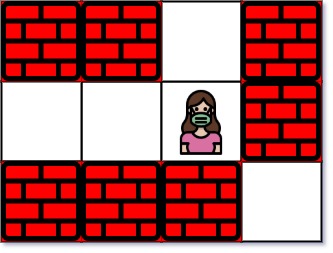
# Nearest Exit from Entrance in Maze

You are given an m x n matrix maze (**0-indexed**) with empty cells (represented as '.') and walls (represented as '+'). You are also given the entrance of the maze, where entrance = [entrancerow, entrancecol] denotes the row and column of the cell you are initially standing at.

In one step, you can move one cell **up**, **down**, **left**, or **right**. You cannot step into a cell with a wall, and you cannot step outside the maze. Your goal is to find the **nearest exit** from the entrance. An **exit** is defined as an **empty cell** that is at the **border** of the maze. The entrance **does not count** as an exit.

Return *the****number of steps****in the shortest path from the*entrance*to the nearest exit, or*-1*if no such path exists*.

**Example 1:**



**Input:** maze = [["+","+",".","+"],[".",".",".","+"],["+","+","+","."]], entrance = [1,2]

**Output:** 1

**Explanation:** There are 3 exits in this maze at [1,0], [0,2], and [2,3].

Initially, you are at the entrance cell [1,2].

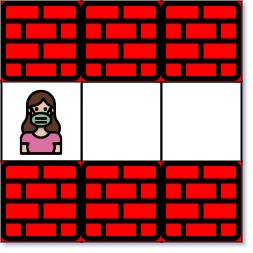
- You can reach [1,0] by moving 2 steps left.

- You can reach [0,2] by moving 1 step up.

It is impossible to reach [2,3] from the entrance.

Thus, the nearest exit is [0,2], which is 1 step away.

**Example 2:**



**Input:** maze = [["+","+","+"],[".",".","."],["+","+","+"]], entrance = [1,0]

**Output:** 2

**Explanation:** There is 1 exit in this maze at [1,2].

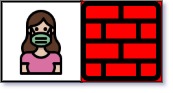
[1,0] does not count as an exit since it is the entrance cell.

Initially, you are at the entrance cell [1,0].

- You can reach [1,2] by moving 2 steps right.

Thus, the nearest exit is [1,2], which is 2 steps away.

**Example 3:**



**Input:** maze = [[".","+"]], entrance = [0,0]

**Output:** -1

**Explanation:** There are no exits in this maze.

**Constraints:**

* maze.length == m
* maze[i].length == n
* 1 <= m, n <= 100
* maze[i][j] is either '.' or '+'.
* entrance.length == 2
* 0 <= entrancerow < m
* 0 <= entrancecol < n
* entrance will always be an empty cell.

# Logic

1. Push the entrance to the queue.

2. Mark the entrance as visited.

3. While the queue is not empty:

For every node in the queue, visit all un visited adjacent nodes.

\* Try all 4 directions if available

\* If it is an exit, return the count of moves

\* Mark it is visited

\* Push it to the queue.

Increment Count

4. Return -1

#include <iostream>

#include <vector>

#include <queue>

#include <utility>

using namespace *std*;

template<typename T>

void fill2DMatrix(*vector*<*vector*<T>>& matrix)

{

T ele;

int rows = matrix.*size*(), cols = matrix[0].*size*();

for (auto i = 0; i < rows; i++)

{

for (auto j = 0; j < cols; j++)

{

*cin* >> ele;

matrix[i][j] = ele;

}

}

}

int nearestExit(*vector*<*vector*<char>>& maze, *vector*<int>& e)

{

*queue*<*pair*<int, int>> q;

// Push the entrance point on to Queue.

q.*push*(*make\_pair*(e[0], e[1]));

auto moves = 1;

int rows = maze.*size*();

int cols = maze[0].*size*();

*vector*<*vector*<int>> offsets = { {0, 1}, // Right

{1, 0}, // Down

{0, -1},// Left

{-1, 0} // Up

};

// Marking the entrance as visited.

maze[e[0]][e[1]] = '+';

while (!q.*empty*())

{

int len = q.*size*();

for (auto k = 0; k < len; k++)

{

auto s = q.*front*();

int i = s.*first*;

int j = s.*second*;

q.*pop*();

// Try all 4 directions.

for (auto p = 0; p < 4; p++)

{

int x = i + offsets[p][0];

int y = j + offsets[p][1];

// Out of bound or already visited.

if (x < 0 || y < 0 || x >= rows || y >= cols || maze[x][y] == '+')

continue;

// Reached the exit.

if (x == 0 || y == 0 || x == rows - 1 || y == cols - 1)

{

e[0] = x;

e[1] = y;

return moves;

}

// Mark the visited cell

maze[x][y] = '+';

// Push the visited cell on to Queue.

q.*push*(*make\_pair*(x, y));

}

moves++;

}

}

return -1;

}

int main()

{

int n, m;

*cin* >> n >> m;

*vector*<*vector*<char>> matrix(n, *vector*<char>(m));

fill2DMatrix(matrix);

int x, y;

*cin* >> x >> y;

*vector*<int> eNe = { x, y }; // To store entrance and exit.

auto ret = nearestExit(matrix, eNe);

if (ret != -1)

{

*cout* << "Number of moves: " << ret << "\nExit Point: "

<< eNe[0] << " " << eNe[1] << *endl*;

}

else

*cout* << "No exit point available!!!" << *endl*;

return 0;

}