William Frank
Project 1 Analysis
1/21/19

1. Depth First Search
    1. The exploration order is more or less what I would expect from depth first search, nothing crazy. Pacman does not go to every explored square, only the ones on his path to the goal.
    2. While DFS can find a least cost solution, it is not guaranteed to, and will simply give the first path it finds to the goal, even if it is circuitous.
2. Breadth First Search
    1. BFS will find a least cost solution if the cost of every action is the same. Otherwise, BFS does not factor in cost, only the number of moves.
3. Uniform Cost Search
    1. UCS uses a priority queue, util.PriorityQueue().
4. A* Search
    1. A* and UCS both find the same path, but A* explores much less of the map, specifically the tiles that would move pacman away from the goal.
5. Finding All Corners
    1. The representation I chose involved adding the visited corners to the state. The state was represented by a tuple where the first position was the x position, then the y position, then the coordinates of any visited corners, such as: (2, 6, (1, 6), (6, 6)). As a branch visited a corner that corner got added to its state, and the state was a goal if it had all four corners in it.
6. Corners Problem: Heuristic
    1. The heuristic I chose first got the Manhattan distance to the nearest corner, then the Manhattan distance to the next nearest corner, and so on for all unvisited corners.
7. Eating All Dots
    1. The heuristic I chose for the FoodSearchProblem was the maze distance to the furthest remaining food, found by calculating the maze distance to every remaining food then returning the largest.
8. Suboptimal Search
    1. The ClosestDotSearchAgent won't always find the shortest path because it only looks for the closest single food from a given point, not the most efficient path for getting all the food.
9. Self Analysis
    1. For me, the hardest part was coming to the realization that the corners problem required me to somehow encode the visited corners into pacman's state.
    2. The easiest part was probably solving the suboptimal search problem, it only required 3 lines of code.
    3. The two heuristic problems probably helped me understand the course material the most. Before this I had a tenuous grasp on what exactly made a good heuristic, as well as admissibility and consistency.
    4. None of these problems felt particularly tedious, even implementing the search algorithms was a good refresher.
    5. No other feedback.