CHAPTER 5 Recursion Part 4

Backtracking, Maze

Backtracking

Section 5.6

Backtracking

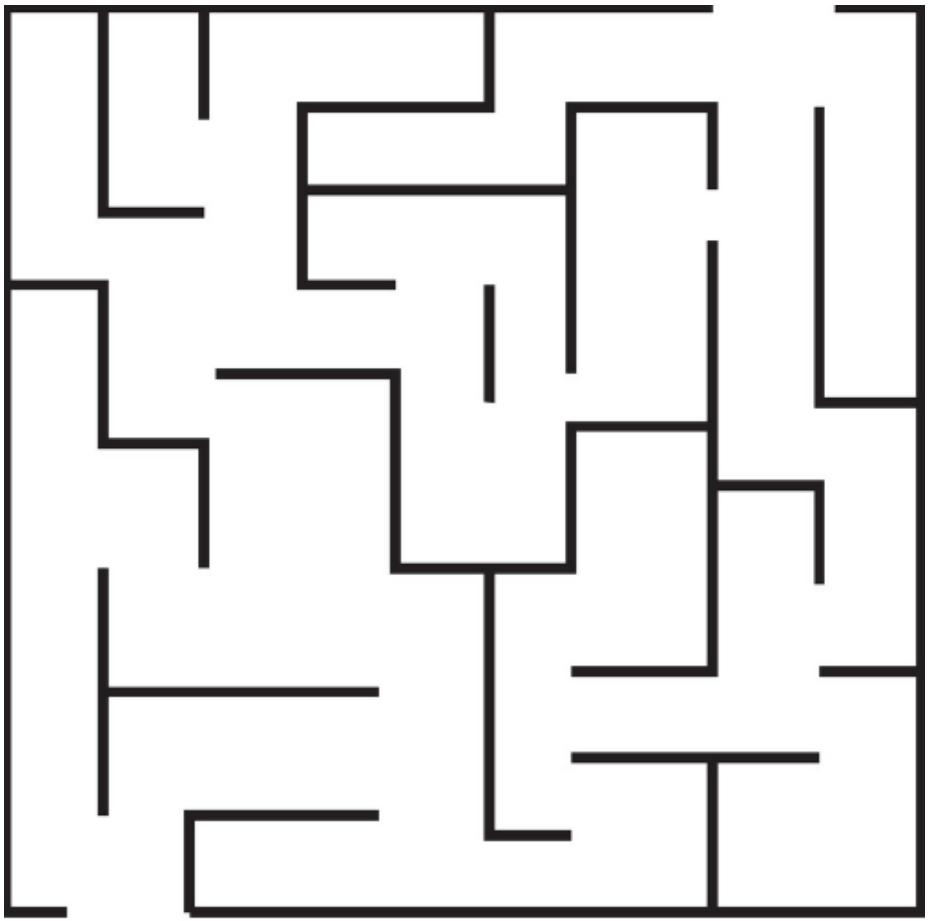
- Backtracking is an approach to implementing a systematic trial and error search for a solution
- An example is finding a path through a maze
- If you are attempting to walk through a maze, you will probably walk down a path as far as you can go
  - Eventually, you will reach your destination or you won't be able to go any farther
  - If you can't go any farther, you will need to consider alternative paths
- Backtracking is a systematic, nonrepetitive approach to trying alternative paths and eliminating them if they don't work

Backtracking (cont.)

- If you never try the same path more than once, you will eventually find a solution path if one exists
- Problems that are solved by backtracking can be described as a set of choices made by some method
- Recursion allows you to implement backtracking in a relatively straightforward manner
  - Each activation frame (i.e. execution context) is used to remember the choice that was made at that particular decision point
- A program that plays chess may involve some kind of backtracking algorithm

Finding a Path through a Maze

- Problem
  - Use backtracking to find and display the path through a maze
  - From each point in a maze you can move to the next cell in a horizontal or vertical direction if the cell is not blocked



Finding a Path through a Maze (cont.)

- Analysis
  - **The maze** will consist of a **grid of colored cells**
  - The **starting point** is at the **top left corner (0,0)**
  - The **exit point** is at the **bottom right corner (`getNCols() - 1, getNRow -1`)**
  - All cells possibly on the path will be **`BACKGROUND`** color initially
  - All cells that represent **barriers** will be **`ABNORMAL`** color
  - Cells that **are visited but found dead end** will be **`TEMPORARY`** color
  - If we find a path, **all cells on the path** will be set to **`PATH` color**

Recursive Algorithm for Finding Maze Path

Recursive Algorithm for `findMazePath(x, y)`

```
if the current cell is outside the maze
      return false (out of bounds)
else if  the current cell is part of the barrier or has been visited already
      return false (barrier or dead end or (discovered & not dead end))
else if  the current cell is the maze exit
      recolor it to the path color and return true (you have successfully
      completed the maze)
else //  Try to find a path from the current cell to the exit:
      mark the current cell as candidate on the path by recoloring it to the path color
      for each neighbor of the current cell
            if a path exists from the neighbor to the maze exit
                  return true
      // No neighbor of the current cell is on the path
      recolor current cell to the temporary color (visited & dead end) and return false
```

Implementation

- Listing 5.4 (`Maze.java`, pages 249-250)

NOTE:

The details inside these classes are not required.

- MazeDemo.java
- TwoDimGrid.java

Testing

- Test for a variety of test cases:
  - Mazes that can be solved
  - Mazes that can't be solved (no path exist)
  - A maze with no barrier cells
  - A maze with a single barrier cell at the exit point