Report for PJ1_Search

杜闻博　18307110359

1 Question 1

Finding a Fixed Food Dot using Depth First Search

1. First, I define a new class named **'node'**, which is the same as that defined in lab1. A node object has four attributes: **state**, **parent** (also a node object), **path_cost** (from start state to itself), and **actions**(using data structure 'list', recording the action sequence it will take from start to reach itself). This class is also used in next questions.

2. Using data structure **Stack**, every time pop the element the last pushed in, and push its successors into **Stack**, until pop the Goal (a **'node'** class object), return its attribute **actions**. At the same time, using a list **'explored'** to record the states which have been explored before, every time pop the element from **Stack**, check if its state is already in **'explored'**, if so, end this cycle and start next one.

3. The solution found by my DFS algorithm for mediumMaze have a length of 130.

2 Question 2

Breadth First Search

1. Using self-defined class **'node'**, the same as DFS. Using data structure **Queue**, every time pop the element the first pushed in, and push its successors into **Queue**, until find (i.e.: when pop an element/scanning an element's successors) the Goal (a **'node'** class object), return its attribute **actions**. Also using a list **'explored'** to record the states which have been explored before, the same as DFS.

2. For eight-puzzle problem, my BFS algorithm also works well without any changes.

3 Question 3

Varying the Cost Function

1. Using self-defined class **'node'**, the same as DFS, BFS. Using data structure **PriorityQueue**, every time pop the element with smallest path_cost, and push its

successors into **PriorityQueue**, until pop the Goal (a **'node'** class object), return its attribute **actions**. Also using a list **'explored'** to record the states which have been explored before, the same as DFS, BFS.

2. I get very low path costs(1) for the StayEastSearchAgent and very high path costs(68719479864) for the StayWestSearchAgent.

4 Question 4

A* search

1. Using self-defined class **'node'**, the same as DFS, BFS, UCS. Using data structure **PriorityQueue**, every time pop the element with smallest (path_cost + heuristic value), and push its successors into **PriorityQueue**, until pop the Goal (a **'node'** class object), return its attribute **actions**. Also using a list **'explored'** to record the states which have been explored before, the same as DFS, BFS, UCS.

2. For bigMaze, A* finds the optimal solution slightly faster than uniform cost search (549 vs. 620 search nodes expanded). And for openMaze, A* also finds the optimal solution slightly faster than uniform cost search (535 vs. 682 search nodes expanded).

5 Question 5

Finding All the Corners

1. I define the state representation as (position, visited_set), where visited_set records the corners visited.

2. Function **isGoalState(self, state)** returns True if and only if t contains 4 corners. For function **getSuccessors(self, state)**, if a successor is a corner, that successor's state not only changes the position but also add its position into visited_set, else only changes the position.

3. With the use of BFS algorithm, the Corner Problem expands 1921 search nodes in mediumCorners.

6 Question 6

Corners Problem: Heuristic

1. **Heuristic function :** Firstly, define 'height' and 'width', represent the short edge and long edge of the maze rectangle respectively. ①If there is only 1 corner left, return the Manhattan distance between the current position and that corner. ②If there are 2 corners left, return (Manhattan distance between that two corners + smaller Manhattan distance from the current position to the corners left). ③If there are 3 corners left, return ('height' + 'width' + smallest Manhattan distance from the current position to the two diagonal corners). ④If there are 4 corners left, return (2×'height' + 'width' + smallest Manhattan distance from the current position to the corners left). 741 nodes expanded using this heuristic function.

2. **Proof of Admissibility :** ①1 corner left, the Manhattan distance between the current position and that corner $\leq$ the real distance between them. ②2 corners left, the Manhattan distance between them + the minimum Manhattan distance between current position to each of them, $\leq$ real distance. ③3 corners left, the Manhattan distance between them(= 'height' + 'width') + the minimum Manhattan distance between current position to the two diagonal corners, $\leq$real distance ④4 corners left, 2×'height' + 'width' + smallest Manhattan distance from the current position to the corners left is obviously $\leq$real distance.

3. **Proof of Consistency :** If we can prove every move-step of a path is consistent, then it must have consistency. Let's consider two situations of a move-step. ①The step doesn't reach an corner. Then Δ(heuristic value) = $\pm 1 \leq 1$ = the real cost for this step. ②The step reaches one of the corners. Because the varying of my heuristic function is continues, doesn't have sudden change when the number of not_visited_corners changes, so Δ(heuristic value) = $1 \leq 1$ = the real cost for this step. So for each step of a path , Δ(heuristic value) $\leq$ the real cost for this step, so consistency holds.

7 Question 7

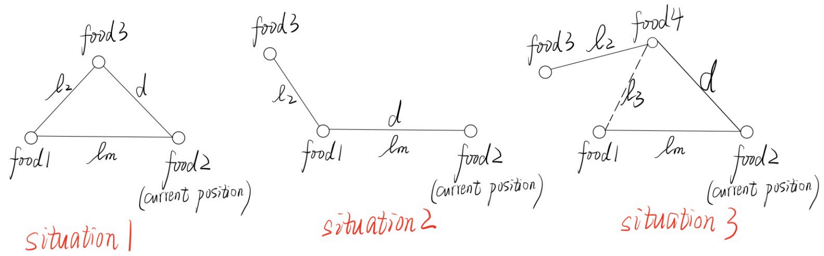
Eating All The Dots

1. **Heuristic function :** When first calling this function, using UCS algorithm finished in **search.py** to calculate the real distance between every two foods, and storing them in the dictionary **problem.heuristicInfo**. ①If there is only 1 food left, return the real distance between the current position and that food. ②If there are $\geq$ 2 foods left, first find the two foods which have the max real distance from each other using **problem.heuristicInfo**, then return (that max real distance +

smaller real distance from the current position to that two foods). 376 nodes expanded using this heuristic function.

2. **Proof of Admissibility :** It is a sub-problem which simple 'eat all foods' to 'eat two farthest foods', and is the exact solution to that sub-problem, whose cost is obviously $\leq$the real problem.

3. **Proof of Consistency :** If we can prove every move-step of a path is consistent, then it must have consistency. Let's consider two situations of a move-step. ①The step doesn't eat one of the two max-distance foods. Then that max real distance of two foods doesn't change, and the smaller real distance from the current position to that two foods will reduce/increase 1, so Δ(heuristic value) $= \pm 1 \leq 1 =$ the real cost for this step. ②The step eats one of the two max-distance foods. Assume that max-distance $= l_m$, the second two max-distance foods has a distance l_2, the smaller real distance from the position after this step (also the position of one of the two max-distance foods) to the second-max-distance two foods is d. What needs to be proved is $l_m + 1 - (l_2 + d) \leq 1$, i.e. $l_m \leq l_2 + d$. There are three situations as follows, the edge in graph is the real distance between two vertex. Because there holds $l_2 \geq l_3$, the sum of the two sides of the triangle is greater than the third side, so $l_m \leq l_2 + d$ always holds. So for each step of a path ,Δ(heuristic value) $\leq$ the real cost for this step, so consistency holds.



8 Question 8

Suboptimal Search

1. In class **AnyFoodSearchProblem**, because self.food is a **Grid** object, so in function **isGoalState(self, state)**, self.food[x][y] = 1 if there is a food in (x, y), otherwise self.food[x][y] = 0.

2. Using BFS algorithm finished in **search.py** to find the solution for **findPathToClosestDot(self, gameState)**.

3. Our agent solves this maze suboptimally in under a second with a path cost of 350.