CS 520 - Exam Q1: Finding Your Way

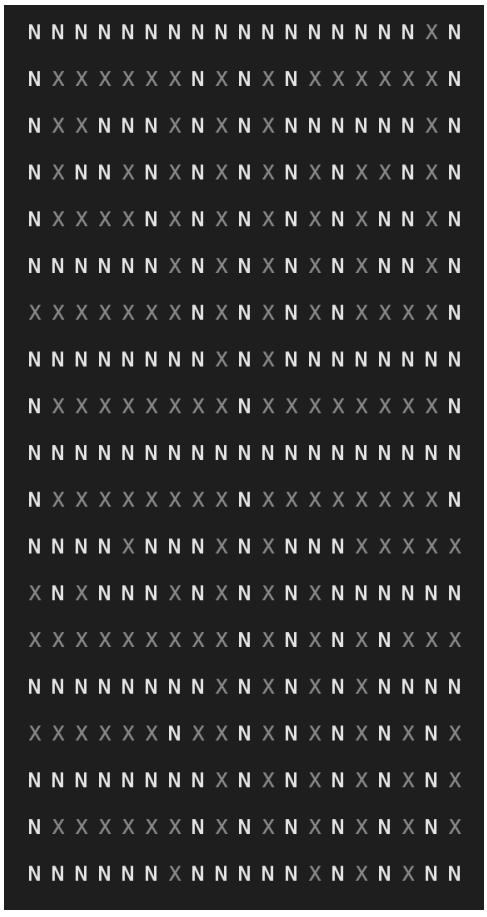
Submitted by Aditya Manelkar (am3369)

Design and Algorithms

Question 1

Explanation

The probability that the drone is in the top left corner (or any other) cell at the start is equivalent to the probability of picking the top left cell out of all the white (unblocked) cells. This value comes to 1/199 or a little over 0.005:



Where X's represent walls, and N's represent those cells whose initial probabilities are 0.005.

Important Codes

find2.py: Has the logic for solving question 1 starting from line 40. The probability values for the drone being each cell are held in the structure disp_maze. This structure is then used by the display() function to display the initial probability spread in the maze using colorama for visualization.
read_map.py: Has the read() function that is called in find2,py and is used to read the reactor map in the .txt format and store it in a 2D list in python.

Question 2

Explanation

Running the “DOWN” command we can imagine all the probability values transitioning downwards. This makes it such that the cells that have no possible incoming transitions and only outgoing transitions (like the top-left cell) have their probabilities go down to zero, and on the other end of the spectrum, those cells that have no possible outgoing transitions and just incoming transitions (like the cell at the bottom-left corner) have their probabilities increase (and actually double for this specific action). Most other normal cells - where there exist incoming and outgoing transitions - have virtually no changes to their probabilities.

An example of what the code does when it wants to transition all the probabilities one step “DOWN”:

```
# Down
for row in range(num_rows - 1, -1, -1):
    for col in range(num_cols):
        if row + 1 < num_rows and disp_maze[row][col] > 0 and disp_maze[row][col] != -1 and disp_maze[row + 1][col] > -1:
            disp_maze[row + 1][col] += disp_maze[row][col]
            disp_maze[row][col] -= disp_maze[row][col]
```

Finally, we see the following probability distribution in the maze:

The cells represented by red L's are those that have their probabilities go down to zero. Those represented by N's have virtually unchanged probabilities. And those given by H's in yellow have their probabilities increased.

Important Codes

find2.py: The code starting line 62 has the change to transition all the probabilities in the maze one step "DOWN". The function display() displays different values (N/H/L) based on the probabilities of the drone being in the cells. The action itself is down in a crude way, but is better defined later in the code for questions 3 and 4.

Question 3

Design

For the question at hand, the thought process was that I had to find a way to maximize the probability of any one cell to 1, and that would mean I would know where the submarine drone is. To achieve this, I split the problem into two parts:

March

This approach is taken at the start of the problem to make most of the probabilities completely vanish. For this the code performs a complete smashing of probabilities in all four directions either in clockwise or counterclockwise direction. What does this mean? It means that in each direction we take the full number of steps possible to make sure that all the probabilities are concentrated on one edge along that direction. This does result in quite a few wasteful moves - so the approach is not the optimal one. But the main idea behind doing this is we get far fewer cells with non-zero beliefs to contend with which then makes life easier computationally while trying to converge to a single cell with highest probability.

Typically after smashing probabilities in all directions once - we are typically left with just 10 or so cells with non-zero probabilities to deal with, which proves to be of great benefit time wise later!

Hunt

Once we have 10-15 cells with non-zero probabilities (for 19 x 19 mazes), what we do is initialize Drone objects in the code on each of the cells (they are not actual drones, I just ended up naming the class as such) and then using a reference Drone object, we try to catch every single one of the remaining drones using shortest path algorithms as heuristics to inform which direction to move in to close in on the Drone object the reference Drone should catch. Here though, unlike the March phase, we transition the probabilities in the maze one move at a time.

The combination of the two approaches have the following pitfalls and benefits:

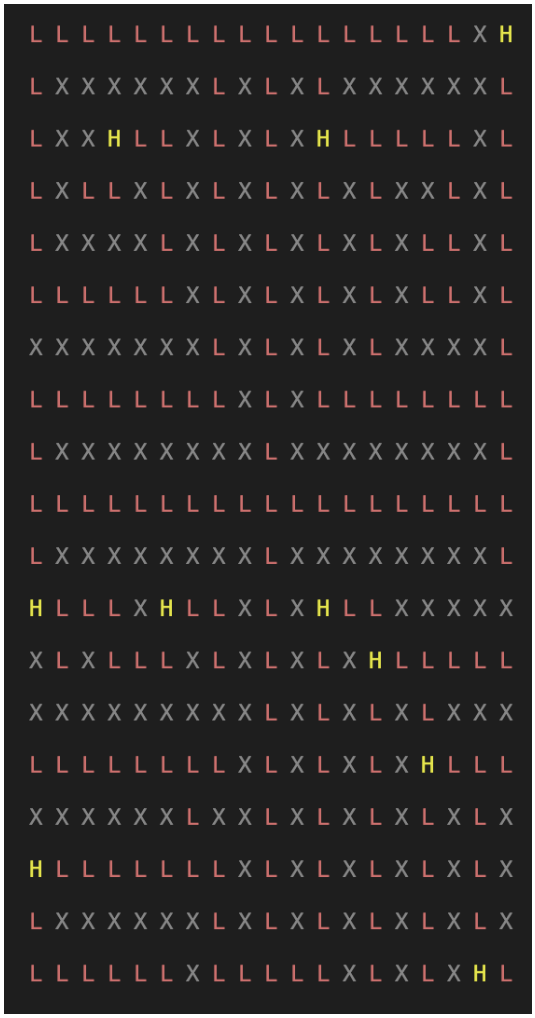
Pitfalls

1. The March phase makes this approach a good one, but not an optimal one.
2. If the number of moves needs to be minimized, this approach does well but not optimally.

Benefits

1. The approach is very consistent, in that it has always been able to solve the reactor maze provided in the write-up in a decent 242 steps (Dr. Cowan had mentioned around 300 to be a decent number).
2. The approach completely avoids any heavy lifting from a processing point of view initially, and hence saves on a lot of time! Even with 9 mazes of size 19 x 19 to process, the code breezes past all of them in just 0.295 seconds! So if speed is what's important than the number of moves, this approach will flourish.

Following is the convergence of the probabilities after the March phase:



The code converges to the points [(0, 18), (2, 3), (2, 11), (11, 0), (11, 5), (11, 11), (12, 13), (14, 15), (16, 0), (18, 17)] in 76 moves.

The code then goes for full convergence through the Hunt phase:

So the code converges to the point (18, 11) in a total (March + Hunt) of **242 steps.**

The actions the code took to achieve this state are:

```
The actions the code took to achieve it are:
['DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN',
'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN',
'DOWN', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT',
'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT',
'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP',
'UP', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP', 'UP',
'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT',
'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT',
'LEFT', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'LEFT',
'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'UP', 'UP', 'UP', 'UP', 'UP',
'UP', 'UP', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT',
'LEFT', 'LEFT', 'LEFT', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'RIGHT',
```

```
'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'UP', 'UP', 'UP', 'LEFT', 'LEFT', 'RIGHT',
'RIGHT', 'DOWN', 'DOWN', 'DOWN', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'UP',
'UP', 'UP', 'UP', 'UP', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT',
'RIGHT', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'LEFT',
'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'DOWN', 'DOWN', 'DOWN',
'DOWN', 'RIGHT', 'RIGHT', 'RIGHT', 'DOWN', 'RIGHT', 'RIGHT', 'LEFT', 'LEFT',
'UP', 'LEFT', 'LEFT', 'LEFT', 'UP', 'UP', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT',
'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'RIGHT', 'DOWN', 'DOWN', 'DOWN', 'DOWN',
'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'RIGHT', 'RIGHT', 'UP', 'UP', 'UP',
'UP', 'UP', 'UP', 'UP', 'RIGHT', 'RIGHT', 'DOWN', 'RIGHT', 'RIGHT', 'RIGHT',
'RIGHT', 'RIGHT', 'UP', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'LEFT', 'UP', 'UP',
'LEFT', 'LEFT', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN',
'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN',
'DOWN', 'DOWN', 'RIGHT', 'RIGHT']
```

The code between lines 86 and 392 of find2.py have the full workflow for this question.

Important Codes

find2.py: The flow for question 3 works in two parts here, where we run the March phase, we repeat the moves until we can close in one just a small fraction of cells with non-zero probabilities. Once we do find a small enough number, we just run the Hunt phase to concentrate all the probabilities to a single point. For this process we use the Drone objects from drone.py.
drone.py: The code has the Drone class which is used to simulate drone movements and also calculate and store heuristics. In the find2.py script, objects of this Drone class are used to keep track of potential cells with drone presence post a transition.

Question 4

Description

The approach I have taken is to create blocked and unblocked cells in a manner such that I am increasing the percentage of blocked cells by 5% each time. To make sure that the maze created is perfect, I am using an explore function that checks for connectedness of all unblocked cells of the maze.

Important Codes

maze.py: The main driver code for maze generation. The code has the create_walls function to create the blocked cells and the explore function to check for maze perfectness. Finally the save_txt function exports the maze created as a text file.
find2.py: Starting line 393, the code has a flow similar to question 3 but repeated over all the mazes generated by maze.py. Upon comparing the number of moves it takes the March and Hunt approach to find the drone in each of the mazes, the code outputs which of the mazes does best to challenge our approach.

Following is a screenshot of the output:

```
The number of moves the code took to find the position (11, 18) for maze with 5 percent walls are: 128
The number of moves the code took to find the position (18, 0) for maze with 10 percent walls are: 158
The number of moves the code took to find the position (18, 16) for maze with 15 percent walls are: 250
The number of moves the code took to find the position (14, 18) for maze with 20 percent walls are: 180
The number of moves the code took to find the position (18, 18) for maze with 25 percent walls are: 163
The number of moves the code took to find the position (17, 0) for maze with 30 percent walls are: 340
The number of moves the code took to find the position (10, 18) for maze with 35 percent walls are: 287
The number of moves the code took to find the position (18, 18) for maze with 40 percent walls are: 340
reactor30.txt is the maze that challenges the algorithm the most with 340 moves required to solve!
```

Note

For the code, there is a readme in the zipped folder with instructions on setting up the environment and site packages necessary for the execution of the code (based on the environment.yml file in the same folder). If there are any issues please let me know.

In the worst case, numpy and colorama are two modules that could be installed using conda.

References

Codes from projects 1 and 2