CS 520 - Final Exam Q1: Finding Your Way

Brief:

In this scenario, we have a maze-like grid, where we must determine the position of a drone with 100% confidence. We can issue directional commands to this drone.

Hence the objective can be better put as:

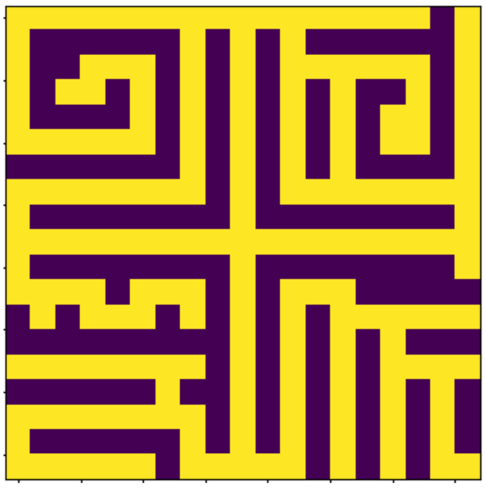
1) Issue probability of drone being present in a cell (which is open) as (1/number of open cells). In the case of my code, for better readability, I am assigning 1 and checking for the number-of-open-cells value to stop my search.
2) Issue a minimal number of commands such that all these probabilities add up and converge to 1 (or, in my case, number-of-open-cells), so we can locate the drone with 100% probability.

> **Question 1 (5 Points):** Before you do anything, what is the probability that the drone is in the top left corner? Why?

Before doing anything, the probability of the drone being in any open cell is 1/total number of open cells, and the probability of the drone is in a blocked cell is 0.
Hence if the top-left corner is open, the probability is 1/open cells, else probability is 0. In case of TH23-SA74-VERW, it is 1/199.

Consider issuing the command 'DOWN'. While you don't know exactly where the drone is, you can say where it *isn't* - it isn't, for instance, in the top left corner anymore.

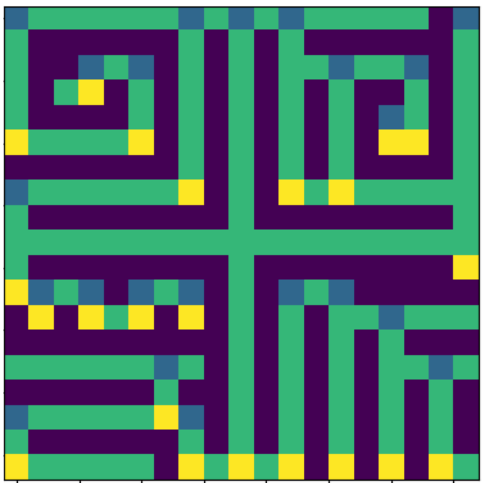
> **Question 2 (5 Points):** What are the locations where the drone is most likely to be? Least likely to be? How likely is it to be in all the other locations? Indicate your results visually.



After a particular sequence of movements, the drone is likelier to be near a wall or stuck in a cell bounded by 3 walls. It is less likely to be in an open space.

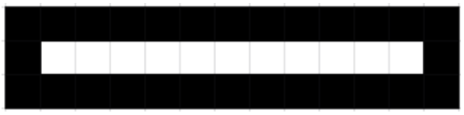
After making one down move from the initial state, here are the cases in which the probability state of the drone can change.

1) If there is no block in the 2nd row in any column, the drone is free to move downwards. Hence the probability of the drone is in the first row is 0 for columns where it is not bounded in 2nd row.
2) If there is a block in the 2nd row, the drone is blocked from moving downwards in that column; in that case, the probability of the drone being in the first column remains the same.
3) As all the probabilities are propagated downward, if there is a block below blocking the drone, the probabilities get summed, resulting in higher probabilities above the block.
4) The same goes for the last row, where all the columns are blocked, and the probability of the drone being in the last row is added.



After DOWN move:

Blue: 0
Yellow: 2
Green: 1
Purple: blocked



In the case of this maze, the optimal number of moves is to make 9 left or 9 right moves.

Question 3 (25 Points): Write a program that takes a reactor schematic as a text file (see associated file for this reactor) and finds a sequence of commands that, at the end of which, you know *exactly* what cell the drone is located in. Be clear in your writeup about how you are formulating the problem, and the algorithms you are using to find this sequence.

What is the sequence for this reactor?

I had initially approached this problem in numerous ways detailed in the file submitted "prevapproaches.py" with many versions of heuristics and/or the optimal number of moves values computed.
A brief on previous approaches:

- Brute force BFS:
  Search algorithm guaranteed to return the minimum sequence by checking for every permutation. Very inefficient computationally.

- Moving in a counterclockwise spiral (L, D, R, U) with the number of moves decided as:
  The maximum distance across the belief matrix from first non-zero belief to a neighboring wall in the corresponding direction.

Partially converging to 8 discrete probabilities spread across the maze in 45 steps. One possible solution was to plug in BFS here to take over to the end, but I found no guarantee that it was optimal.

- A * search approach with above-computed heuristic values

Again, I was unsure about how good the heuristic was and how accurately it represented the current state, so I had to rework the entire thing.

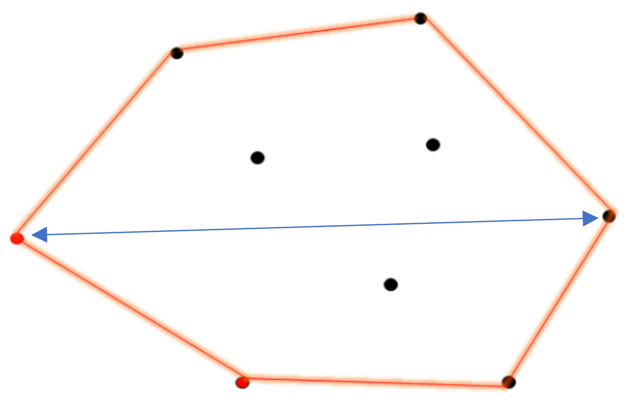
The approach that performed best both in the case of a minimal sequence of moves and computationally optimal was the Best First search, especially since the emphasis was on finding minimal moves.

I chose Best First Search because I knew that breadth-first search would give me an optimal sequence, and I had a heuristic to speed up the above process.

Here, there is a tradeoff between optimal sequence and computational performance. While breadth-first search is guaranteed to give me the best sequence, it is a polynomial time algorithm. To reorder my search space, I prioritize my moves using a priority queue and heuristic. This means I could miss out an optimal answer by ignoring moves at the bottom of the queue. Hence, I am responsible for developing a heuristic that reflects a move's utility as perfectly as possible to ensure an optimal solution while ensuring I do not burn up my poor Intel chip.

My heuristic for Best first search:

- I get all the indices for a given belief state where my belief values are non-zero.
- I am trying to construct a fence/boundary bounding all these points such that the length of the fence is minimal. This gives me a way to quantify convergence and minimizing it should be a goal.
- In addition, I am also using BFS to find the longest distance between any two points for the points lying over the boundary computed above.
- To minimize it, I also included the number of non-zero states in the heuristic.
- A combination of these values (which I envisioned as circumference and diameter) gives me an estimation of the spread of non-zero belief values and how I can converge them.

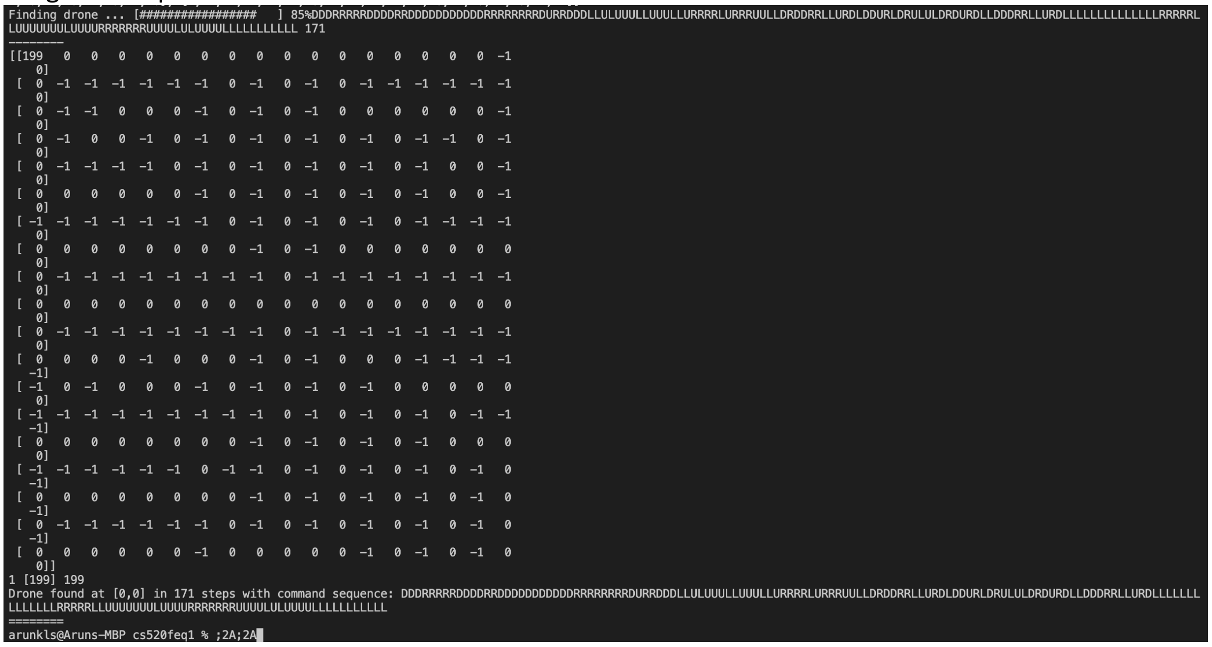


Since we are using a min heap to minimize the heuristic and have the length of sequence as a factor, we are bound to arrive at the shortest sequence first assuming the other heuristics convey the state of convergence well.

My sequence output for the grid given in the question paper is:

DDDRRRRRDDDDRRDDDDDDDDDDDDRRRRRRRRDURRDDDLLULUUULLUUULLURRRRLURRRUULLDRDD
RRLLURDLDDURLDRULULDRDURDLLDDDRRLLURDLLLLLLLLLLLLLLRRRRRLLUUUUUUULUUUURRRRRRR
UUUULULUUUULLLLLLLLLL

Length of sequence: 171



```
Finding drone ... [#################   ] 85%DDDRRRRRDDDDRRDDDDDDDDDDDDRRRRRRRRDURRDDDLLULUUULLUUULLURRRRLURRRUULLDRDDRRLLURDLDDURLDRULULDRDURDLLDDDRRLLURDLLLLLLLLLLLLLLRRRRRL
LUUUUUUULUUUURRRRRRRUUUULULUUUULLLLLLLLLL 171
--------
[[199   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  -1
    0]
 [  0  -1  -1  -1  -1  -1  -1   0  -1   0  -1   0  -1  -1  -1  -1  -1  -1
    0]
 [  0  -1  -1   0   0   0  -1   0  -1   0  -1   0   0   0   0   0   0  -1
    0]
 [  0  -1   0   0  -1   0  -1   0  -1   0  -1   0  -1   0  -1  -1   0  -1
    0]
 [  0  -1  -1  -1  -1   0  -1   0  -1   0  -1   0  -1   0  -1   0   0  -1
    0]
 [  0   0   0   0   0   0  -1   0  -1   0  -1   0  -1   0  -1   0   0  -1
    0]
 [ -1  -1  -1  -1  -1  -1  -1   0  -1   0  -1   0  -1   0  -1  -1  -1  -1
    0]
 [  0   0   0   0   0   0   0   0  -1   0  -1   0   0   0   0   0   0   0
    0]
 [  0  -1  -1  -1  -1  -1  -1  -1  -1   0  -1  -1  -1  -1  -1  -1  -1  -1
    0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0]
 [  0  -1  -1  -1  -1  -1  -1  -1  -1   0  -1  -1  -1  -1  -1  -1  -1  -1
    0]
 [  0   0   0   0  -1   0   0   0  -1   0  -1   0   0   0  -1  -1  -1  -1
   -1]
 [ -1   0  -1   0   0   0  -1   0  -1   0  -1   0  -1   0   0   0   0   0
    0]
 [ -1  -1  -1  -1  -1  -1  -1  -1  -1   0  -1   0  -1   0  -1   0  -1  -1
   -1]
 [  0   0   0   0   0   0   0   0  -1   0  -1   0  -1   0  -1   0   0   0
    0]
 [ -1  -1  -1  -1  -1  -1   0  -1  -1   0  -1   0  -1   0  -1   0  -1   0
   -1]
 [  0   0   0   0   0   0   0   0  -1   0  -1   0  -1   0  -1   0  -1   0
   -1]
 [  0  -1  -1  -1  -1  -1  -1   0  -1   0  -1   0  -1   0  -1   0  -1   0
   -1]
 [  0   0   0   0   0   0  -1   0   0   0   0   0  -1   0  -1   0  -1   0
    0]]
1 [199] 199
Drone found at [0,0] in 171 steps with command sequence: DDDRRRRRDDDDRRDDDDDDDDDDDDRRRRRRRRDURRDDDLLULUUULLUUULLURRRRLURRRUULLDRDDRRLLURDLDDURLDRULULDRDURDLLDDDRRLLURDLLLLLLL
LLLLLLLRRRRRLLUUUUUUULUUUURRRRRRRUUUULULUUUULLLLLLLLLL
========
arunkls@Aruns-MBP cs520feq1 % ;2A;2A
```

This algorithm takes 10-15 seconds to converge at a cell for an input 19x19 matrix.

I referred to this problem while arriving at my heuristic: https://leetcode.com/problems/erect-the-fence/

Steps to run code:
Create an object of class DroneFinder present in dronefinder.py
Run converge method.

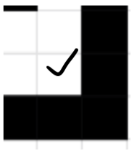
df = DroneFinder("Thor23-SA74-VERW-Schematic (Classified).txt")
seq, belief = df.converge(df.belief)

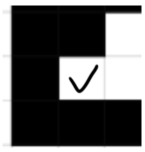
seq variable has the optimal sequence.

Or, run dronefinder.py after editing in the input file name in line 278

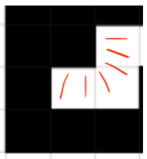
Question 4 (15 Points): Write a program that outputs a a 19x19 reactor schematic that has the longest possible sequence of commands needed to locate the drone that you can find. Be clear in your writeup about how you are formulating the problem, and the algorithms you are using to find this reactor schematic.

The largest possible sequences occur in a grid when minimal probabilities converge at every move. Probabilities can converge if they are caught with no way to move while neighbouring probabilities move in to add up.
In order to do this, we can edit an existing maze, and try to minimize the number of open spaces bounded by 2 or 3 blocks.


cell bounded by 2 blocks


cell bounded by 3 blocks

Take an existing dense graph with single connected component.
Iterate through the graph to keep track of such cells.
Prioritize the removal of cells bounded by 3 blocks by making blocked cells such as


, unbounded.

I tried an approach to above algorithm in the file largestSequence.py where I am compiling a list of open cells bounded by 3 walls.
I am popping one cell and making one of it's walls empty and checking the sequence length for locating drone for the new maze.
Largest maze sequence across the above operation should be the maze for which sequence length is largest.

Steps to run code:
run largestSequence.py after editing in the input file name in line 96