Previous Next

490. The Maze

May 22, 2017 | 57.8K views

Average Rating: 4.08 (48 votes)

There is a **ball** in a maze with empty spaces and walls. The ball can go through empty spaces by rolling **up**, **down**, **left** or **right**, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the ball's **start position**, the **destination** and the **maze**, determine whether the ball could stop at the destination.

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The start and destination coordinates are represented by row and column indexes.

Example 1:

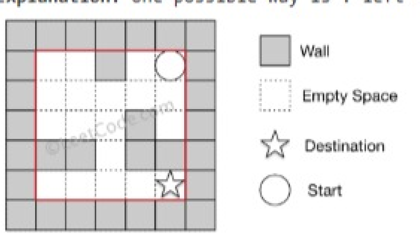
```
Input 1: a maze represented by a 2D array

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

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (4, 4)

Output: true

Explanation: One possible way is : left -> down -> left -> down -> right -> down -> ri
```



Example 2:

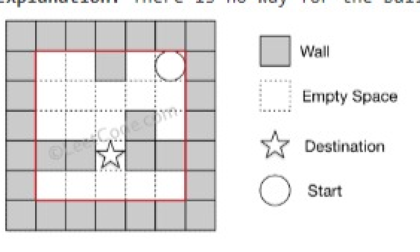
```
Input 1: a maze represented by a 2D array

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

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (3, 2)

Output: false

Explanation: There is no way for the ball to stop at the destination.
```



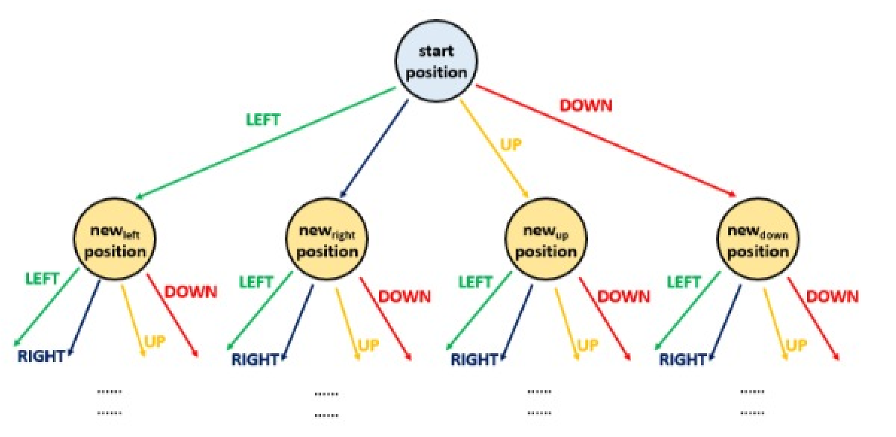
Note:

1. There is only one ball and one destination in the maze.
2. Both the ball and the destination exist on an empty space, and they will not be at the same position initially.
3. The given maze does not contain border (like the red rectangle in the example pictures), but you could assume the border of the maze are all walls.
4. The maze contains at least 2 empty spaces, and both the width and height of the maze won't exceed 100.

Solution

Approach #1 Depth First Search [Time Limit Exceeded]

We can view the given search space in the form of a tree. The root node of the tree represents the starting position. Four different routes are possible from each position i.e. left, right, up or down. These four options can be represented by 4 branches of each node in the given tree. Thus, the new node reached from the root traversing over the branch represents the new position occupied by the ball after choosing the corresponding direction of travel.

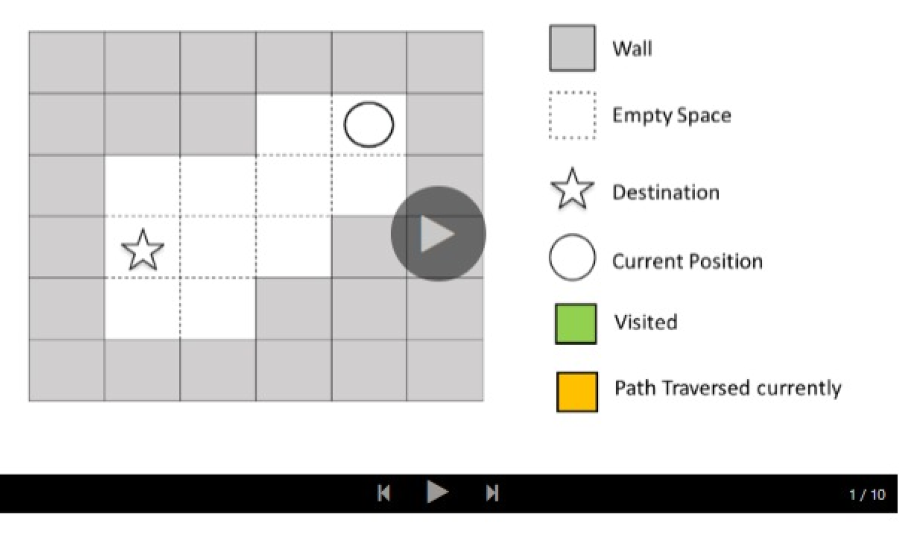


In order to do this traversal, one of the simplest schemes is to undergo depth first search. In this case, we choose one path at a time and try to go as deep as possible into the levels of the tree before going for the next path. In order to implement this, we make use of a recursive function `dfs(maze, start, desination, visited)`. This function takes the given $maze$ array, the $start$ position and the $destination$ position as its arguments along with a $visited$ array. $visited$ array is a 2-D boolean array of the same size as that of $maze$. A True value at $visited[i][j]$ represents that the current position has already been reached earlier during the path traversal. We make use of this array so as to keep track of the same paths being repeated over and over. We mark a True at the current position in the $visited$ array once we reach that particular positon in the $maze$.

From every $start$ position, we can move continuously in either left, right, upward or downward direction till we reach the boundary or a wall. Thus, from the $start$ position, we determine all the end points which can be reached by choosing the four directions. For each of the cases, the new endpoint will now act as the new start point for the traversals. The destination, obviously remains unchanged. Thus, now we call the same function four times for the four directions, each time with a new start point obtained previously.

If any of the function call returns a True value, it means we can reach the desination.

The following animation depicts the process:



Java Copy

```java
public class Solution {
    public boolean hasPath(int[][] maze, int[] start, int[] destination) {
        boolean[][] visited = new boolean[maze.length][maze[0].length];
        return dfs(maze, start, destination, visited);
    }
    public boolean dfs(int[][] maze, int[] start, int[] destination, boolean[][] visited) {
        if (visited[start[0]][start[1]])
            return false;
        if (start[0] == destination[0] && start[1] == destination[1])
            return true;
        visited[start[0]][start[1]] = true;
        int r = start[1] + 1, l = start[1] - 1, u = start[0] - 1, d = start[0] + 1;
        while (r < maze[0].length && maze[start[0]][r] == 0) // right
            r++;
        if (dfs(maze, new int[] {start[0], r - 1}, destination, visited))
            return true;
        while (l >= 0 && maze[start[0]][l] == 0) //left
            l--;
        if (dfs(maze, new int[] {start[0], l + 1}, destination, visited))
            return true;
        while (u >= 0 && maze[u][start[1]] == 0) //up
            u--;
        if (dfs(maze, new int[] {u + 1, start[1]}, destination, visited))
            return true;
        while (d < maze.length && maze[d][start[1]] == 0) //down
            d++;
        if (dfs(maze, new int[] {d - 1, start[1]}, destination, visited))
            return true;
        return false;
```

Complexity Analysis

- Time complexity : $O(mn)$. Complete traversal of maze will be done in the worst case. Here, m and n refers to the number of rows and coloumns of the maze.
- Space complexity : $O(mn)$. $visited$ array of size $m * n$ is used.

Approach #2 Breadth First Search [Accepted]

Algorithm

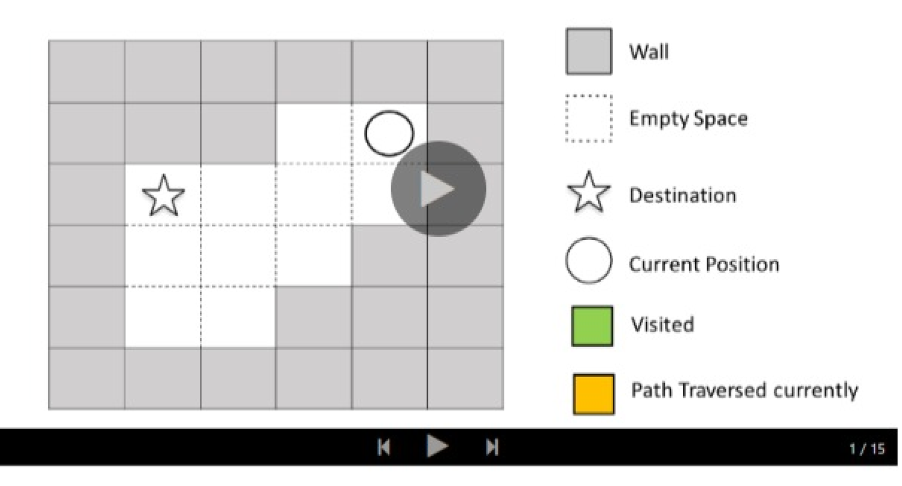
The same search space tree can also be explored in a Depth First Search manner. In this case, we try to explore the search space on a level by level basis. i.e. We try to move in all the directions at every step. When all the directions have been explored and we still don't reach the destination, then only we proceed to the new set of traversals from the new positions obtained.

In order to implement this, we make use of a $queue$. We start with the ball at the $start$ position. For every current position, we add all the new positions possible by traversing in all the four directions(till reaching the wall or boundary) into the $queue$ to act as the new start positions and mark these positions as True in the $visited$ array. When all the directions have been covered up, we remove a position value, s, from the front of the $queue$ and again continue the same process with s acting as the new $start$ position.

Further, in order to choose the direction of travel, we make use of a dir array, which contains 4 entries. Each entry represents a one-dimensional direction of travel. To travel in a particular direction, we keep on adding the particular entry of the $dirs$ array till we hit a wall or a boundary. For a particular start position, we do this process of dir addition for all all the four directions possible.

If we hit the destination position at any moment, we return a True directly indicating that the $destination$ position can be reached starting from the $start$ position.

The following animation depicts the process:



Java Copy

```java
public class Solution {
    public boolean hasPath(int[][] maze, int[] start, int[] destination) {
        boolean[][] visited = new boolean[maze.length][maze[0].length];
        int[][] dirs={{0, 1}, {0, -1}, {-1, 0}, {1, 0}};
        Queue < int[] > queue = new LinkedList < > ();
        queue.add(start);
        visited[start[0]][start[1]] = true;
        while (!queue.isEmpty()) {
            int[] s = queue.remove();
            if (s[0] == destination[0] && s[1] == destination[1])
                return true;
            for (int[] dir: dirs) {
                int x = s[0] + dir[0];
                int y = s[1] + dir[1];
                while (x >= 0 && y >= 0 && x < maze.length && y < maze[0].length && maze[x][y] == 0) {
                    x += dir[0];
                    y += dir[1];
                }
                if (!visited[x - dir[0]][y - dir[1]]) {
                    queue.add(new int[] {x - dir[0], y - dir[1]});
                    visited[x - dir[0]][y - dir[1]] = true;
                }
            }
        }
        return false;
    }
}
```

Complexity Analysis

- Time complexity : $O(mn)$. Complete traversal of maze will be done in the worst case. Here, m and n refers to the number of rows and coloumns of the maze.
- Space complexity : $O(mn)$. $visited$ array of size $m * n$ is used and $queue$ size can grow upto $m * n$ in worst case.

Analysis written by: @vinod23

Rate this article:

Previous Next

Comments: 27 Sort By

Type comment here... (Markdown is supported)

Preview Post

edaengineer 255 June 8, 2018 9:56 PM
@vinod23
why does DFS TLE but not BFS when time complexity is same???
56 Share Reply
SHOW 5 REPLIES

earthaa 22 December 28, 2018 10:01 AM
Actually, DFS solution will not cause TLE error. You can copy the code and it will pass
21 Share Reply
SHOW 2 REPLIES

leduykhanh 34 September 1, 2018 1:22 PM
Very concise DFS solution :

```
class Solution {
    private int[][] dir = new int[][]{{-1,0},{1,0},{0,-1},{0,1}};
```

Read More
12 Share Reply

Han_V 159 March 17, 2018 1:12 AM
I've implemented BFS as my first idea yet still I don't understand why tests make DFS fail due to TL. BFS in this case is more efficient yet they have same worst case time complexity so to be honest I don't necessarily agree with this decision.
4 Share Reply
SHOW 1 REPLY

xguo-Will 51 September 5, 2017 3:31 AM
@vinod23
My dfs solution just got accepted.
Share my code and comments:

```
// dfs solution
```

Read More
4 Share Reply

wds8807 121 May 23, 2019 6:57 AM
Regardless of the running time, the DFS code is so messy.
2 Share Reply

moonstone1011 149 November 18, 2018 4:55 AM
first line of bfs has typo? "The same search space tree can also be explored in a Depth First Search manner"
2 Share Reply

utsav_popli 10 July 2, 2019 8:40 AM Report
Seems like the use case:
[[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]]
[0,4]
[1,2]
Read More
2 Share Reply
SHOW 4 REPLIES

ajay13 120 December 21, 2018 11:35 AM
@vinod23 What software do you use to create these visualization of data ? Its pretty helpful !!
1 Share Reply
SHOW 1 REPLY

s961206 623 January 11, 2020 1:57 AM
Why the running time of DFS is O(M*N)?
0 Share Reply