COMP7404 Assignment1 report

Wang Chu
3035348059

Q1 Depth First Search:
Firstly, pacman doesn't go to every square that it explored. It just goes to the squares within the solution path. Furthermore, the exploration order is the same as my expectation. Because according to the function getSuccessors() of PositionSearchProblem class in searchAgents.py, the order of nodes added to the frontier is "North, South, East, West" . And the tie breaker of dfs is LIFO, so the next explored position should be the west one of initial position, which is exactly what pacman has done.

Secondly, using dfs in cases of tinyMaze and mediumMaze, pacman doesn't follow the best path which might have the least cost as I have expected. In bigMaze, dfs can find the best path, because there might be only one path to the goal position.

Q2 Breadth First Search:
Since the tie breaker of bfs is FIFO, the first explored node after the initial position should be the south one. BFS will expand more nodes than dfs, but it can find the solution with the smallest cost.

Q3 Uniform Cost Search:
With the same cost of each successor generated by every action, ucs is actually equivalent to bfs. Because the PriorityQueue data structure of util.py uses heapq.heappop() to get the lowest-priority item, which will return the same priority items in the order of being added, i.e. FIFO.

However, if you define another cost function like the one of StayEastSearchAgent which means if pacman goes to west it will have more cost, then the pacman will have different solutions.

Q4 A* search:

Cost/Expanded	DFS	BFS	UCS	A*+ ManH	A*+ EucH
openMaze	298/576	54/682	54/682	54/535	54/550

In this case, dfs has the worst performance, because the solution of this problem lies in the shallow layer of the search tree. On the other hand, compared with uniformed search strategy, A* can reduce the number of nodes expanded significantly.

Q4.1 Comparisons of the Four Search Algorithms:

Cost/Expanded	DFS	BFS	UCS	A*+ ManH	A*+ EucH
tinyMaze	10/15	8/15	8/15	8/14	8/13
mediumMaze	130/146	68/269	68/269	68/221	68/226
bigMaze	210/390	210/620	210/620	210/549	210/557

Q5 Representation for Corners Problem:
In this problem, I use a tuple (position, corners_hasFood) as the representation for the state, where "position" is a tuple (x,y) to show the position of the agent and "corners_hasFood" is another tuple whose initial value is (Ture, True, True, True) to show whether every corner has food. The index represents the corner in "self.corners". Once the agent has been to one corner such as "self.corners[0]", we will set the corresponding value "self.corners[0]" to False.

Q6 Heuristics for Corners Problem:
Firstly, calculate the manhattan distance between the agent and its closest corner A. Then calculate the manhattan distance between A and its closest corner B and so on. We use the sum of these manhattan distances as the heuristic of this problem.

Q6.1: Check Heuristic: Q1: ==; Q2: non-trivial; Q3:admissible; Q4: >; Q5: consistent.

Q7 Food Heuristic:
In this problem, I calculate the manhattan distance between the agent and its closest dot A. Then calculate the manhattan distance between A and its farthest dot B. Then use the sum of these two distances to be the heuristic.

Actually, the best heuristic might be the smallest manhattan distance of the path which can go through all the dots. This is a NP problem and very difficult to implement, therefore I use a simpler one.

Q8 Hill-climbing Search for 8-Queens Problem:
In the function getBetterBoard() of class Board in solveEightQueens.py, I use a list "minCost_List" to store the positions which have the same minimum "numberOfAttacks" and randomly choose one of them as the next move to generate a better board.

For the stop criteria, I allow the 50 consecutive moves when the "currentNumberOfAttacks" is less than or equal to "newNumberOfAttacks". After 50 times of iteration, if "currentNumberOfAttacks" is still not equal to 0, I will randomly generate a board and do the same thing. If we have randomly generated 50 boards and still no solution, then we will break the loop.