Artificial Intelligence Course

Project 1: Search in Pacman

First name	Last name	Student number
Hao	**Ban**	**2591928**
Shengdong	**Wang**	**2616227**

Comments about the assignment (if you have)

Question 1: Finding a Fixed Food Dot using Depth First Search (3 points)

In this Question, we used **stack** as the data structure to solve this problem. We need to record the coordinate and path for each node. The tuple in stack likes ((x,y), [path]).We added the first node in the stack, and then add its child nodes which are in different directions.

For example, every node has four child nodes in different directions. We put successors (four child nodes) in the end of stack. In the data structure of stack, the last node to be pushed is the first to be popped. And then, when we got the last one child node, we need to find the whole successors of it. After that, we put all of successors in the stack. And we also got the last one child node. It will access each successor in only one branch until the currency node has been visited before.

So, the depth first search algorithm depends on the data structure of stack! After figuring out this, we used the data structure of list to store path and nodes has been visited. We always find the child nodes and their child nodes only in same one direction. If all of the nodes in this direction have been visited, the brother node of leaf node in that direction should be visited.

Question 2: Breadth First Search (3 points)

In this Question, we used **queue** as the data structure to solve this problem.

Except the data structure, other things are same with last one. We need to record the coordinate and path for each node. The tuple in stack likes ((x,y), [path]). We added the first node in the queue, and then add its child nodes which are in different directions.

For example, every node has four child nodes in different directions. We put successors (four child nodes) in the end of queue. In the data structure of queue, the first node to be pushed is the first to be popped. And then, when we got the first one child node, we need to find the whole successors of it. After that, we put all of successors in the end of queue. And we got the second child node of the initial node. It will access all successors layer by layer until all of nodes has been visited.

So, the breadth first search algorithm depends on the data structure of queue! In this part, we also used the data structure of list to store path and nodes has been visited. We always find the all child nodes of each layer nodes. The order of parents in the queue depends the order of child node in the queue.

Question 3: Varying the Cost Function (3 points)

In this Question, we used **priority queue** as the data structure to solve this problem and implement Uniform Cost Search. In order to solve the problem of updating priority queue, we use data structure of heap. In our assignment, we use min heap. The tuple in Priority Queue of this Question likes (((x,y), [path]), priority).

After checking the class Priority Queue in util.py, we found the structure of each item. Priority, count and item are stored in heap. And then, we designed our algorithm.

At first, we need to judge the state of initial node and get the successor of initial one. Because we need to update all of the path until all of the nodes has been visited and found the least cost path. So, all nodes have been divided into two groups. One is the position already has one path, another is the position hasn't been connected from start.

For the position which doesn't have a path to it, we need to record this path and priority. For the position which has an existed path, we need to judge the cost(priority) of this path. If the cost of new path is less than old one, we need to update it.

Question 4: A* search (3 points)

In this Question, we also used **priority queue** as the data structure to solve this problem and implement A * Search. In order to solve the problem of updating priority queue, we use data structure of heap. In our assignment, we use min heap.

After checking the class Priority Queue in util.py, we found the structure of each item. The tuple in Priority Queue likes (((x,y), [path]), heuristic). In our algorithm, we set the default of heuristic is nullHeuristic. In the initial state, It means f(n) = g(n) + h(n), and g(n) = 0.

At first, we push the state of initial node (coordinate) and path in priority queue. And then, we get successors of the head of queue (the top of min heap). In each access, we put all of successors in queue, and always get the min cost direction. In the next round, we also put all successors in queue and find the least cost one. The loop will terminate until all of nodes in priority queue have been visited or we find the goal state.

Question 5: Finding All the Corners (3 points)

In this Question, we modified the class of CornersProblem. In initial function, __init__, we set 0 means corner hasn't been visited, and 1 means corner has been visited. In getStartState function, it will get a state include (startingPosition, initialState). Starting Position means initial position and initial State record the state of four corners.

In the function of isGoalState, we checked all the value in State[1]. If any of them (corners) are 0, it means False.

In the function of getSuccessors, we just follow the annotation. We got the position coordinate and corners state. And then, we set dx, dy, nextx and nexty. And then, we use self.walls to judge if the state is wall. If we arrived at four corners, we will update the states[1].

Question 6: Corners Problem: Heuristic (3 points)

Explain your Heuristic function …

Question 7: Eating All The Dots (4 points)

Explain your new state representation and the heuristic function…

Question 8: Suboptimal Search (3 points)

Explain how did you do the suboptimal search …