Programming homework for searching algorithms

The problem is to provide a solution path to the maze with Depth First Search algorithm.

As specified in assignments,

- Copy this notebook from the tab "File > Save a copy in Drive"
- Open the notebook that copied to your drive
- Go the drive folder that shared with the link
- Add the shared folder to your drive to access it from the notebook.

The problem specifications:

- Initial position for all the multiple mazes that read is the point "**(0,0)**"
- Goal is to reach the max point "**(N-1, M-1)**", where N and M corresponds to the height and width of the maze.
- The multiple mazes will be read from the file that is provided in the drive file shared with you
- The maze consists of **0**s and **1**s which **0s indicate a clear path** and **1s indicate a wall** that can not be moved
- To reach to the goal, you are required to provide a path consist of clear roads(0s)
- The reading and converting the path to the desired outputs have already been implemented which you **CANNOT** change in order to get full credits
- The exact outputs that your function expected to provide are printed it in the last code block given the expected output file.
- You need the provide the required function(s) that finds the path from initial position to the goal position using **Depth First Search**, which you may implement it with stack or recursively as you wish.

The submission:

- Run all the code blocks after you finished your homework
- Download and submit the .ipynb file from the tab "File > Download .ipynb"

For example, the solve_dfs function will take maze parameter as:

```
maze = [[0,0,0,0,0,0],
        [0,1,0,0,0,0],
        [0,0,1,1,1,0],
        [0,0,0,0,0,0],
        [1,0,0,0,1,0]]
```

The returned path should be:

```
path => [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2, 5), (3, 5), (4, 5)]
```

where (x, y) is tuples.

The directions extracted from this path is:

```
direction => R R R R R D D D D
```

Mount your drive to access the files

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318

Enter your authorization code:
··········
Mounted at /content/drive
```

Modules that needed (You won't need any other module to implement the search algorithm)

```
import collections
import numpy as np
```

Functions that already implemented

```
def read_mazes(input_file):
    mazes = []

    with open(input_file, 'r') as maze_file:
        maze = []

        for line in maze_file:
            if line != '\n':
                maze.append(line.replace('\n','').split(','))
            else:
                mazes.append(np.array(maze, dtype=int))
                maze = []

```

```
        if len(maze) > 0:
            mazes.append(np.array(maze, dtype=int))

    return mazes
```

```
def get_directions(path):
    directions = ""

    current_cell = path[0]

    for cell in path[1:]:
        if current_cell[0] == cell[0]:
            if cell[1] - current_cell[1] > 0:
                directions += "R "
            else:
                directions += "L "
        else:
            if cell[0] - current_cell[0] > 0:
                directions += "D "
            else:
                directions += "U "
        current_cell = cell

    return directions.strip()
```

Depth First Search algorithm which you will implement

- **The function takes 2d numpy array maze as a single parameter**
- **Returns a list of points that starts from (0,0) an ends with (N-1,M-1)**
- **The direction priorities are as follows Right-Down-Left-Up**
- **Returns None if goal can not be reached from the initial position**
- Read the initial instructions if not clear

Also, you can implement multiple functions as you like or just use this function.

HOWEVER, the **solve_dfs** function name **MUST** remain same and **MUST** take a **single parameter maze**

So, any other functions that you would fine it useful should be called from inside the solve_dfs function

```
def solve_dfs(maze):
    start = (0, 0)
    goal = (maze.shape[0]-1, maze.shape[1]-1)
    width = maze.shape[0]
    height = maze.shape[1]
    path=[]
    maze[0][0]=1
```

```
    path.append(start)
    solve_dfs_2(start,width,height,path,maze,goal)
    if path==[]:
      return None
    else:
      return path

def solve_dfs_2(node,width,height,path,maze,goal):
  if node == goal:
    return True
  elif node_valid((node[0],node[1]+1),width,height,maze)==True:
    node=(node[0],node[1]+1)
    maze[node[0]][node[1]]=1
    path.append(node)
    solve_dfs_2(node,width,height,path,maze,goal)
  elif node_valid((node[0]+1,node[1]),width,height,maze)==True:
    node=(node[0]+1,node[1])
    maze[node[0]][node[1]]=1
    path.append(node)
    solve_dfs_2(node,width,height,path,maze,goal)
  elif node_valid((node[0],node[1]-1),width,height,maze)==True:
    node=(node[0],node[1]-1)
    maze[node[0]][node[1]]=1
    path.append(node)
    solve_dfs_2(node,width,height,path,maze,goal)
  elif node_valid((node[0]-1,node[1]),width,height,maze)==True:
     node=(node[0]-1,node[1])
     maze[node[0]][node[1]]=1
     path.append(node)
     solve_dfs_2(node,width,height,path,maze,goal)
  elif path==[]:
    path=[]
  else:
    path.pop()
    for i in range(0, len(path)):
         if i == (len(path)-1):
           node=path[i]


    solve_dfs_2(node,width,height,path,maze,goal)


def node_valid(node,width,height,maze):

  if (node[0]<0) or (node[0] >= width) or (node[1]<0) or (node[1]>=height) or maze[node[0]
    return False
  else:
    return True
```

```
    ### You may define this solve_dfs function in any way you want or just use this functi
```

Main code block that reads the mazes, run the search algorithm and returns the path and prints the directions that reach to the goal

```
mazes = read_mazes('/content/drive/My Drive/CS404_DFS_HW/input.txt')

for maze, ind in zip(mazes, range(1, len(mazes)+1)):
    path = solve_dfs(maze)
    print(path)

    if path != None:
        directions = get_directions(path)
        print(str(ind) + ") " + directions + '\n')
    else:
        print(str(ind) + ') Could not find a path...\n')
```

```
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (1, 9)
1) R R R R R R R R R D D L D D R

[(0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4), (4, 4)]
2) D D R D R R R D

[(0, 0), (0, 1), (0, 2), (1, 2), (1, 1), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2)
3) R R D L D L D D R R U R U R R D D D R R U U U U U R R D D D D D D

[(0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4), (4, 4)]
4) D D R D R R R D

None
5) Could not find a path...

[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2, 5), (3, 5), (4, 5)]
6) R R R R R D D D D

[(0, 0), (0, 1), (0, 2), (1, 2), (1, 1), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2)
7) R R D L D L D D R R R U U R R D R R R D D

[(0, 0), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (0, 4), (0, 5), (0, 6), (0, 7), (1, 7)
8) D R R R R U R R R D R R R R D R R D R R R R R R U R U U R R R R R R D D D R R D D L

[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 6), (1, 7), (1, 8), (1, 9)
9) R R R R R R D R R R R R D R D R R R R R R D R D L D L D D L L U R U L L L D D D L L

[(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (6, 3), (7, 3)
10) D R D D D D D R R D D L L D D R D R R R D L D D R R R R R R D L D D R R R R D L L D
```

Expected output that your algorithm should print

```
with open('/content/drive/My Drive/CS404_DFS_HW/expected_output.txt') as expected_output:
    for expected_direction, ind in zip(expected_output, range(1, len(mazes)+1)):
        print(str(ind) + ') ' + expected_direction)
```

```
1) R R R R R R R R D D L D D R

2) D D R D R R R D

3) R R D L D L D D R R U R U R R D D D R R U U U U U U R R D D D D D D

4) D D R D R R R D

5) Could not find a path...

6) R R R R R D D D D

7) R R D L D L D D R R R U U R R D R R R D D

8) D R R R R U R R R D R R R R D R R D R R R R R R U R U U R R R R R R D D D R R D D L

9) R R R R R R D R R R R R D R D R R R R R R R D R D L D L D D L L U R U L L L D D D L L

10) D R D D D D D R R D D L L D D R D R R R D L D D R R R R R R D L D D R R R R R D L L D
```

```
1
```