Project1_Search 实验报告

小组成员

傅宇千 刘炳德 刘志豪

Question 1

使用DFS搜索，过程如下：

1. 将起点加入一个栈
2. 将栈顶元素出栈，判断其是否为目标点，若是，则跳出循环；若不是，则进入步骤3
3. 将栈顶弹出，调用 `getSuccessors` 函数获取其后继位置，并将弹出的栈顶加入closed集合
4. 将后继结点入栈，回到步骤2

Question 2

使用BFS搜索，相较于DFS，变化仅仅是将栈换成队列，其循环结构不变.BSF过程如下：

1. 将起点加入一个队列
2. 将队头元素出队，判断其是否为目标点，若是，则跳出循环；若不是，则进入步骤3
3. 将队头弹出，调用 `getSuccessors` 函数获取队头结点后继位置，并将弹出的队头加入closed集合
4. 将后继结点入队，回到步骤2

Question 3

使用UCS搜索，相较于BFS，变化仅仅是将队列替换为优先队列，其循环结构不变.UCS过程如下：

1. 将起点加入一个优先队列
2. 将队头元素出队，判断其是否为目标点，若是，则跳出循环；若不是，则进入步骤3

3. 将队头弹出，调用 `getSuccessors` 函数获取队头结点后继位置，并将弹出的队头加入closed集合
4. 将后继结点入队，回到步骤2

Question 4

在这个问题中，我们选择A*算法.A*算法选择当前位置与目标处的曼哈顿距离加上节点已行走过的长度作为启发函数，定义当前位置与目标的曼哈顿距离为$h(x)$，从起点到当前位置所需的代价为$g(x)$，则启发函数$f(x)=h(x)+g(x)$.A*算法流程如下：

1. 将起点加入一个优先队列
2. 将队头元素出队，判断其是否为目标点，若是，则跳出循环；若不是，则进入步骤3
3. 将队头弹出，调用 `getSuccessors` 函数获取队头结点后继位置，并将弹出的队头加入closed集合
4. 将后继结点入队
5. 以启发式函数$h(x)$更新优先队列，回到步骤2

Question 5

在这个问题中，我们设定状态state为((y,x),list)，其中(y,x)为目前pacman所处位置，而最大长度为4的list用于表示已经抵达过的corners，其中每个元素都是二元组(y,x)，用来表示corner的坐标.

对于 `getSuccesors` 函数，我们返回的状态不仅要更新下一个状态pacman所处位置，还要更新下一个状态已经抵达过的corners.

对于 `isGoalState` 函数，当state[1]的长度为4时，表示所有corners都已经达到，返回True,否则返回False.

只需更改状态的表示，无需更改bfs算法的具体实现.

Question 6

在这个问题中，我们选择的启发式函数表示如下：

设4个角落的集合为C，p为当前pacman所在位置，定义：

$$nearest(p,C)=\begin{cases}\text{集合}C\text{中离}p\text{最近的点} & C\neq\emptyset\\ \text{空} & C=\emptyset\end{cases}$$

则有：

$$\begin{cases} p1 = nearest(p, C) \\ p2 = nearest(p1, C - \{p1\}) \\ p3 = nearest(p2, C - \{p1, p2\}) \\ p4 = nearest(p3, C - \{p1, p2, p3\}) \end{cases}$$

定义

$$manhattan(p1, p2) = \begin{cases} p1到p2的曼哈顿距离 & (p1 \neq 空, p2 \neq 空) \\ 0 & (p1 = 空或p2 = 空) \end{cases}$$

则启发式函数的值为：

$$h = manhattan(p, p1) + manhattan(p1, p2) + manhattan(p2, p3) + manhattan(p3, p4)$$

Question 7

在这个问题中,我们选择的启发式函数表示如下:

利用 `searchAgents.py` 内置的 `mazeDistance` 函数,求得当前点到各个食物之间的距离,并返回其最大距离作为启发式函数.

首先说明选择 `mazeDistance` 函数的原因,如果和 `Question 6` 一样使用曼哈顿距离的话,可能会因为墙壁的原因,反而需要绕远路才能吃掉,这样不容易满足一致性和可接受性的判定.(而在角落的问题中,曼哈顿距离则比较容易满足该判定)

同时，我们没有将最近的实际距离作为启发值，而是用最远的距离.因为在吃豆人在吃掉最远的食物时,一定会顺带吃掉路上的食物，而最远的食物的距离一定是小于吃掉所有食物的距离，这样也可以满足对可接受性和一致性的判定.

Question 8

实现贪心算法比较容易,只需要将问题目标转化为单个食物点即可,即每次将问题目标转化为到达单个食物点(修改 `isGoalState` 函数,判断当前位置是否位于食物点),这样难以达到全局最优,但求解速度大大提升,也被称作Suboptimal(次优)路径.

实验结果

实验结果如下图:

```
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
```

除在进行 `question 7` 的 `food_heuristic_grade_tricky.test` 时会等待较长时间外,其余测试都能短时间内得到答案.