LEETCODE | 1926 |

DATE

Nearest Exit from Entrance in Maze

Input: maze = $\begin{bmatrix} + & + & . & + \\ . & . & . & + \\ + & + & + & . \end{bmatrix}$ entrance = [1,2]

Approach As we need to find the nearest Exit we need to travel all 4 directions so we need to go for Breadth First Search Algorithm (BFS)

Step1 int m = maze.length //rows
int n = maze[0].length //cols

Create a visited such as

maze = $\begin{bmatrix} + & + & . & + \\ . & . & . & + \\ + & + & + & . \end{bmatrix}$ visited = $\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

if (maze[i][j] == '+') → visited[i][j]=1

Step2 We know entrance row & column as [1, 2]

A. so create a class Pair as below

```
class Pair {
    int row;
    int col;
    int distance;
}
```

B. Create a Queue of type Pair as below

```
Queue<Pair> queue =
    new LinkedList<Pair>();
```

C. Add entrance coordinates to queue with distance = 0

```
queue.offer(new Pair
    (1, 2, 0)
```

entrance[0];
entrance[1];

Step3 Mark visited[entrance[0]][entrance[1]] = 1;

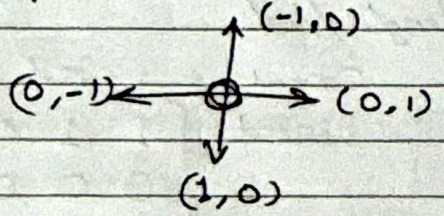
DATE

Step 4 Run a while loop (BFS pattern)

```
while (!queue.isEmpty()) {

    Pair current = queue.poll();
    int row = current.row;
    int col = current.col;
    int dist = current.direction;
    int distance = current.distance;
```

Step 5 We can move in four directions



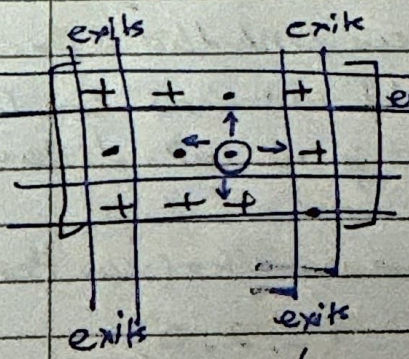

```
deltaRow = {-1, 0, 1, 0}
deltaCol = {0, 1, 0, -1}

direction = 4
```

Step 6

```
for (int i=0; i < direction; i++){
    int effRow = row + deltaRow[i];
    int effCol = col + deltaCol[i];

    // Secure Boundaries + check visited &
    // maze conditions here
```




```
    if (effRow >= 0 && effRow < m &&
        effCol >= 0 && effCol < n &&
        visited[effRow][effCol] == 0 &&
        maze[effRow][effCol] == '.') {

        // mark visited as 1
        visited[effRow][effCol] = 1;

        if (effRow == 0 || effRow == m-1 ||
            effCol == 0 || effCol == n-1) {
```

classmate PAGE

DATE

```
            return distance + 1; // offset from start
        }
        else {
            queue.offer(new Pair(effRow, effCol,
                                 distance+1));
        }
    }
}
return -1;
}
```