**Task2**

**实验报告 20214809 凌鹏**

**Question 1 (4 points): Finding a Fixed Food Dot using Depth First Search：**

**实验记录**：在完成DFS算法之前，我们需要先对整个迷宫的数据结构进行了解，主要包括：

①迷宫的表示形式。在查看**search.py**和**game.py**等源码后，发现*“problem.walls”*代表当前的迷宫形状，其类型为**Grid**，定义在**game.py**中，并且每一个位置的值都是bool类型，True表示为墙壁，False表示为道路。

②起始点和终止点位置以及表达形式。查看源码后，发现起始点表示为“***problem.getStartState()***”，而终止点表示为“***problem.goal***”，每一个状态都是一个**（i, j）**元组，可以通过“***problem.* *isGoalState(nowState)***”来判断当前是否已经到达终点。

③迷宫中“行走”的方式，主要是指行走的方向。经过查看源码，发现**game.py**文件中定义了***Actions***和***Directions***两个相关类，动作形式也同时定义。

④我们发现不同的**problem**类对于具有不同的属性和方法（比如***searchAgents.PositionSearchProblem***类就不具备**getExpandedStates()方法,而searchTestClasses.GraphSearch类就有该方法），**这种差异要求我们在编写DFS函数时必须小心的处理。

⑤使用***autograde.py***函数进行评价时我们发现，正确答案包括两个部分，一个是路径（类似A->B这种），此时最后一条路径的终点必定是**目标**位置；另一部分是状态转变（类似[A, B, C]这种），其实就是每条路径的起始点，所以一定不能包含目标位置在内。这一差异又要求我们在DFS中特殊处理。

了解上述的基本情况后，就可以动手实现DFS算法，思路比较简单，不做过多叙述，唯一需要注意的：不要忘记记录已经走过的位置。下面是结果截图：

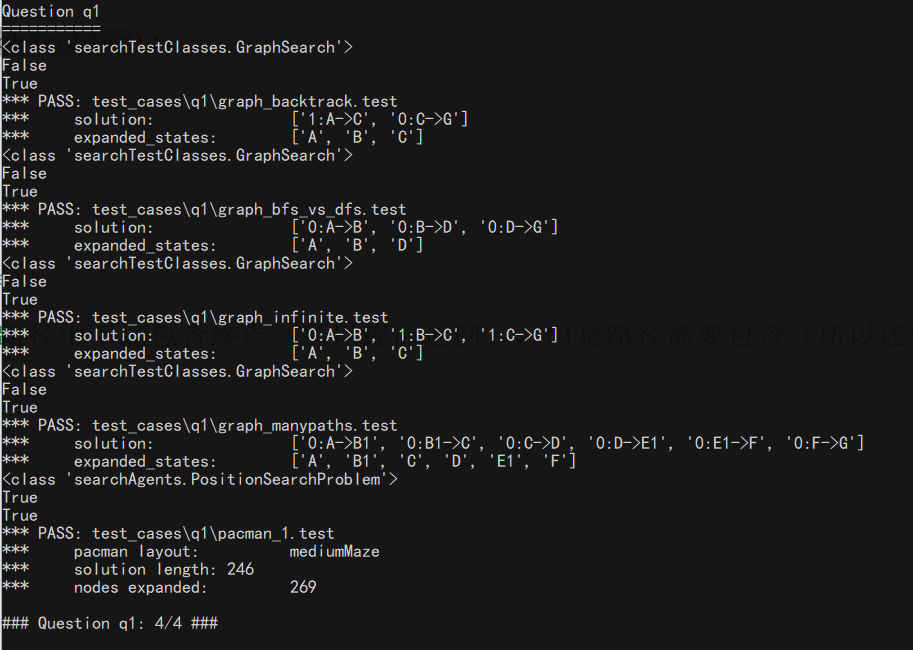


图1：Question1结果图

**-------------------------------------------------------------------------------**

**Question 2 (4 points): Breadth First Search：**

**实验记录：**这个实验在完成的时候问题比较多。首先，主要是由于不同的problem类函数和属性存在差异，导致我们需要在BFS内部实现兼容。其次，由于最终需要的是从起点到终点的路径，由于BFS的特征，在保存每一次BFS的状态转移时（包括动作），必须保存相反的状态转移，以便最终根据目标反向回溯。Autograde结果如下图所示；总的来说BFS比DFS在该项任务中更难。

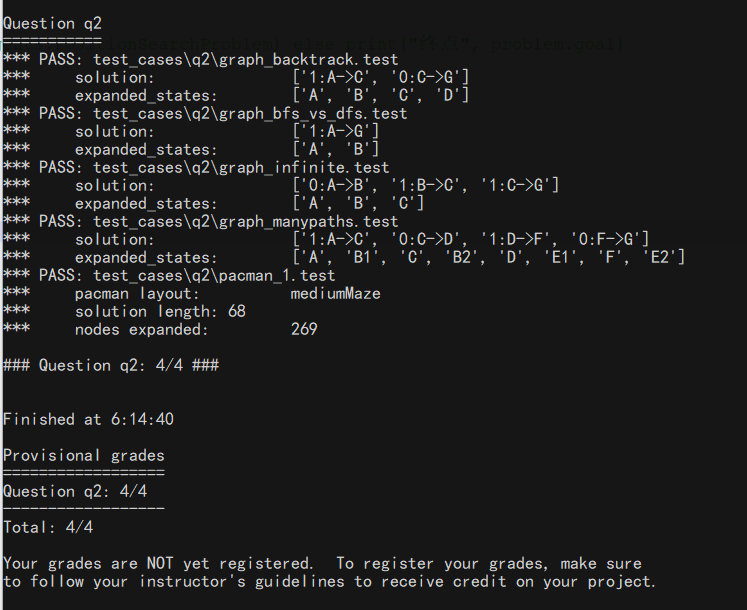


图2：Question2结果图

-------------------------------------------------------------------------------------------

**Question 3 (4 points): A\* search：**

**实验记录：**A\*算法总体思路和BFS类似，但是与BFS不同点在于：BFS每个状态的优先级一样，如果使用队列装载所有的状态，那么每个状态出队时按照先进先出的规则即可；而如果使用的是A\*算法，那么从状态队列中取出下一个转移状态之前，需要考虑队列中所有状态的价值（cost），在Q3中则是直接使用的城市距离来衡量每个状态的cost：计算每个状态与终点goal直接的城市距离。因此，在实现的时候，我们只需要将队列改为优先队列，然后操作方式和BFS一样即可。实验结果如:3所示。

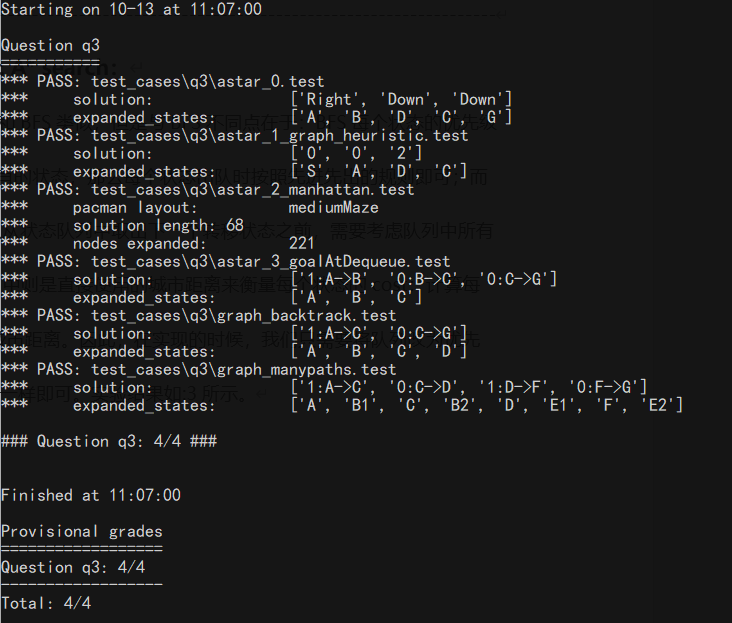


图3：Question3结果图

-------------------------------------------------------------------------------------------

**Question 4 (3 points): Finding All the Corners**

**实验记录：**比较简单，直接上图4。

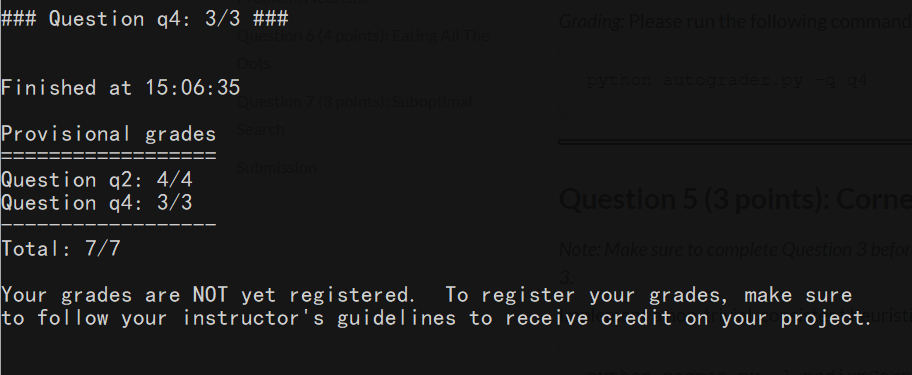


图4：Question4结果图

-------------------------------------------------------------------------------------------

**Question 5 (3 points): Corners Problem: Heuristic**

**实验记录：**Q5需要我们设计一个能够使得agent从起始点出发，使用最短时间到达四个边角的启发式算法。在这之前，我们完成过从起始点使用最短时间到达一个终点的方法，在那里我们使用了A\*算法来完成。然后我们回到现在的问题，并且思考得到解决方案如下：1）找到当前状态时，所有的尚未访问过的终点corners；2）我们计算从当前状态(state)到所有未访问过的终点之间的距离，这里使用DFS计算最佳；3）我们选择返回从当前状态到所有未访问过的corners的距离最大的那个作为cost。通过上述三步即可在尽可能短的时间内访问所有的corners。



图5：Question5结果图

-------------------------------------------------------------------------------------------

**Question 6 (4 points): Eating All The Dots**

**实验记录：**该问题为从起始点出发，吃掉maze中所有的点。该问题其实又是Q5的更难版本。但是我们认为仍然可以采取解决Q5类似的步骤解决该问题，但是，考虑到此时的dots一般会比较多（至少会比Q5中的corners多），所以，我们在解决Q6时不需要再使用DFS计算距离，而是直接使用城市距离即可。其他和Q5解决方案一样。

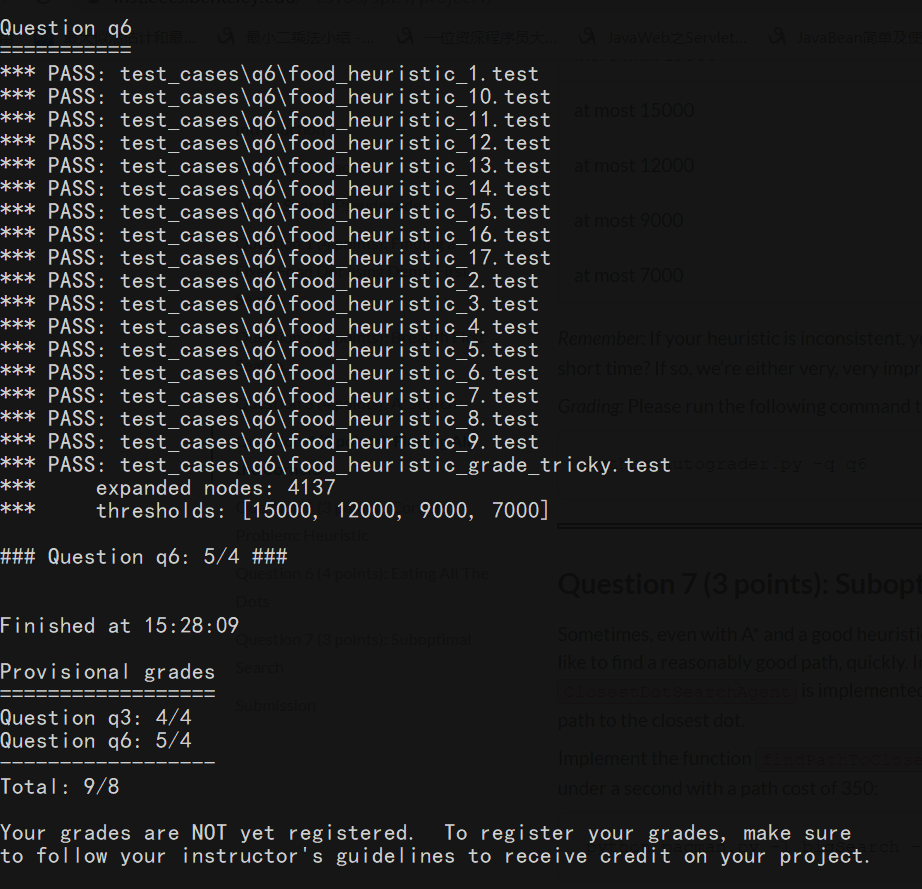


图6：Question6结果图

-------------------------------------------------------------------------------------------

**Question 7(3 points): Suboptimal Search**

**实验记录：**在该问题中，根据问题描述，我们需要always greedily的吃dot，因此，我们需要对Q6中的方案进行更改，在这里我们使用A\*算法来解决该问题，A\*的启发信息和Q3中类似。

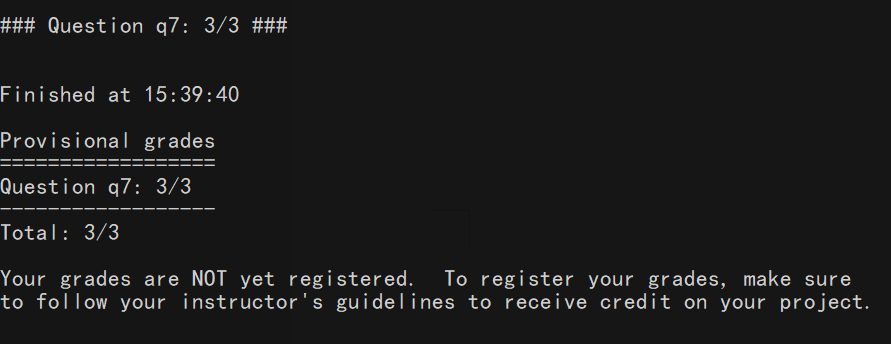


图7：Question7结果图