**BONUS Problem**

SOLN:

1. **WALL FOLLOWER**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Consider this maze,



start position = A

goal position as B.



|  |
| --- |
|  |

Let this be a compartment/ unit based on which a binary value list is defined, to determine its status of open region (which side is, and which is not blocked by a wall). The basic rule that this algorithm works on, is- following the Left Wall (or the Right).

Every time, it *first checks if the left side is open*, if it is then the mouse takes a left turn. If not, next it checks for the front side, then the right and finally the back (u- turn, if all three sides are blocked by walls). This data can be stored as a binary value.



For above compartment, binary value = 0111. The order of sides blockage checking (Left-Front-Right-Back) is represented as the digits, when read backwards. The 1st number is 1(as we are reading backwards), which implies that a wall exists on the left side. Similarly, we can determine that a wall exists on front and right as well.

**EXPLORATION STRATEGY/ ALGORITHM:**

Defining a function that aligns/remaps the wall’s sensor readings along the mouse’s perspective or the direction along which it is headed/ facing.

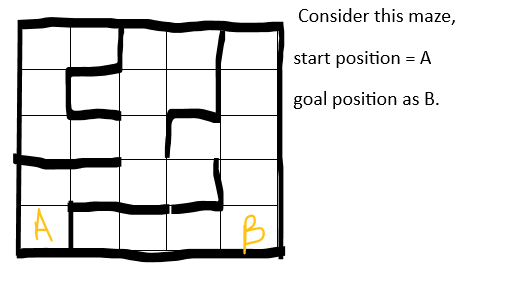
Next, another function is defined. It uses the turtle library to determine the graphic details (pen size, colour, position) of the turtle (the object/pen that is moving on the grid and marks the path that it has moved along).

Now inside the same function (above), the walls’ presence is checked for all sides (in a specific order), and this checking code block is run in a loop until it reaches the goal.

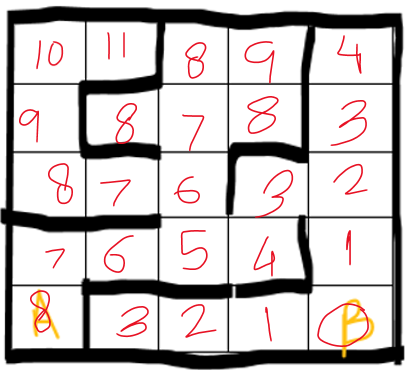
The functions are called, with suitable variables/ values.

1. **FLOOD- FILL ALGORITHM**

Consider the same grid:

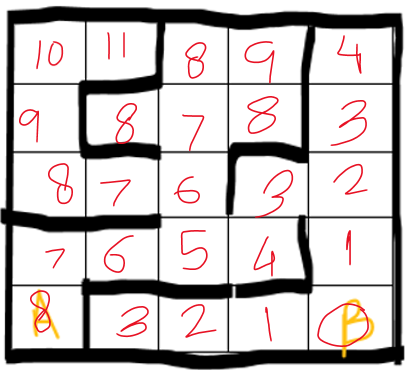


The idea of this algorithm is to assign value to each cell. A value of 0 is assigned to the goal position, and every farther cell is assigned a value higher to its adjacent one (as it goes farther from the goal position) unless a wall is present in between. For the grid above, the values assigned will be-





The mouse now has to follow a strategy which requires it to follow the cells that hold an assigned value lesser than the one it is currently present in.



This will be the path followed by the mouse in this grid. This is widely used in robotics (e.g., Micromouse competitions) due to its efficiency and adaptability to dynamic mazes. This way it guarantees shortest paths in unweighted grids.

**EXPLORATION STRATEGY/ ALGORITHM:**

First, it defines a function to calculate the shortest distances of the cells from the goal cell and assigns values to them accordingly. It does this by using the bfs algorithm

Then a 2nd function is defined to check whether a path exists, and then reconstruct the shortest path. The main idea is for the mouse to follow a cell that holds smallest value (among its current 4 neighbours) than its current cell value.

Then, the functions are called appropriately, and the grid, start and goal points are provided.