Description of the Chosen Data Structure and Its Formal Specification

Introduction

In this report, we present the data structure selected for representing a maze in our project, along with its formal specification. The objective was to design a lightweight and efficient structure that allows easy manipulation and rapid access to cell information, which is crucial for maze generation and pathfinding algorithms.

Maze Representation

We represent the maze as a two-dimensional matrix of integers. Each integer in the matrix corresponds to a cell in the maze and encodes multiple boolean properties using bitwise operations. This approach minimizes memory usage and enables quick querying and updating of cell properties.

Bitwise Encoding of Cell Properties

Each cell uses bits to represent the presence of walls and special markers. The significance of each bit (from least significant to most significant) is as follows:

- **Bit 0** (1): Wall on the **North** side.
- **Bit 1** (2): Wall on the **East** side.
- **Bit 2** (4): Wall on the **South** side.
- **Bit 3** (8): Wall on the **West** side.
- **Bit 4** (16): Cell is an **Entrance**.
- **Bit 5** (32): Cell is an **Exit**.
- **Bit 6** (64): Cell has been **Visited**.
- **Bit 7** (128): Cell is part of the **Path**.

Using bitwise operations allows efficient checking and modification of these properties for any cell.

Formal Specification

We define the maze data structure and its associated operations as follows.

Data Types

- **Maze**: The maze data structure.
- **Height**, **Width**: Positive integers representing the maze dimensions.
- **X**, **Y**: Integers representing cell coordinates.
- **Orientation**: One of $\{\text{North}, \text{East}, \text{South}, \text{West}\}$.
- **Boolean**: A boolean value (`True` or `False`).

Function Specifications

1. $\text{init}(h : \text{Height}, w : \text{Width}) \to \text{Maze}$
   Initializes a new maze with height h and width w.
2. $\text{open\_wall}(m : \text{Maze}, x : \text{X}, y : \text{Y}, o : \text{Orientation}) \to \text{Maze}$
   Removes the wall in cell (x, y) in the direction o.
3. $\text{set\_entrance}(m : \text{Maze}, x : \text{X}, y : \text{Y}) \to \text{Maze}$
   Marks cell (x, y) as the entrance.
4. $\text{set\_exit}(m : \text{Maze}, x : \text{X}, y : \text{Y}) \to \text{Maze}$
   Marks cell (x, y) as the exit.
5. $\text{divide}(m : \text{Maze}) \to (\text{Maze}, \text{Maze})$
   Divides the maze m into two sub-mazes.
6. $\text{fusion}(m_1 : \text{Maze}, m_2 : \text{Maze}, o : \text{Orientation}) \to \text{Maze}$
   Merges mazes m_1 and m_2 along orientation o.
7. $\text{destroy}(m : \text{Maze}) \to \text{Void}$
   Deallocates memory associated with maze m.
8. $\text{is\_open}(m : \text{Maze}, x : \text{X}, y : \text{Y}, o : \text{Orientation}) \to \text{Boolean}$
   Returns `True` if the wall in direction o at cell (x, y) is open.
9. $\text{is\_entrance}(m : \text{Maze}, x : \text{X}, y : \text{Y}) \to \text{Boolean}$
   Returns `True` if cell (x, y) is the entrance.
10. $\text{is\_exit}(m : \text{Maze}, x : \text{X}, y : \text{Y}) \to \text{Boolean}$
    Returns `True` if cell (x, y) is the exit.
11. $\text{size}(m : \text{Maze}) \to (\text{Height}, \text{Width})$
    Returns the dimensions of maze m.

Preconditions and Postconditions

- **Preconditions**:
  - Coordinates x and y must be within maze bounds: $0 \leq x < \text{Width}$, $0 \leq y < \text{Height}$.
  - Orientation o must be valid: $o \in \{\text{North}, \text{East}, \text{South}, \text{West}\}$.
- **Postconditions**:

- After open_wall, the wall at (x, y) in direction o is removed, and the corresponding wall in the adjacent cell is updated to maintain symmetry.
- After set_entrance or set_exit, cell (x, y) is marked appropriately.
- divide returns two valid sub-mazes that together represent the original maze.
- fusion returns a maze that combines m_1 and m_2 along o.

Conclusion

The selected data structure efficiently represents the maze using a compact matrix of integers and bitwise encoding. This design facilitates quick access and modification of cell properties, which is essential for the maze generation and solving algorithms. The formal specification provides a clear interface and ensures the correct manipulation of the maze through well-defined operations.