Autonomous Systems
Sayan Goswami
Iñaki Lorente

29.01.2021
email@sayan.page
lorente.inaki@gmail.com

Homework 1

Solution to Question 1

To implement depth first search (DFS), the stack data structure is used. A primitive implementation of this data structure is already provided in the class `util.Stack`. In this case, our state is a tuple containing the position and the path traversed so far.
Our implementation of DFS solves the `tinyMaze` environment with a total cost of 10 by expanding 15 nodes, the `mediumMaze` environment with a total cost of 130 by expanding 146 nodes and the `bigMaze` environment with a total cost of 210 by expanding 390 nodes.

Solution to Question 2

To implement breadth first search (BrFS), the queue data structure is used. A primitive implementation of this data structure is already provided in the class `util.Queue`. In this case, our state is a tuple containing the position and the path traversed so far.
Our implementation of BrFS solves the `mediumMaze` environment with a total cost of 68 by expanding 269 nodes and the `bigMaze` environment with a total cost of 210 by expanding 620 nodes.

Solution to Question 3

To implement uniform cost search (UCS, also know as Djikstra's algorithm), the priority queue data structure is used. A primitive implementation of this data structure is already provided in the class `util.PriorityQueue`. In this case, our state is a tuple containing the position, the path traversed so far and the distance (cost) value of the node. The priorities of the nodes are set according to this distance metric.
Our implementation of UCS solves the `mediumMaze` environment with a total cost of 68 by expanding 269 nodes, the `mediumDottedMaze` environment with a total cost of 1 by expanding 186 nodes and the `mediumScaryMaze` environment with a total cost of 68719479864 by expanding 108 nodes.

Solution to Question 4

To implement A* search, the priority queue data structure is also used, the priorities of the nodes are the sum of the heuristic value and the distance (cost) value of the nodes. A primitive implementation of this data structure is already provided in the class `util.PriorityQueue`. In this case, our state is a tuple containing the position, the path traversed so far and the distance value of the node.
Our implementation of A* solves the `bigMaze` environment with a total cost of 210 by expanding 591 nodes.

Solution to Question 5

For the problem, the state representation that we chose was a tuple with two elements: the current position of Ms. Pac-Man on the map and a list containing the location of the unvisited corners. The goal state for the agent is thus to reach the last unvisited corner. In other words, our initial state is as follows:

```python
def getStartState(self):
    """
    Returns the start state (in your state space, not the full Pacman state
    space)
    """
    unvisited_corners = deepcopy(self.corners)
    return (self.startingPosition, unvisited_corners)
```

and, we check if we have reached our goal state as follows:

```python
def isGoalState(self, state):
    """
    Returns whether this search state is a goal state of the problem.
    """
    pos, unvisited_corners = state
    return pos in unvisited_corners and len(unvisited_corners) == 1
```

Solution to Question 6

To solve corners problem using heuristics, we used the `manhattanDistance` function. This function is already implemented in `util.py`. `manhattanDistance` takes two points (current position and goal position in this case)

$$\text{manhattanDistance}((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1, -y_2|$$

where, (x_1, y_1) is the current position and (x_2, y_2) is the goal position. We finally `return` the max of these distances. Our heuristic is admissible because under no circumstance can Ms. Pac-Man travel a distance that is greater than the distance that is given by the maximum of the Manhattan distances to its targets if it follows an optimal path. The heuristic is consistent as every move decreases the the heuristic value by 1 for all states.

```python
def cornersHeuristic(state, problem):
    hvalue = [0]
    for goal in state[1]:
        # Distance from current state position to goal corners
        hvalue.append(util.manhattanDistance(state[0], goal))

    return max(hvalue)
```

Solution to Question 7

This problem is solved using same logic used in previous question. In this case the positions passed to heuristic `mazeDistance` are the current position and the next food position. We return the maximum of these distance as the heuristic value. Our heuristic is admissible as we are finding the optimal distance in the maze from the current state to the target states. We return the maximum of these distances, hence our heuristic cannot exceed the optimal heuristic under any circumstances. Our heuristic is also consistent as moving a single state towards a goal decrements the heuristic by 1 for all states.

```python
def foodHeuristic(state, problem):
    hvalue = [0]

    for food in foodGrid.asList():
        # Distance from current state position to next food position
        hvalue.append(mazeDistance(state[0], food, problem.startingGameState))

    return max(hvalue)
```

Solution to Question 8

This problem can be solved using the previous A* search algorithm implemented in question 4. In addition to it, when checking if we have the reached goal state, we used the Ms. Pac-Man's position to see if there is any food at that position which means goal reached. The call to A* search is as follows:

```python
def findPathToClosestDot(self, gameState):
        problem = AnyFoodSearchProblem(gameState)
        return search.aStarSearch(problem)

class AnyFoodSearchProblem(PositionSearchProblem):
    ...
    def isGoalState(self, state):
            x,y = state
            return self.food[x][y]
```

On running `autograder.py` on our solution, the output produced is as follows:

```
Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 3/3
Question q6: 3/3
Question q7: 5/4
Question q8: 3/3
------------------
Total: 26/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```