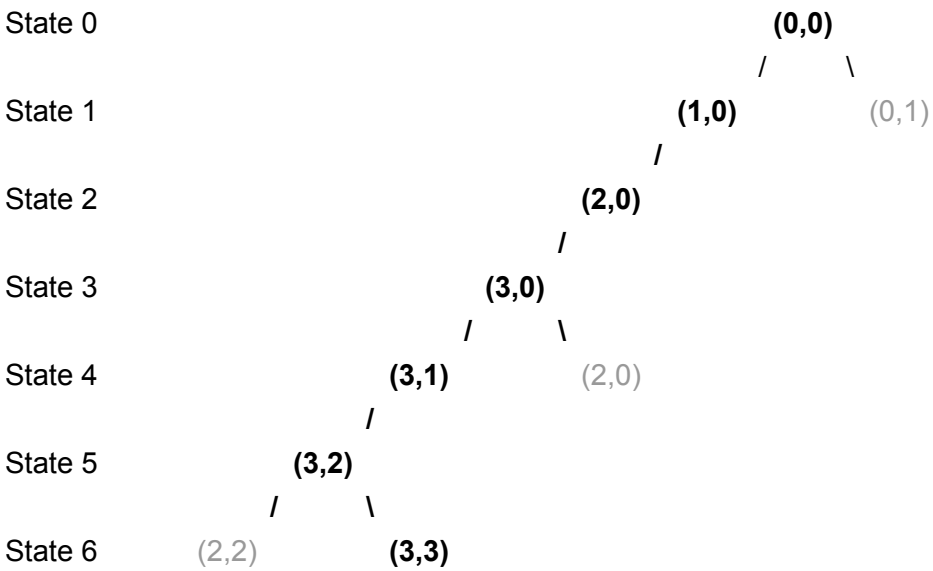
Calvin Chen
EN.605.645.81.SP19
Self Check 1

1. A state in the problem is the raw data of the microworld, including the current location of the robot and the open and closed positions. It may also include the previously visited indices of the microworld. The set of states is all possible states of the board, or all the positions that the robot can be in. There are 13 states in this set of states.
2. The set of transitions define the valid moves given the current state. That is, where the robot can move while staying on an open area. They do not need to be specified explicitly and can be called upon as a function. This function would return the successors as possible transition states. Given the starting state, the successors function would determine that DOWN (1, 0) and RIGHT (0,1) are the only valid moves.
3. My transition set is defined as a DFS exploring the microworld in the following order: RIGHT, DOWN, LEFT, UP. At each state, the robot will first try to move right, then try to move down, and so on and so forth. After the first successor function, the graph will look like this:

```
0 S 0 0
0 x x 0
0 0 0 0
0 0 x G
```

4. Using the implementation described above, the DFS will look like this:
   The frontier is denoted in gray



5. Yes, this is the optimal path assuming that the cost of each unit of movement is the same. This path takes the robot 6 states to traverse.

6. The fold is caused by the order of the successor states. In this example, one possible order is DOWN, RIGHT, UP, LEFT, taking care not to traverse previously visited indices. One way to go directly from (0, 2) directly to (1, 2) would be if the order placed RIGHT before DOWN, such as RIGHT, DOWN, LEFT, UP. However, if this were the order used throughout all states, then the robot would never access (0, 2) in a DFS.
7. To recover the path, we need to keep track of the previous actions taken. This will provide the information necessary to revert to previous states. We also need to retain the previous state of every action in our DFS stack. This can be represented by storing the previous index and the action performed within each element of the stack. Finally, we also need to know the current index is.

Additional Questions

1. Use Queue instead Stack to store successors. Should return shortest solution.
2. Depends on tiebreaker used to determine which action to perform?
3. Use Priority Queue ordered by minimum path length. Should return shortest solution.