Question 1 Implement the depth-first search (DFS) algorithm in the depthFirstSearch function in search.py. To make your algorithm *complete*, write the graph search version of DFS, which avoids expanding any already visited states.

Data structure used: Stack

(Memory usage has been taken as the number of number of nodes expanded.)

Tiny maze:
Path cost: 10
Memory Usage/Number of nodes expanded: 15
Score: 500
Running time: 0.0 seconds

Medium maze:
Path cost: 130
Memory Usage/Number of nodes expanded: 146
Score: 380
Running time: 0.0 seconds

Big maze:
Path cost: 210
Memory Usage/Number of nodes expanded: 390
Score: 300
Running time: 0.0 seconds

Question 2 Implement the breadth-first search (BFS) algorithm in the breadthFirstSearch function in search.py. Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search.

Data structure used: Queue

(Memory usage has been taken as the number of Memory Usage/Number of nodes expanded.)

Tiny maze:
Path cost: 8
Memory Usage/Number of nodes expanded: 15
Score: 502
Running time: 0.0 seconds

Medium maze:
Path cost: 68
Memory Usage/Number of nodes expanded: 269

Score: 442
Running time: 0.0 seconds

Big maze:
Path cost: 210
Memory Usage/Number of nodes expanded: 620
Score: 300
Running time: 0.0 seconds

Question 3 Implement the uniform-cost search (UCS) algorithm in the uniformCostSearch function in search.py.

Data structure used: Priority Queue

(Memory usage has been taken as the number of Memory Usage/Number of nodes expanded.)

Tiny maze:
Path cost: 8
Memory Usage/Number of nodes expanded: 15
Score: 502
Running time: 0.0 seconds

Medium maze:
Path cost: 68
Memory Usage/Number of nodes expanded: 269
Score: 442
Running time: 0.0 seconds

Big maze:
Path cost: 210
Memory Usage/Number of nodes expanded: 620
Score: 300
Running time: 0.0 seconds

Question 4 Implement A* graph search in the empty function aStarSearch in search.py.

Data structure used: Priority Queue

(Memory usage has been taken as the number of Memory Usage/Number of nodes expanded.)

Tiny maze:
Path cost: 8
Memory Usage/Number of nodes expanded: 14

Score: 502
Running time: 0.0 seconds

Medium maze:
Path cost: 68
Memory Usage/Number of nodes expanded: 221
Score: 442
Running time: 0.0 seconds

Big maze:
Path cost: 210
Memory Usage/Number of nodes expanded: 549
Score: 300
Running time: 0.0 seconds

Question 5 Implement the CornersProblem search problem in searchAgents.py.

Tiny corners:
Path cost:28
Memory Usage/Number of nodes expanded: 252
Score: 512
Running time: 0.0 seconds

Medium corners:
Path cost: 106
Memory Usage/Number of nodes expanded: 1966
Score: 434
Running time: 0.3

Big corners:
Path cost: 162
Memory Usage/Number of nodes expanded: 7949
Score: 378
Running time: 2.9 seconds

Question 6 Implement a heuristic for the CornersProblem in cornersHeuristic.
I used the euclidean distance to the farthest corner as the heuristic.
Tiny corners:
Path cost: 28
Memory Usage/Number of nodes expanded: 229
Score: 512
Running time: 0.0 seconds

Medium corners:
Path cost: 106
Memory Usage/Number of nodes expanded: 1767
Score: 434
Running time: 0.1 seconds

Big corners:
Path cost: 162
Memory Usage/Number of nodes expanded: 7123
Score: 378
Running time: 2.5 seconds

Question 7 Fill in foodHeuristic in searchAgents.py with a consistent heuristic for the FoodSearchProblem.
I used the euclidean distance to the farthest food pellet as the heuristic.
Tricky search:
Path cost: 60
Memory Usage/Number of nodes expanded: 10352
Score: 570
Running time: 8.6 seconds