    path[r][c] = 0;
}
```

## 📊 Time & Space Complexity (for mazeGridPrint)

- Time Complexity: O(4^(N*M))
- Space Complexity:
  - O(N*M) for recursion.
  - O(N*M) for the path matrix (extra space).

## 🧩 Key Concepts in This Program

| Concept | Explanation |
|---|---|
| Visited Marking | maze[r][c] = false; marks current cell as visited. |
| Backtracking Step | maze[r][c] = true; reverses the visited mark on returning. |
| Path Recording | path[r][c] = count; logs the step number during traversal. |
| 4-Directional Search | Explores all possible directions from the current cell. |
| Avoiding Loops | Visited marking prevents revisiting and infinite loops. |

## ⬅ END Summary

| Function | Role | Backtracking Applied |
|---|---|---|
| mazeGrid() | Prints all valid paths | Yes |
| mazeGridPrint() | Prints paths with step tracing | Yes |

**Backtracking here ensures that each path is explored independently, and changes to the maze or path state do not affect other recursive branches.**

**Would you like me to help you convert this logic into an iterative version or optimize it for a larger grid (e.g., with memoization)?**