

Maze

# Table of Contents

- Introduction
- Problem Description
- Identify Problem
- Investigation
- Solution
- Objective

- **Depth First Traversal-Maze**
- **Implementation**
- **Approach 1 Depth First Search : TREE**
- **Python - code**
- **Test**
- **Enhancement Ideas**
- **Conclusion**
- **References**

# Problem Description

- **Robot - Clear Route (Street, Highway)**

The planner's problem is then to find a sequence of stances in  $L$  which take the robot from the start to the goal, such that the stances and the surfaces. It requires a clear path from start and can go as far as it can, and backtracks until it finds an unexplored path and then explore it step by step.

## **Self Driving Car- Unclear Route (Hotel, Hospital)**

While the self driving car detects curbs and other vehicles when parking. Its like while rolling in an empty space until it detects any hurdle or curb. Self driving cars are looking for available space move in that direction until it achieve its desired path.

# Introduction

- Breadth First Approach has been used to get the shortest path for ball. Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration

# Identify Problem

1 = Walkable Path

0 = Non-walkable Path

S = Starting point (start will be (0,0))

D = Destination

When you see problem first thing should come to your mind is what if starting point is not walkable(0) if it is not walkable then you should immediately return False or Not possible.

If the ball doesn't stop at destination because the ball is rolling through all cell, It means there is no possible way for ball.

# Investigation

We'll solve with two different methods:

1. Depth first search
2. Breadth first search

If you're not concerned about memory, you can pick either.

They have similar running time, but either may greatly outperform the other on any given problem simply due to the order in which the cells are visited.

However, if you're looking for the shortest path to some given cell in a general grid, use BFS , as that guarantees to find the shortest path



# Solution

When a maze has multiple solutions, the solver may want to find the shortest path from start to finish.

There are several algorithms to find shortest paths, most of them coming from graph theory.

One such algorithm finds the shortest path by implementing a breadth first -search, while another, the A\* Algorithm, uses a heuristic technique

# Objective

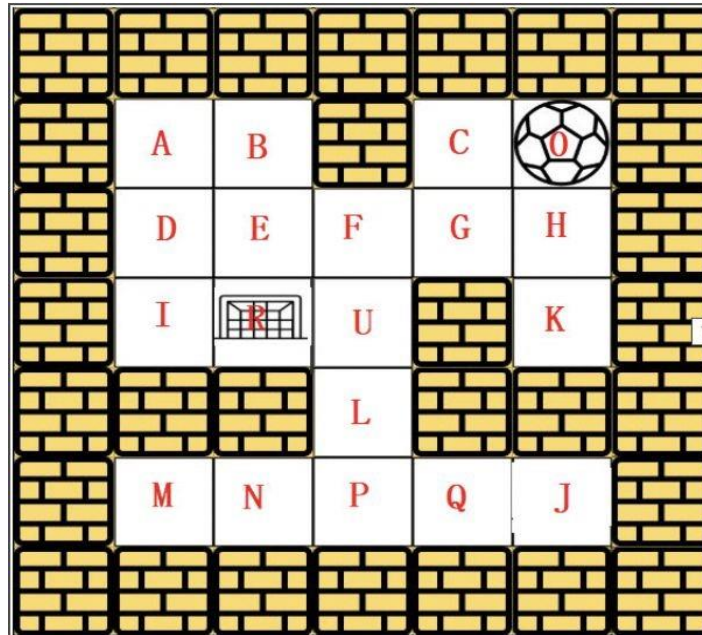
There is a ball in maze with empty spaces (represented as 0) and wall(represented as 1).

Our goal is to find the shortest path for ball through empty spaces by rolling up, down, left, or right, but it won't stop the rolling until hitting a wall .

When the ball stops , it could choose the next direction

# Implementation

## Approach 5: Wheeled robots move in a Hotel: BFS



# Process

Visited : 0

Queue: 0

1.Add 0 to queue

2. Mark 0 as visited

Visited : 0

Queue : 1

1. Remove 0 from Queue

2. Print 0

Visited : 0   C   K

1   1   1

Queue : C   K

1. Add C and K to queue

2. Mark C and K as visited

Visited : 0 C K

1 1 1

Queue: K

1. Remove C from Queue

2. Print 0 C

Visited : 0 C K G

1 1 1 1

Queue : K G

1. Add G to queue

2. Mark G as visited

Visited : 0 C K G

1 1 1 1

Queue: G

1. Remove K from Queue

2. Print 0 C K

Visited : 0 C K G

1 1 1 1

Queue :

1. Remove G from queue

2. Print 0 C K G

Visited : 0 C K G D

1 1 1 1 1

Queue: D

1. Add D to the Queue

2. Mark D as visited

Visited : 0 C K G D

1 1 1 1 1

Queue:

1. Remove D from queue

2. Print 0 C K G D



Visited : 0 C K G D A I

1 1 1 1 1 1 1

Queue: A I

1. Add A I to the Queue

2. Mark A I as visited

Visited : 0 C K G D A I

1 1 1 1 1 1 1

Queue : I

1. Remove A from queue

2. Print 0 C G D A

Visited : 0 C K G D A I B  
1 1 1 1 1 1 1 1

Queue: I B

1. Add B to the Queue

2. Mark B as visited

Visited : 0 C K G D A I B R  
1 1 1 1 1 1 1 1 1

Queue : B R

1. Add R to Queue

2. Mark R as Visited

Visited : 0 C K G D A I

B R

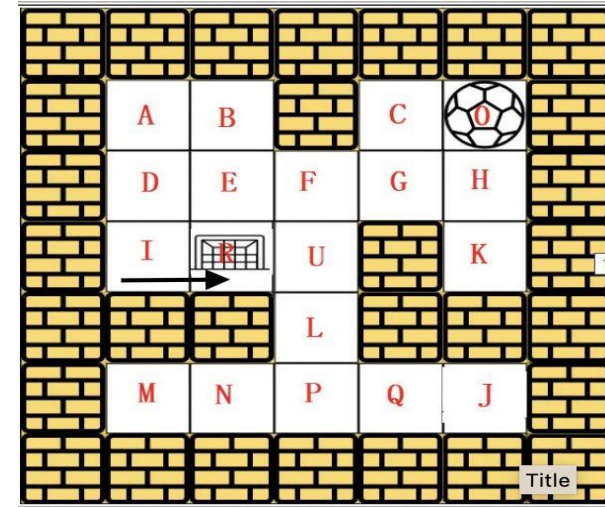
1 1 1 1 1 1 1 1 1

Queue : R

1. Remove B from the  
Queue

2. Print 0 C K G A I B

3. Reached destination R



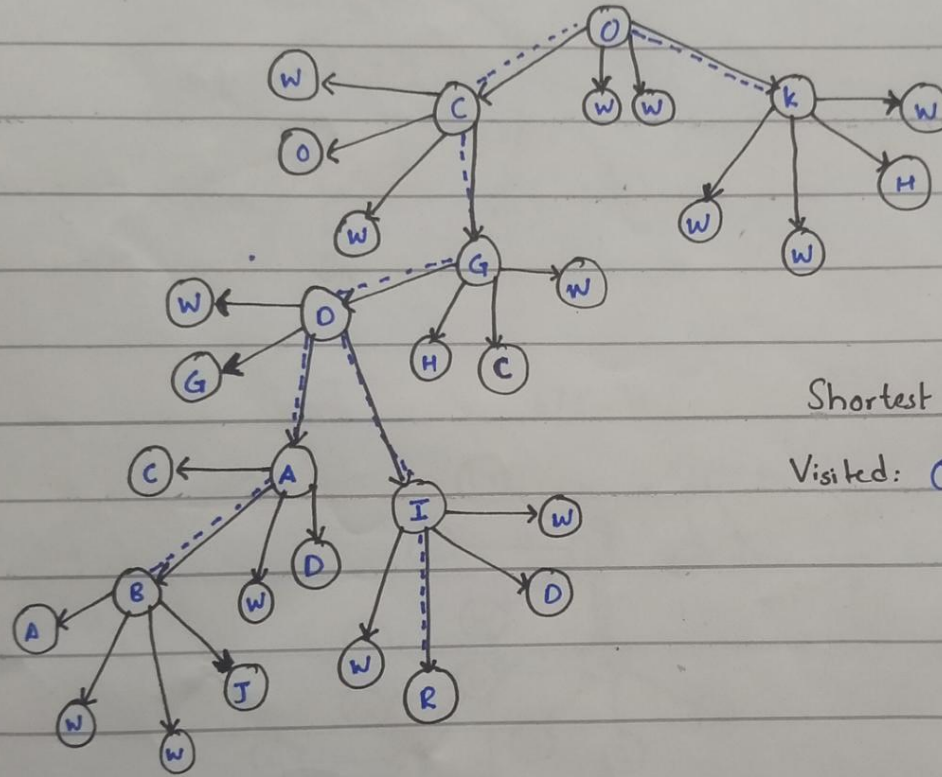
# Result

The ball can go through the empty spaces by rolling up, down, left, right, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

In the above case we found the possible from 0 to solution because it pass through the destination which means ball has visited target.

Shortest path: 0 C K G D A I B R


# BFS - TREE



Shortest Path

Visited: OCKGDAIBR

# Python Code

 Maze-40.py - C:\Users\hp\Desktop\Jawad-MS-Data\Sem-2\ALGO\Maze-40.p

File Edit Format Run Options Window Help

```
from collections import deque
from typing import List

class Solution:
    def hasPath(self, maze: List[List[int]], start: List
        m = len(maze)
        n = len(maze[0])
        dirs = [0, 1, 0, -1, 0]
        q = deque([(start[0], start[1])])
        seen = {(start[0], start[1])}

    def isValid(x: int, y: int) -> bool:
        return 0 <= x < m and 0 <= y < n and maze[x][y]

    while q:
        i, j = q.popleft()
        for k in range(4):
            x = i
            y = j
            while isValid(x + dirs[k], y + dirs[k + 1]):
                x += dirs[k]
                y += dirs[k + 1]
            if [x, y] == destination:
                return True
            if (x, y) in seen:
                continue
            q.append((x, y))
            seen.add((x, y))

    return False
```

# Result

```
collections import deque
typing import List

s Solution:
f hasPath(self, maze: List[List[str]], start: List[int], destination: List[int]) -> bool:
    m = len(maze)
    n = len(maze[0])
    dirs = [0, 1, 0, -1, 0]
    q = deque([(start[0], start[1])])
    seen = {(start[0], start[1])}

    def isValid(x: int, y: int) -> bool:
        return 0 <= x < m and 0 <= y < n and maze[x][y] != 'X'

    while q:
        i, j = q.popleft()
        for k in range(4):
            x = i
            y = j
            while isValid(x + dirs[k], y + dirs[k + 1]):
                x += dirs[k]
                y += dirs[k + 1]
                if [x, y] == destination:
                    return True
            q.append((x, y))
            seen.add((x, y))
    return False
```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, AMD64) on win32  
Type "help", "copyright", "credits" or "license()"

```
>>>
===== RESTART: C:\Users\hp\Desktop\Jawad-MS-Dat
True
False
False
>>> |
```

# Enhancement Ideas

The problem can be solved by using using different approaches i-e Dijkstra's algorithm, Bellman Ford's algorithm, breadth first approach and Depth first approach etc.

However, shortest path is required thats why BFS has been considered to solve this problem.



# Conclusion

## Time complexity

**$O(m\ n)$**  Complete traversal of maze will be done in the worst case. Here,  $m$  and  $n$  refers to the number of rows and columns of the maze.

## Space complexity

**$O(m\ n)$**  visited array of size  $m\ n$  is used and queue size can grow up to  $m\ n$  in worst case.

The breadth-first search algorithm uses a queue to visit cells in increasing distance order from the start until the finish is reached.

Each visited cell needs to keep track of its distance from the start or which adjacent cell nearer to the start caused it to be added to the queue.

Hence, in the given example we are unable to find the shortest path.

# References

[BFS](#)

[The Maze](#)

[Leet Code 490](#)