### **Isabella Dougherty**

Project 6

CS4412

Prepared for Advanced Algorithm Graders and Dr. David Beard

### **Big O Analysis of solveMaze() (Depth-First Search Algorithm)**

**Algorithm Overview:**

* The solveMaze() method employs a **Depth-First Search (DFS)** strategy to explore the maze from the start cell (0,0) to the end cell (rows-1, cols-1).
* It recursively explores as far as possible along each branch before backtracking.

**Big O Analysis:**

* **Time Complexity:**
  + **Worst-Case Scenario:** In the worst case, DFS explores every cell in the maze exactly once.
  + **Number of Cells:** The maze has R rows and C columns, totaling N = R × C cells.
  + **Operations per Cell:** For each cell, the algorithm checks up to four neighboring cells (top, right, bottom, left).
  + **Overall Time Complexity:**
    - O(N)=O(R×C)O(N) = O(R \times C)O(N)=O(R×C)
* **Space Complexity:**
  + **Recursive Call Stack:** In the worst case, the recursion depth can be O(N) if the maze is a long, winding path without branches.
  + **Visited Array:** Uses a bool[,] visited array of size N to keep track of visited cells.
  + **Overall Space Complexity:**
    - O(N)=O(R×C)O(N) = O(R \times C)O(N)=O(R×C)

**Justification:**

* **Time Complexity Justification:**
  + Each cell is visited once, and for each visit, a constant amount of work is done (checking up to four neighbors).
  + Therefore, the time scales linearly with the number of cells.
* **Space Complexity Justification:**
  + The visited array requires space proportional to the number of cells.
  + The recursion stack can, in the worst case, grow linearly with the number of cells if the path is linear without branches.

### **Big O Analysis of findPath() (Breadth-First Search Algorithm)**

**Algorithm Overview:**

* The findPath(Point start, Point end) method utilizes a **Breadth-First Search (BFS)** approach to find the shortest path between any two arbitrary cells in the maze.
* BFS explores all neighboring nodes at the present depth before moving on to nodes at the next depth level.

**Big O Analysis:**

* **Time Complexity:**
  + **Worst-Case Scenario:** BFS examines every cell in the maze until the end cell is found.
  + **Number of Cells:** The maze has R rows and C columns, totaling N = R × C cells.
  + **Operations per Cell:** For each cell, BFS checks up to four neighboring cells.
  + **Overall Time Complexity:**
    - O(N)=O(R×C)O(N) = O(R \times C)O(N)=O(R×C)
* **Space Complexity:**
  + **Queue Storage:** In the worst case, the queue can hold up to O(N) paths, especially in dense mazes.
  + **Visited Set:** Uses a HashSet<Point> to keep track of visited cells, requiring O(N) space.
  + **Overall Space Complexity:**
    - O(N)=O(R×C)O(N) = O(R \times C)O(N)=O(R×C)

**Justification:**

* **Time Complexity Justification:**
  + BFS systematically explores all reachable cells from the start cell until the end cell is found.
  + Each cell is enqueued and dequeued exactly once, leading to linear time complexity relative to the number of cells.
* **Space Complexity Justification:**
  + The queue may store multiple paths simultaneously, but each cell is part of at most one path in the queue.
  + The visited set ensures that each cell is processed only once, requiring space proportional to the number of cells.

### **Formal Proof that the Algorithm Will Eventually Generate a Path from the Start Cell to the End Cell**

**Statement:** The maze generation algorithm implemented ensures that there exists at least one valid path from the start cell (0,0) to the end cell (R-1, C-1).

**Proof:**

**1. Maze Generation Process Overview:**

* **BuildMaze:** Initializes the maze grid with all walls intact.
* **BreakMaze:** Utilizes a randomized algorithm to carve out paths by removing walls between adjacent cells.
* The algorithm ensures that the maze is a **perfect maze**, meaning:
  + There is exactly one unique path between any two cells (i.e., the maze is a tree structure without cycles).
  + All cells are reachable from any other cell.

**2. Properties of the Generated Maze:**

* **Connectivity:**
  + By definition, a perfect maze forms a spanning tree over the grid graph representing the maze.
  + A spanning tree connects all vertices (cells) without forming any cycles.
* **Uniqueness of Paths:**
  + Since the maze forms a tree, there is exactly one unique path between any two cells.

**3. Ensuring a Path from Start to End:**

* **Existence of Path:**
  + In a spanning tree, all cells are interconnected.
  + Therefore, there must exist a unique path from the start cell (0,0) to the end cell (R-1, C-1).
* **Construction of the Path:**
  + The solveMaze() method (DFS) and findPath() method (BFS) are designed to traverse the maze based on the connectivity defined by the removed walls.
  + Given that the maze is fully connected, these algorithms are guaranteed to find the path from start to end.

**4. Absence of Barriers:**

* The maze generation algorithm systematically removes walls to create passages.
* No group of walls is arranged in a manner that would isolate a section of the maze from the start or end cell.
* Therefore, no impassable barriers exist between any two cells.

**5. Conclusion:**

* Since the maze forms a connected graph (spanning tree) with all cells reachable from one another, and there are no isolated sections:
  + **There exists at least one path** from the start cell (0,0) to the end cell (R-1, C-1).
  + **Uniqueness of the Path:** There is exactly one such path due to the tree structure of the maze.

**Thus, the maze generation algorithm guarantees the existence of a path from the start to the end cell.**

### **Summary**

* **DFS (solveMaze()):**
  + **Time Complexity:**
    - O(R×C)O(R \times C)O(R×C)
  + **Space Complexity:**
    - O(R×C)O(R \times C)O(R×C)
* **BFS (findPath()):**
  + **Time Complexity:**
    - O(R×C)O(R \times C)O(R×C)
  + **Space Complexity:**
    - O(R×C)O(R \times C)O(R×C)
* **Maze Generation Guarantee:**
  + The algorithm ensures that the maze is a perfect maze (spanning tree), thereby guaranteeing at least one unique path between any two cells, including from the start to the end cell.