United International **University**

Dept. of Computer Science & Engineering

Trimester: Spring 2024

Data Structure and Algorithms II Laboratory Report

Topic: Shortest Path in a Maze- Lee's Algorithm

Team Information

1. Rashedul Islam Rashed	**- 011221102**
2. Billal Hossain Hridoy	**- 011222280**
3. Md. Shahin	**- 011222251**
4. Md. Junayid Islam	**- 011222294**

Objective:

Design and implement Lee's algorithm to efficiently find the shortest path between two points in a maze.

What is Maze- Lee’s Algorithm?

Maze-Lee's Algorithm Is a pathfinding algorithm used to find the shortest path through a maze or grid from a starting point to a destination. It was developed by C.Y. Lee in 1961.

The algorithm is a breadth-first search based algorithm that uses queue to store the steps. It usually works first to choose a starting point and add it to the queue. Add the valid neighboring cells to the queue. Remove the position you are on from the queue and continue to the next element. Repeat those steps until the queue is empty.

Why do we need to solve this problem in Computer Science?

Lee’s algorithm solves the path-connection problems that arise in logical drawing, wiring diagramming or optimal route finding. Its parallel version has been widely used as a benchmark to test transactional memory systems. It exhibits transactions of large size and duration that stress these systems exposing their limitations. In fact, Lee’s algorithm has been proved to perform similarly to sequential in commercial hardware transactional memory systems due to persistent capacity overflows.

Lee's Algorithm is widely used for:

- Pathfinding in Robotics and Autonomous Systems.
- Computer Games and Simulations.
- Network Routing.
- Optimization Problems.
- Algorithm Design and Analysis.
- Educational Purposes.

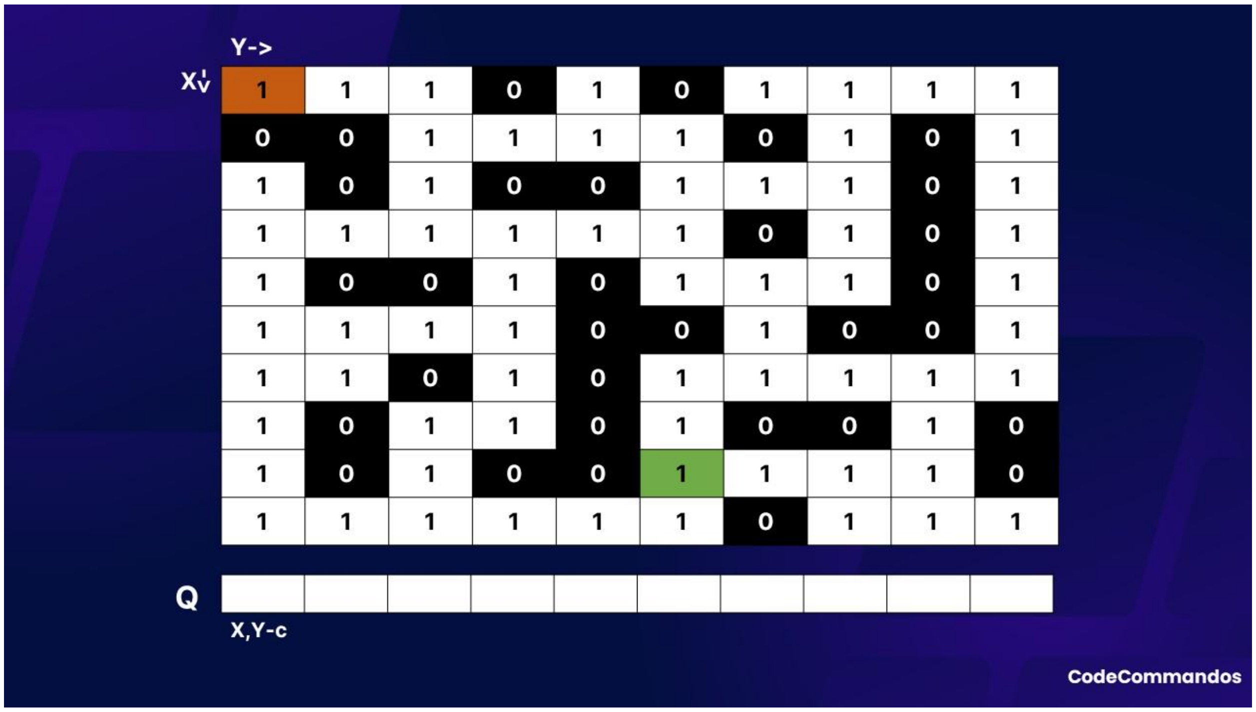
Problem Statement:

In this problem, we are provided with a maze in the form of a binary rectangular matrix and our task is to **find the shortest path between a given source cell to a destination cell**. The maze is represented as a M*N matrix where each element can either be 0 or 1. A path can be created out of a cell only if its value is 1. At any given time, we can only move one step in one of the 4 directions.

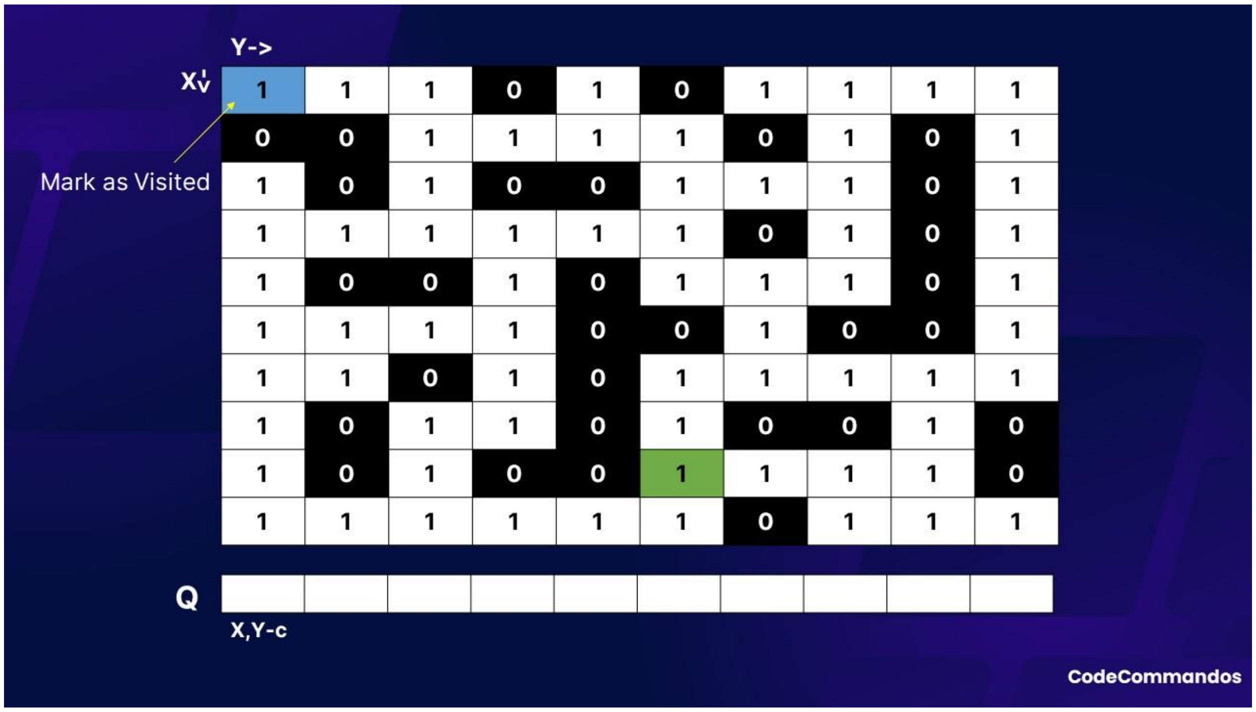
Lee’s Algorithm(Pseudocode):

```
Initialize movement arrays=>
rowNum := [-1, 0, 0, 1], colNum := [0, -1, 1, 0]

bfsLee(mat, src, dest)

    if mat[src.x][src.y]!=1 or mat[dest.x][dest.y]!=1, then
        return -1


    Initialize matrix: visited[Row][Col]:= False
    visited[src.x][src.y] := True
    Initialize empty queue: q := Queue()

    q.add(src)

 while q is not Empty, loop

        curr := q.Pop()

        pt := curr.coordinates

        if pt.x = dest.x and pt.y = dest.y, then
            return curr.distance


        for i in 0 to 4
            row := pt.x + rowNum[i]
            col := pt.y + colNum[i]
            if mat(row, col) is valid and mat(row,col) = 1 and mat(row,col) is not visited, then
                visited[row][col] := True
                q.add(current_Adjacent_col)

     return -1
```

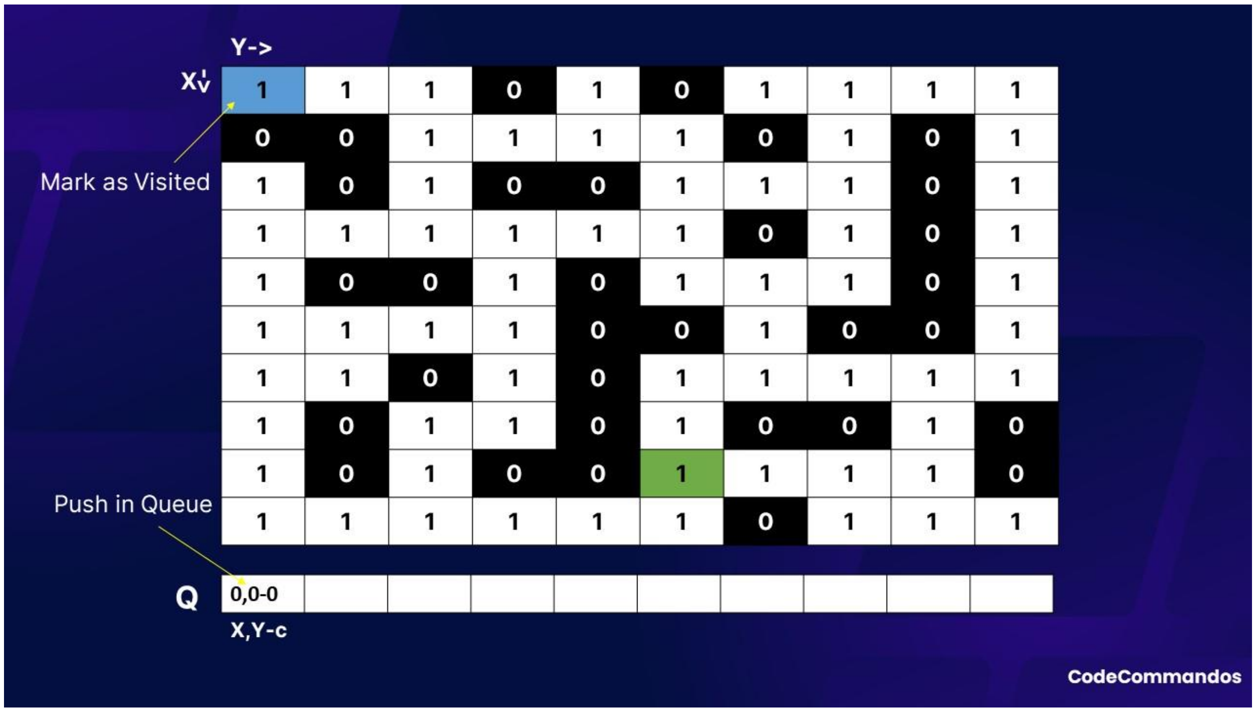
Simulation:



1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

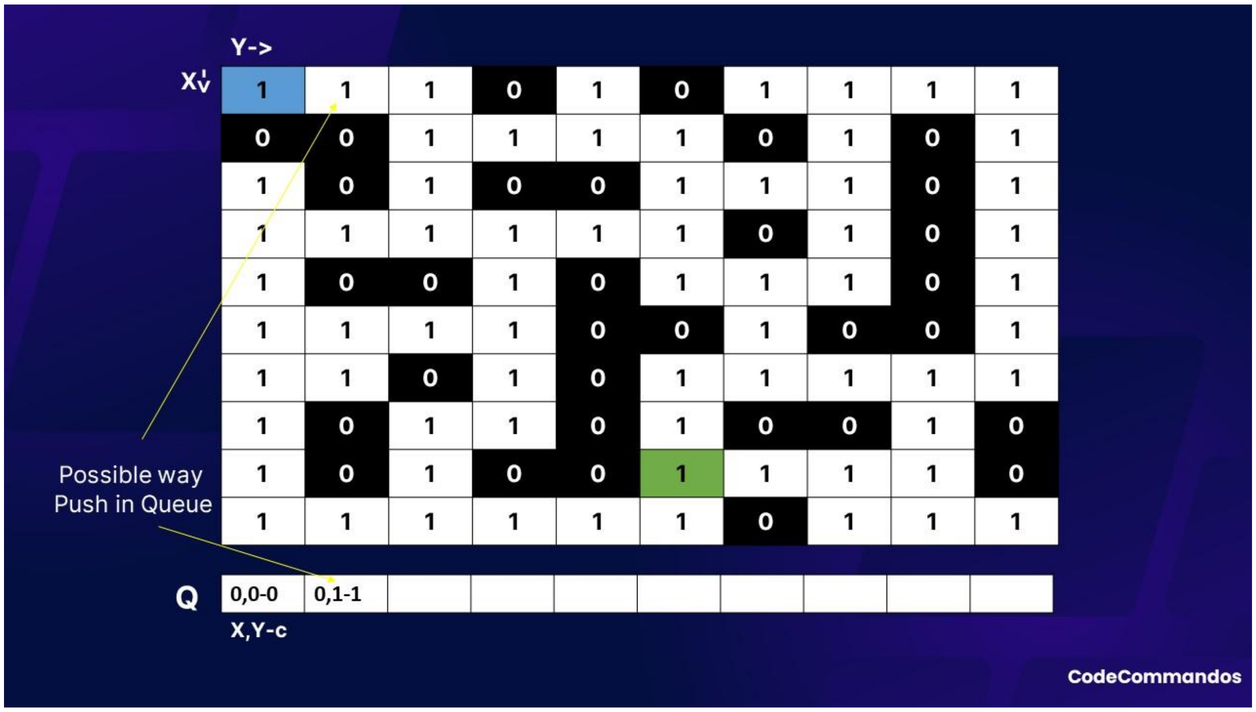


1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1



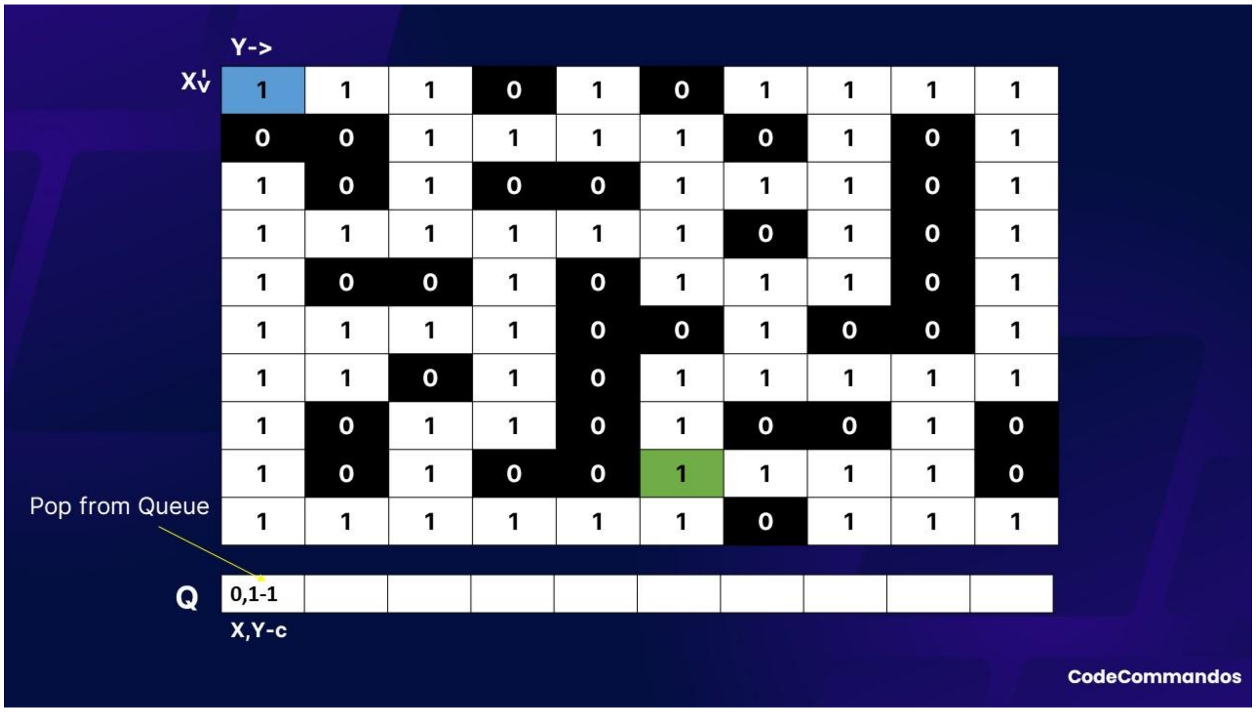
1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

0,0-0									

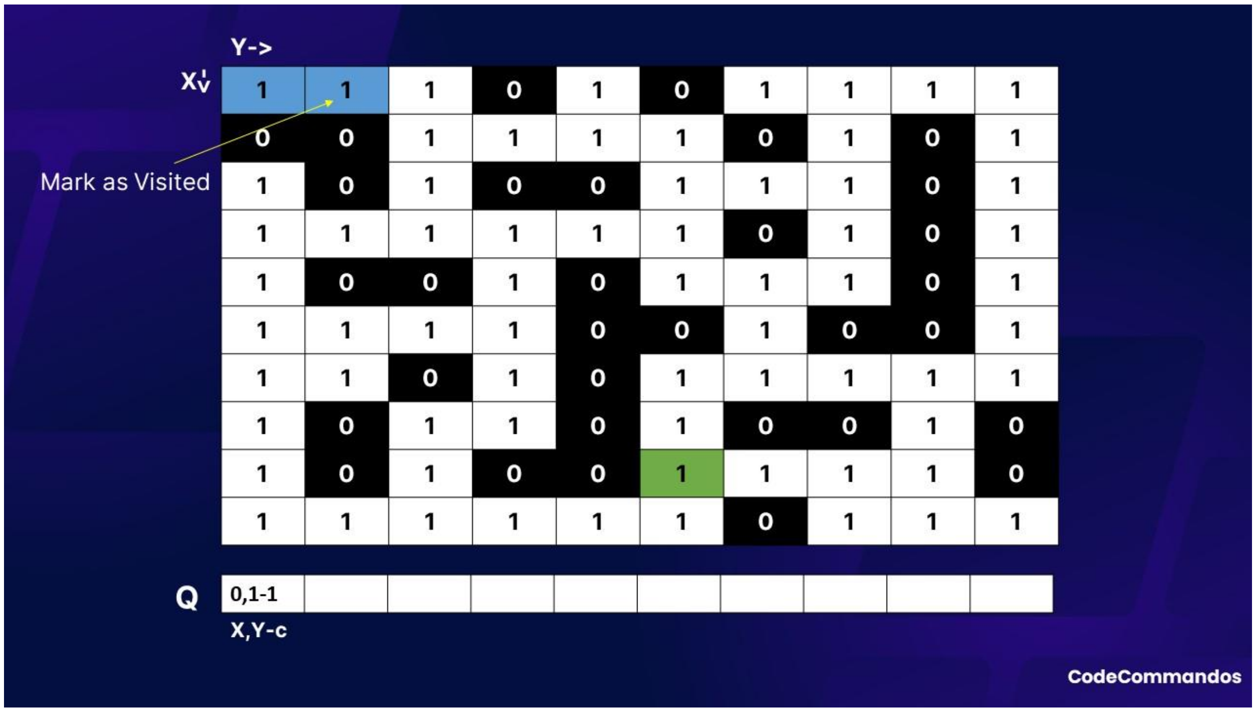


1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

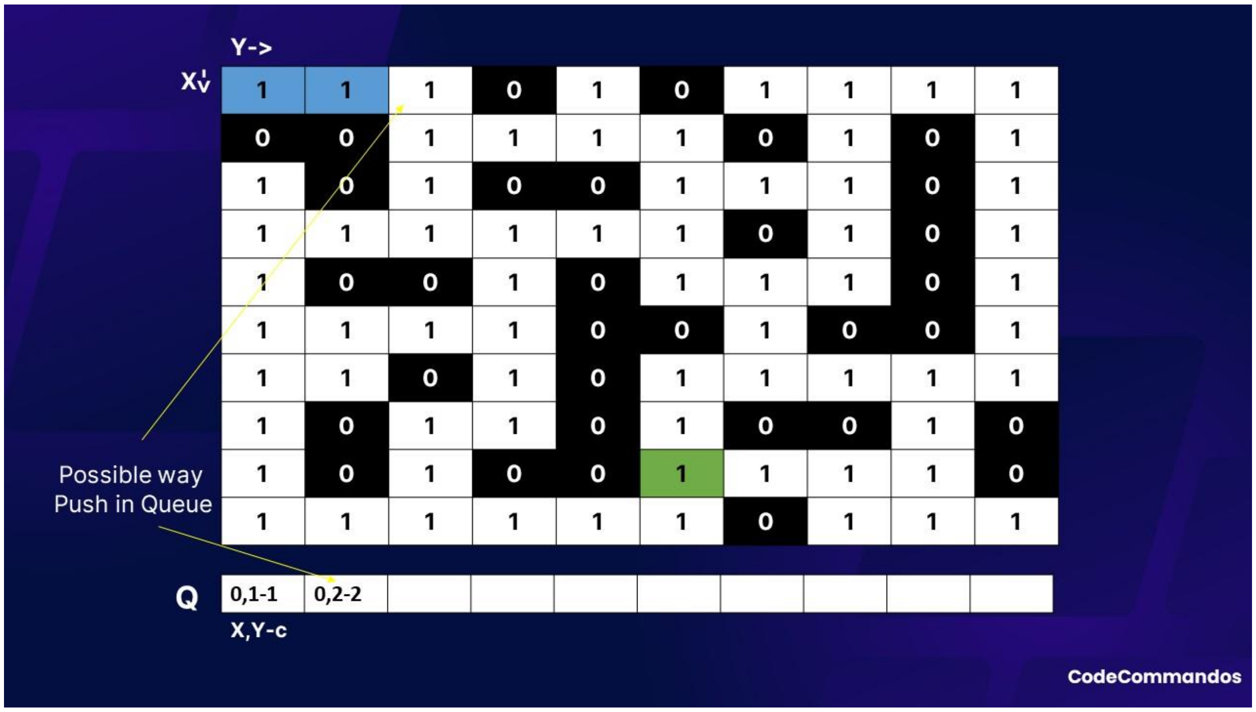
0,0-0	0,1-1								



1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1



1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1


Y->

X↓

1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

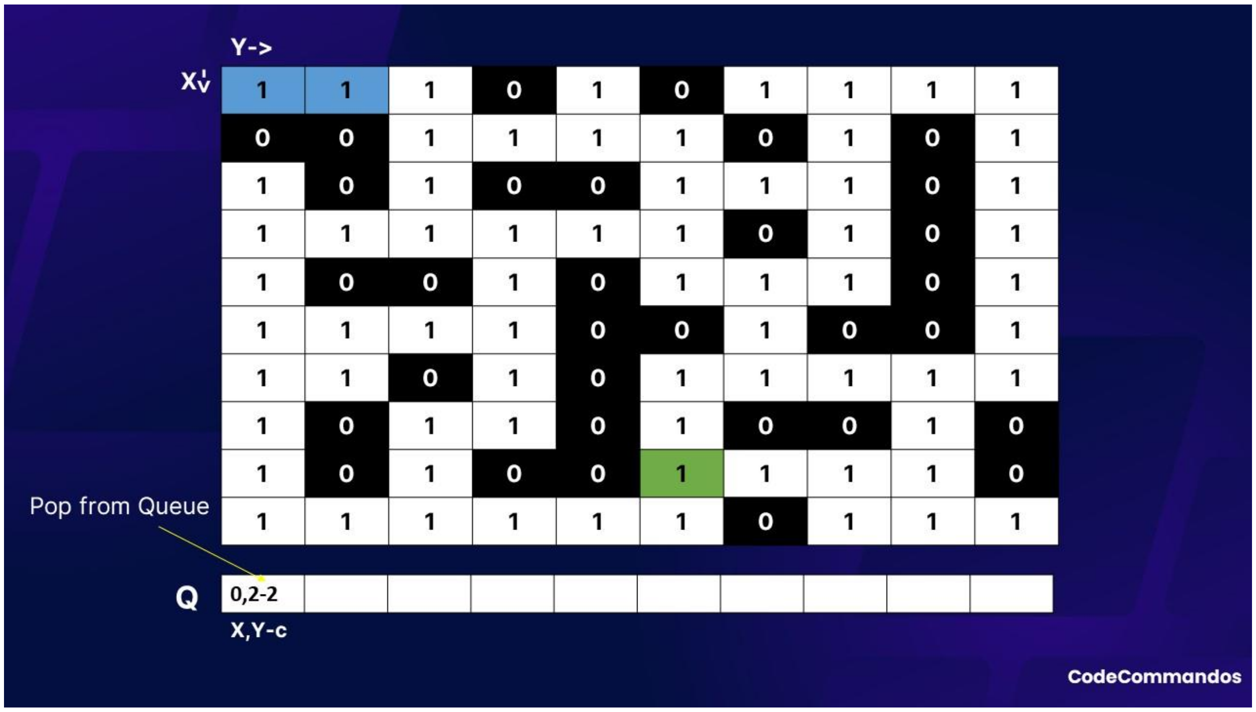
Possible way
Push in Queue

Q

0,1-1	0,2-2								

X,Y-c

CodeCommandos


Y->

X↓

1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

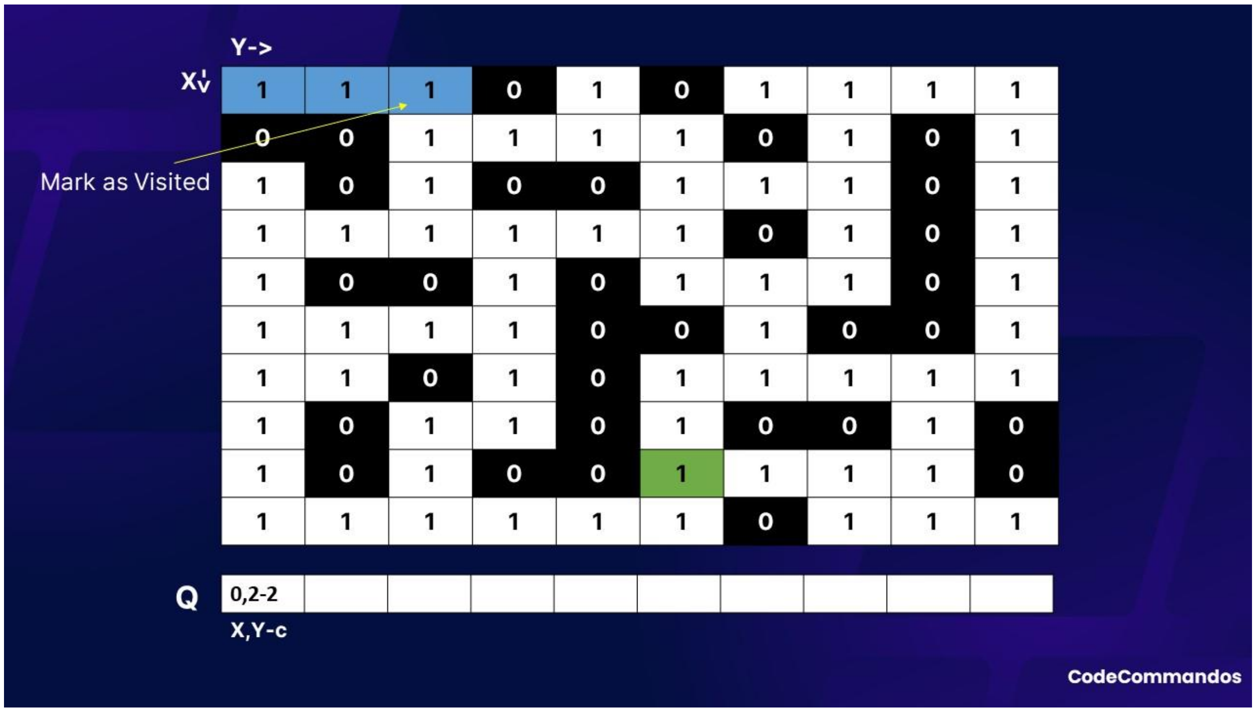
Pop from Queue

Q

0,2-2									

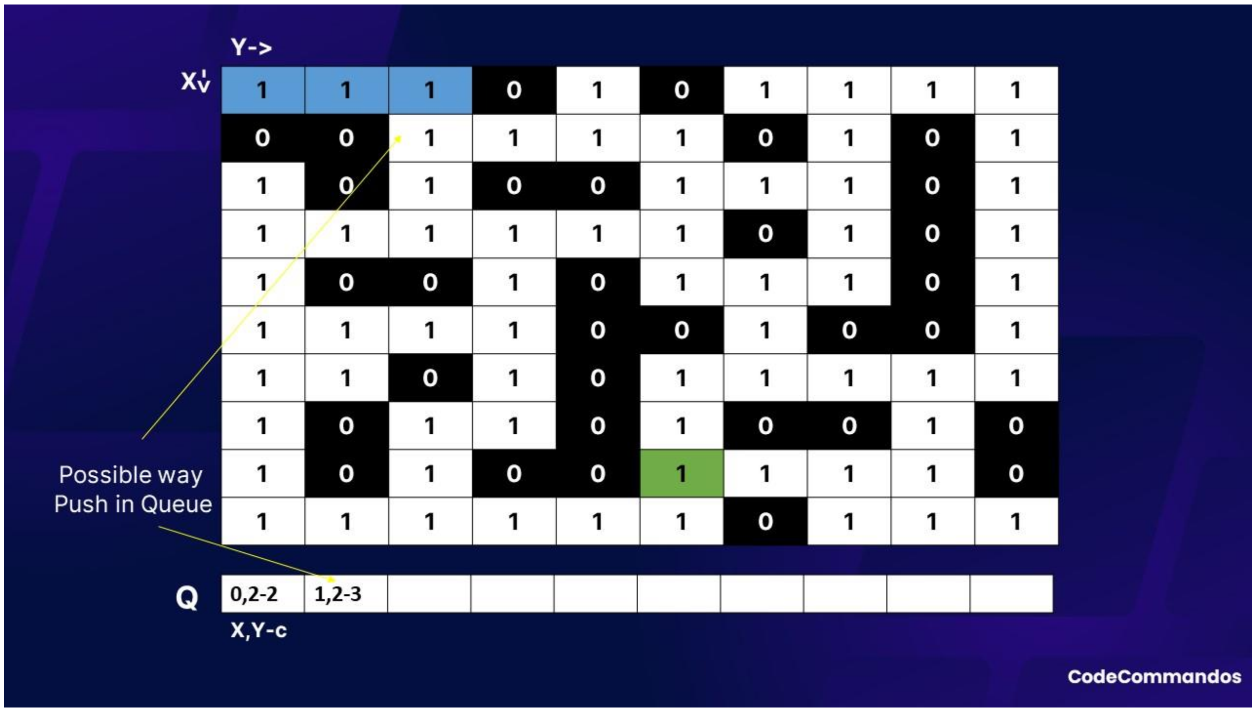
X,Y-c

CodeCommandos



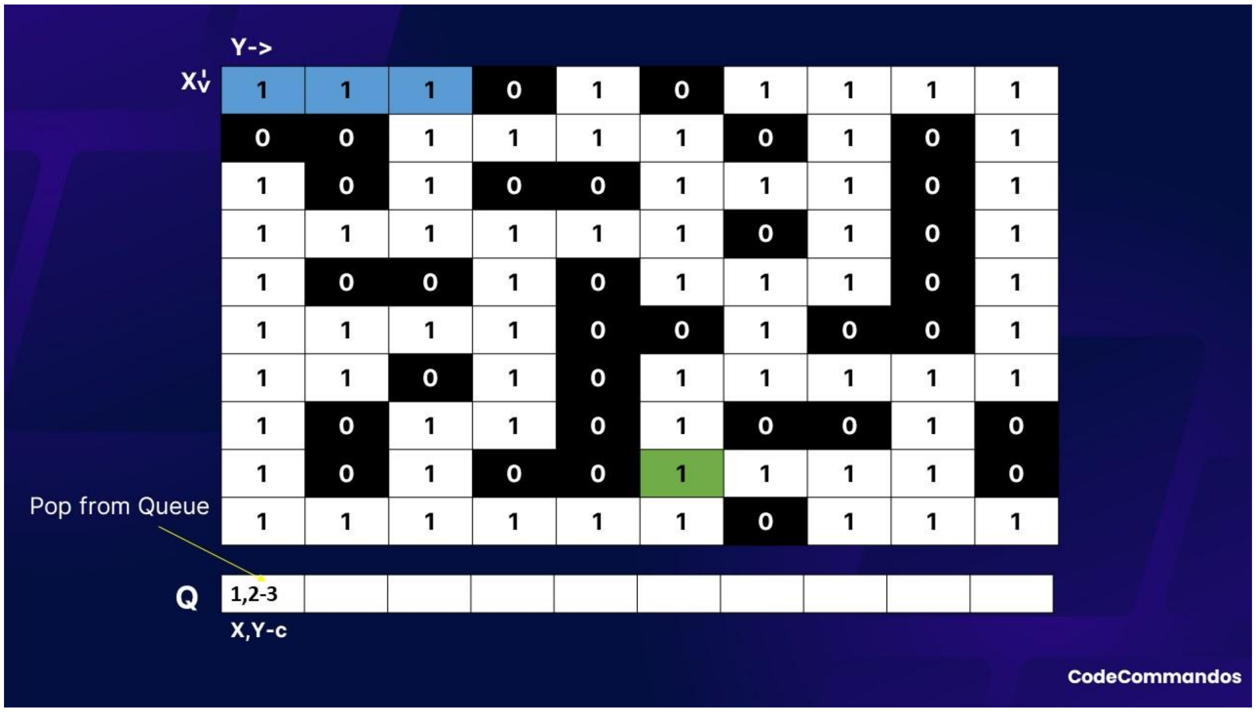
1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

0,2-2									

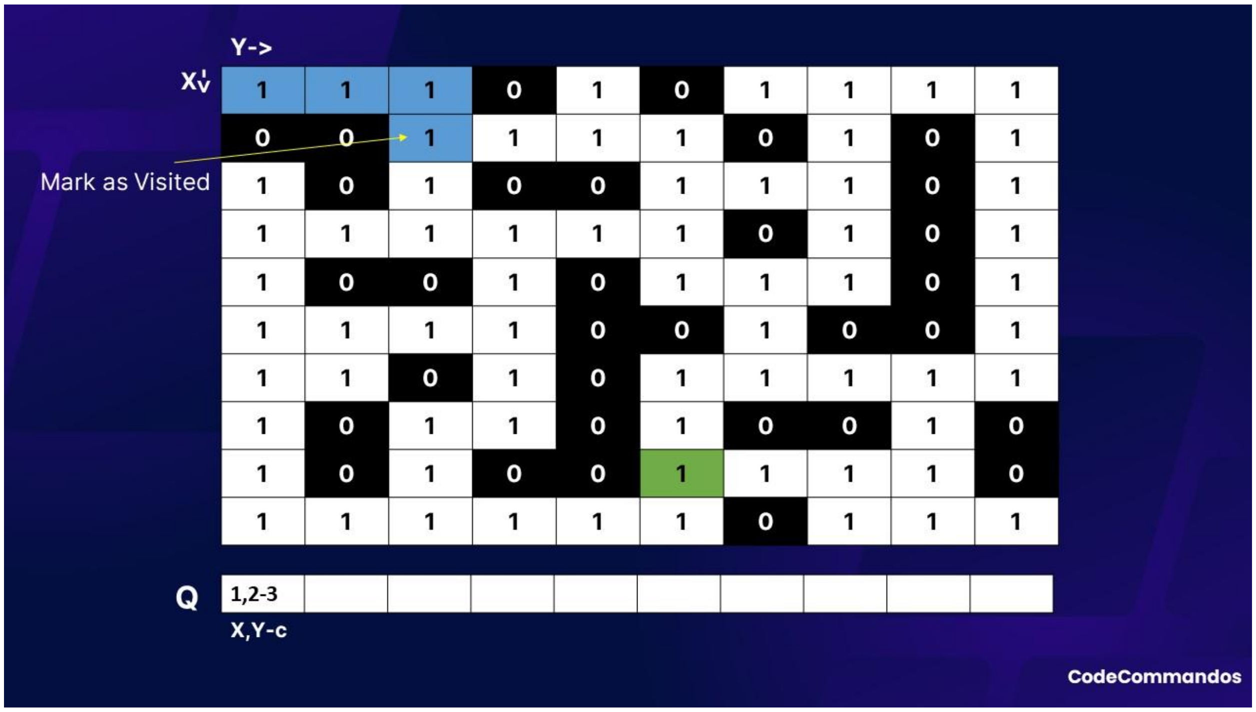


1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

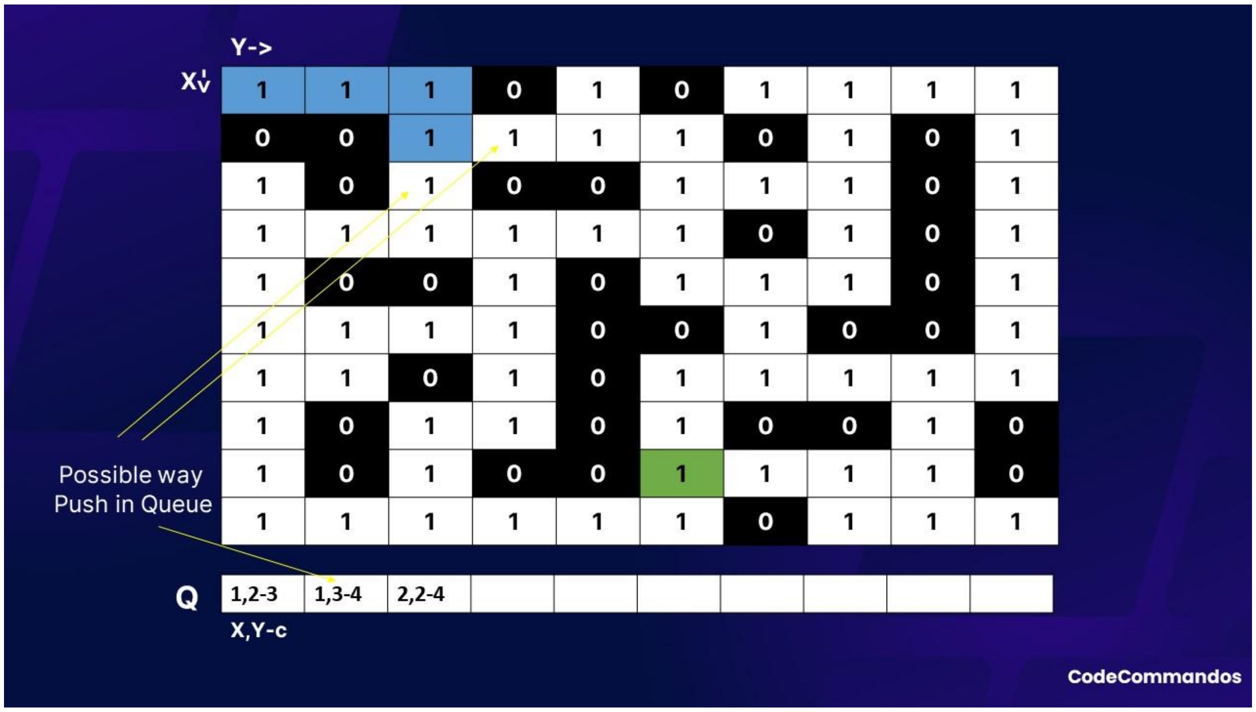
0,2-2	1,2-3								



1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

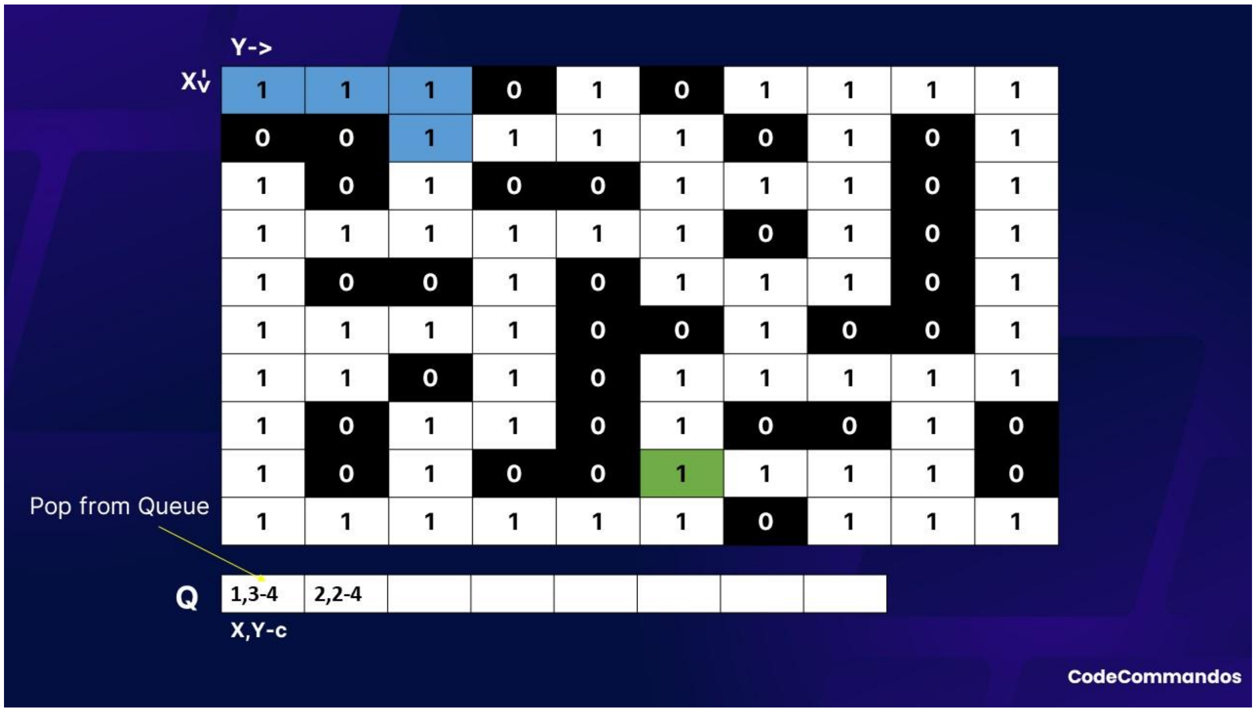


1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1



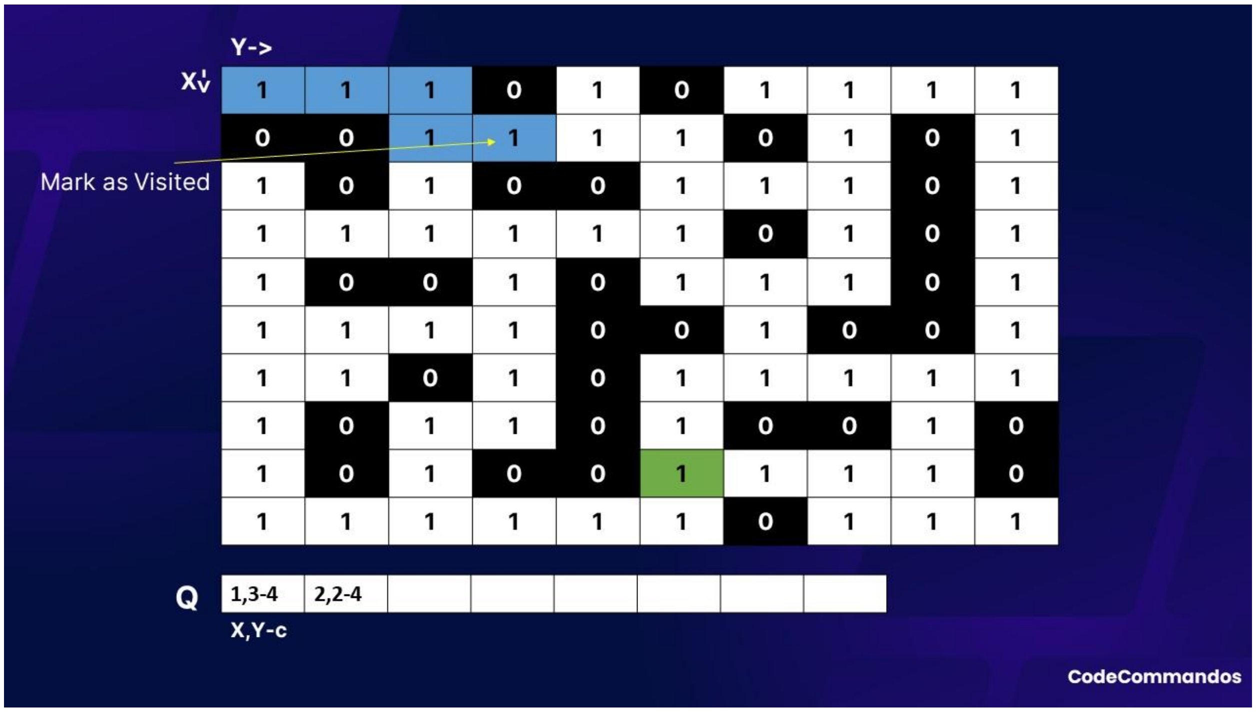
1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

1,2-3	1,3-4	2,2-4							



1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

1,3-4	2,2-4						



Y->

X↓

Mark as Visited

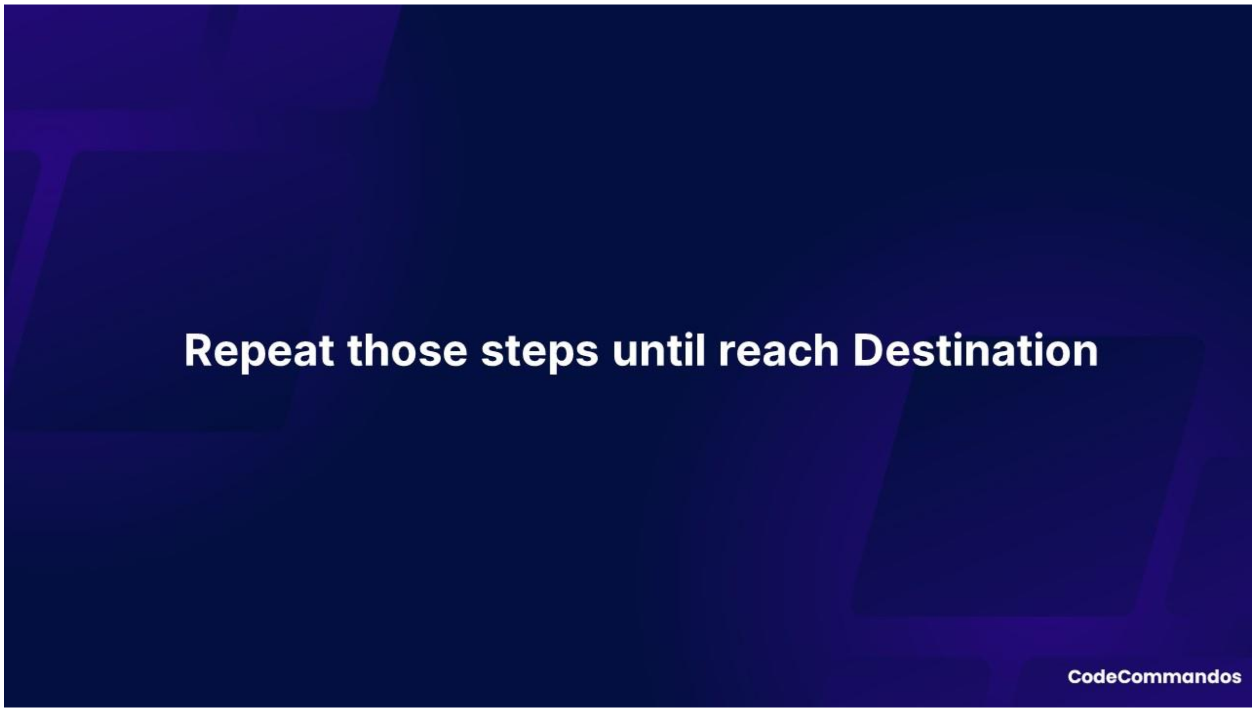
1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

Q

1,3-4	2,2-4						

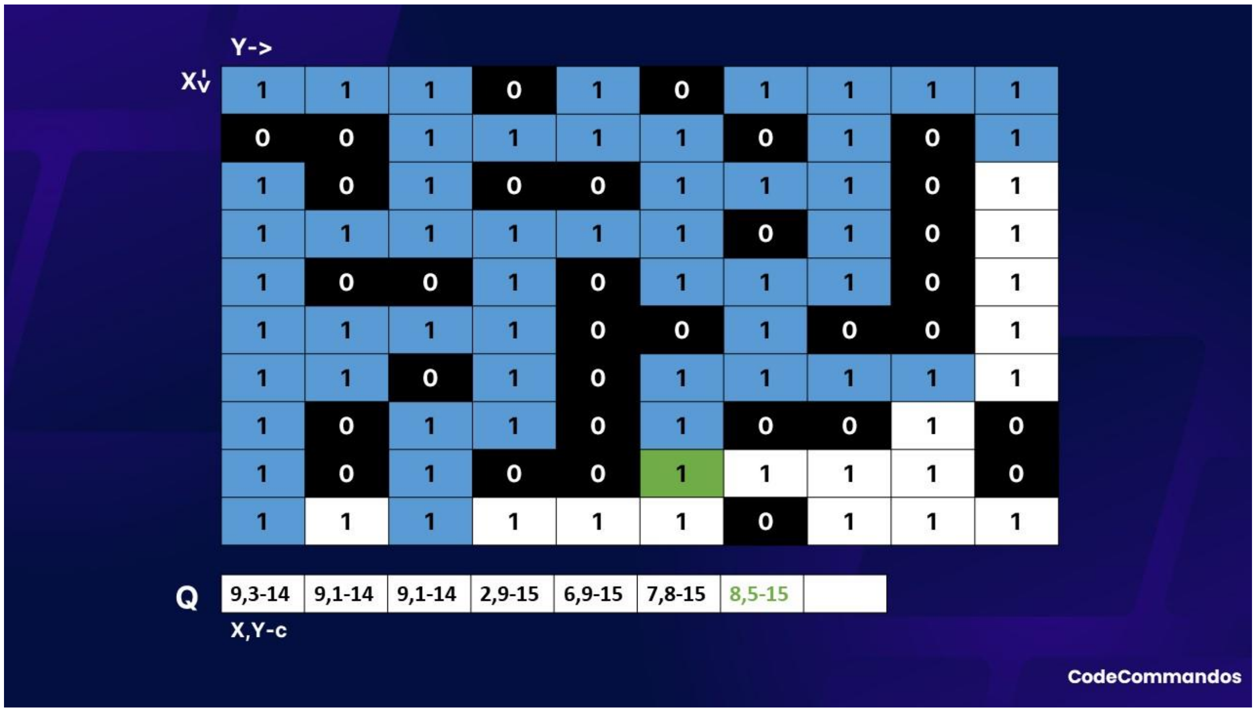
X,Y-c

CodeCommandos



Repeat those steps until reach Destination

CodeCommandos



Y->

X↓

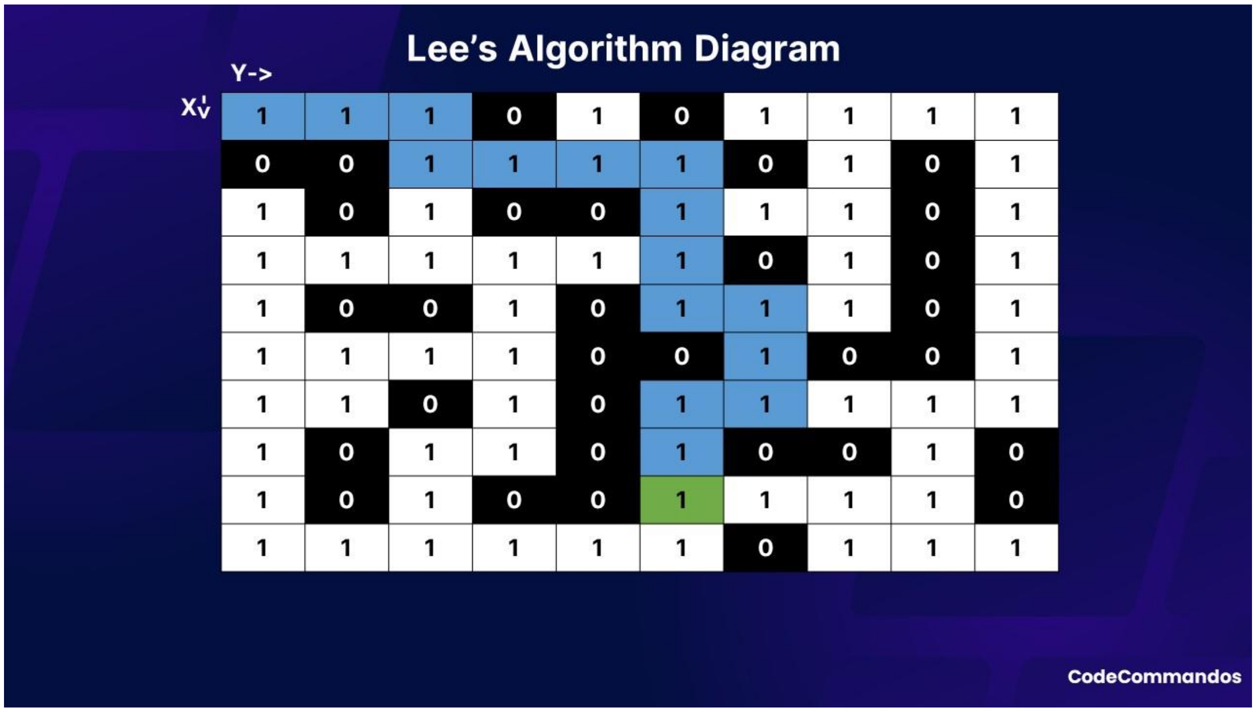
1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

Q							
	9,3-14	9,1-14	9,1-14	2,9-15	6,9-15	7,8-15	8,5-15

X,Y-c

CodeCommandos

Hence, The shortest path:



Lee's Algorithm Diagram

Y->

X↓

1	1	1	0	1	0	1	1	1	1
0	0	1	1	1	1	0	1	0	1
1	0	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	1	0	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	1	1	1

CodeCommandos

Difference between the Lee's Algorithm and Brute Force:

Brute Force Approach:

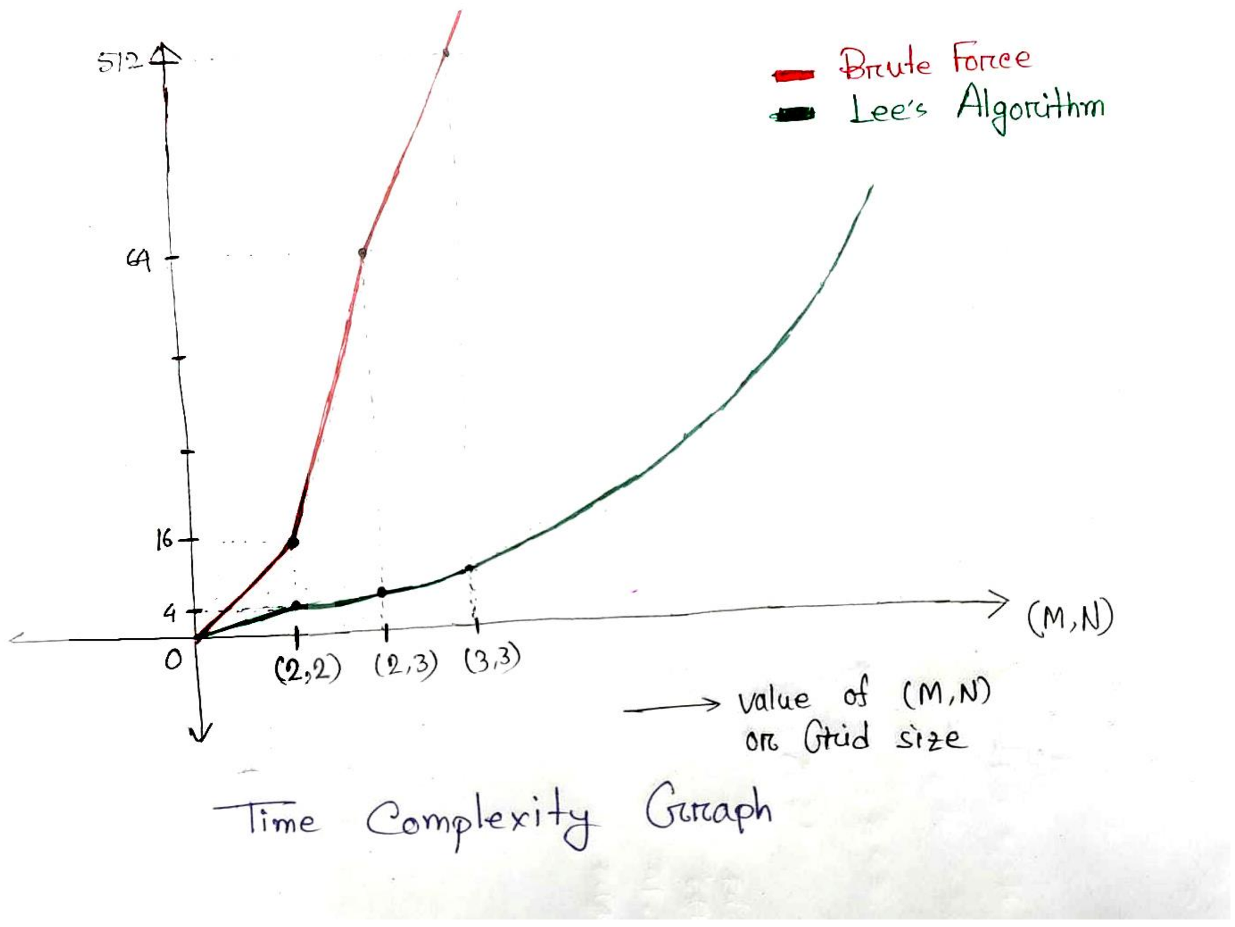
- In a brute force approach, would typically explore all possible paths from the source to the destination without any particular strategy.
- This might involve recursively trying all possible moves from each cell until the destination is reached or all possibilities are exhausted.
- There's no guarantee of finding the shortest path using brute force unless all possible paths are evaluated.

Lee's Algorithm Approach:

- It utilizes a queue-based BFS traversal from the source cell to the destination cell, exploring all possible paths in a systematic manner.
- It maintains a visited array to keep track of cells that have already been visited to avoid revisiting them.
- The algorithm guarantees to find the shortest path if it exists, due to the nature of BFS traversal.

Main Differences: Efficiency, Completeness, Time Complexity.

Time complexity graph with various parameters:



Time Complexity of Maze Lee's Algorithm: O (M*N)

Time Complexity of Brute Force Algorithm: O (2^{M*N})

Drawback of Maze Lee's Algorithm:

- **Memory Intensive:** Lee's Algorithm requires storing a considerable amount of information, particularly when dealing with large or complex mazes, leading to significant memory consumption.

- **Inefficient for Large Mazes:** In very large mazes, Lee's Algorithm may become inefficient due to its need to explore and evaluate all possible paths, resulting in longer computation times.

- **Limited Path Flexibility:** The algorithm may not be suitable for mazes with complex layouts or obstacles that require more flexible pathfinding approaches, as it strictly follows a breadth-first search strategy.

- **Not Suitable for Dynamic Environments**: Lee's Algorithm is designed for static environments and may not adapt well to dynamic changes in the maze layout or obstacles during runtime.

- **Difficulty Handling Multiple Solutions:** In scenarios where multiple shortest paths exist between two points, Lee's Algorithm may not efficiently handle the exploration of all these paths, potentially leading to suboptimal or incomplete solutions.

Conclusion:

The Lee algorithm, a specialized form of Breadth-First Search, offers a powerful solution to the problem of finding the shortest path in a grid with obstacles. While the brute force approach may seem intuitive, it quickly becomes impractical due to its exponential time complexity. The Lee algorithm, on the other hand, efficiently navigates the grid, guaranteeing the shortest path if it exists. Its significance in computer science lies not only in its practical application in various fields such as robotics, gaming, and network routing but also in its embodiment of algorithmic principles like optimization and problem-solving efficiency.

Through our exploration of the brute force and Lee algorithms, we've seen how the Lee algorithm's systematic approach outperforms brute force, particularly in terms of time complexity and reliability. By employing a queue-based traversal and maintaining a visited array, the Lee algorithm navigates the grid efficiently, ensuring the shortest path is found without redundant exploration.