Maze Generation and Resolution Algorithms

Maze Generation (Recursive Division)

The maze generation follows the **recursive division algorithm**. This algorithm is a form of binary tree maze generation where the maze is divided into two parts recursively, and walls are placed between the sections to create pathways. The procedure ensures that the maze will always have a single exit located at the bottom-right corner.

Steps for Maze Generation:

1. **Recursive Division:** The space is recursively divided by drawing walls. For each division:
   - A wall is placed, and a single door (passage) is carved through it.
   - The largest dimension (either width or height) is chosen for division, and the process continues on the resulting subspaces.
2. **Random Door Placement:** In each step, the algorithm randomly selects a location along the dividing wall and creates a "door" by not placing a wall in that spot.
3. **Termination:** The recursion continues until the dimensions of the current space are small enough (either width or height equals 1).

This approach results in a maze that is guaranteed to have a single solution and a clear exit. It is simple to implement and ensures that all parts of the maze are interconnected.

Algorithm Pseudocode:

```
function generate_maze(width, height):
    create initial empty space
    recursively divide space by walls:
        if the space is too small, stop
        else:
            pick a random position for a door
            divide the space into two subspaces
            repeat for each subspace
```

Why This Algorithm?

- **Guaranteed single exit:** Since the recursion divides the space until small subspaces are formed, the exit at the south-east corner is always reachable.
- **Simplicity:** The algorithm is straightforward and requires minimal computation.

- **Efficiency:** The recursive approach works well even for relatively large mazes.

Maze Resolution (Breadth-First Search)

To solve the maze, we use **Breadth-First Search (BFS)**, which is a well-known graph traversal algorithm that finds the shortest path between a starting vertex and a target vertex. BFS is ideal for maze resolution because it explores all possible paths level by level and guarantees the discovery of the shortest path in an unweighted grid-like structure.

Steps for BFS Resolution:

1. **Graph Representation:** The maze is represented as a graph where each cell in the maze corresponds to a vertex. The edges between vertices represent the possible movements (up, down, left, right).
2. **Queue Initialization:** BFS uses a queue to explore the maze. Starting from the initial vertex (a random position), we add it to the queue.
3. **Exploration:** For each vertex, we explore its neighboring vertices (up, down, left, right). If a neighboring vertex is not yet visited, it is added to the queue and marked with the current vertex as its parent.
4. **Termination:** BFS continues until the exit (south-east corner) is found. Once the exit is reached, the path is reconstructed by tracing back from the exit to the start using the parent pointers.
5. **Path Reconstruction:** The path to the exit is constructed by following the parent pointers from the exit back to the start. The path is then reversed to give the solution from start to exit.

Algorithm Pseudocode:

```
function find_exit(start, maze):
    initialize a queue and add the start vertex
    while the queue is not empty:
        current_vertex = dequeue
        if current_vertex is the exit:
            reconstruct path from exit to start using parent pointers
            return the path
        for each neighbor of current_vertex:
            if the neighbor is not visited:
                mark it visited and set its parent to current_vertex
                enqueue the neighbor
```

Why BFS for Resolution?

- **Shortest Path:** BFS guarantees that the first time we reach the exit, it is through the shortest path.
- **Simplicity:** The BFS algorithm is easy to implement and works well for finding the shortest path in grid-like mazes.
- **Efficiency:** BFS explores the maze systematically and ensures that all possible paths are explored in the shortest possible number of steps.

Conclusion

The combination of **recursive division** for maze generation and **Breadth-First Search (BFS)** for maze resolution is a powerful and efficient solution for this maze problem. The recursive division guarantees that the maze is well-structured, with only one exit, while BFS ensures that the shortest path is found in an optimal manner. These algorithms are chosen due to their simplicity, efficiency, and suitability for this type of maze problem.

This LaTeX code will format your text with appropriate sections, enumerations, and pseudocode for a well-structured document.