哈爾濱工業大學

人工智能实验报告

题　　目　搜索策略

专　　业　视听觉信息处理

学　　号　1170300513

学　　生　陈鋆

指 导 教 师　李钦策

同 组 人 员　强文杰，张亚博，王家琪，束魏琦

一. 简介/问题描述

1.1 待解决问题的解释

实验要求采用且不限于课程第四章内各种搜索算法此编写一系列吃豆人程序解决以下列出的问题 1-8，包括到达指定位置以及有效的吃豆等。

1.2 问题的形式化描述

问题 1-4：分别利用深度优先算法，宽度优先算法,代价一致算法和 A* 算法找到一个特定的位置的豆；

结合上述的搜索算法：

问题 5：实现 CornersProblem，从而使吃豆人将以最短的路径找到迷宫的四个角落作为目标；

问题 6：为 CornersProblem 给出一个非平凡且一致的（non-trivial, consistent）启发式函数 cornersHeuristic，以通过扩展尽量少的节点找到 4 个角落，吃掉角落上的食物；

问题 7：选择一个良好的启发式函数，用尽可能少的步数，也就是扩展较少的节点以 找到所有的食物；

问题 8：实现一个优先吃最近的食物的路径规划算法，需要 实现 AnyFoodSearchProblem 中的 isGoalState 函数和 ClosestDotSearchAgent 中的 findPathToClosestDot 函数。

1.3 解决方案介绍（原理）

问题 1-4：使用相同的搜索方案，分别用序列记录当前已经过的路径（actions），遍历过的 节点（close 表）。开始时首先将问题的初始节点与空的 actions 作为列表添加到 open 表中。之 后进入循环，弹出当前 open 表中第一个元素，并将此节点加入到 close 序列中。如果当前节点 为目标节点则返回路径 actions。否则，保存路径部分 actions。遍历刚才弹出节点的所有后继 节点,如果其还没有被遍历过(即还没有被加入 close 中)，那么将该节点与相应的 action 添加到 open 表中。本轮循环结束。若直到栈空仍没有到达目标节点，说明搜索失败，返回一个空的序列。

利用实验所给的数据结构，问题 1-4 的 open 表分别使用栈，队列，权值为路径和的优先队列，权值为路径和与 h(x)之和的优先队列。

问题 5：实现搜索所有角落：

定义问题的状态为一个二元组，分别为当前的位置与所遍历过的角落。

GetStarState 函数返回这个二元组。

isGoalState 函数返回一个布尔值，判断已经遍历过的顶点个数是否为 4， 若为 4 则停止遍历；

getSuccessor 函数从当前为位置开始朝四个方向移动，如果该位置不是 墙且未遍历过那么将其加入返回的序列中。

问题 6：构建合适的启发函数，完成 searchAgents.py 文件中的 cornersHeuristic 角落搜索问题。启发性函数的启发策略为从当前位置依次向四个角落中还未被遍历过的且曼哈顿距离最小的位置的开始。并继续重复上述动作，直至所有角落被遍历。最后将这些曼哈顿距离之和作为返回值。

问题 7：用尽可能少的步数吃掉所有的豆子。完成 searchAgents.py 文件中的 FoodSearchProblem 豆子搜索问题。利用之前 A*算法进行搜索，启发性函数的启发策略为在当前位置下吃到下一个食物的最短距离。在这个问题后有一个实现了的求在迷宫中的两点的距离的函数。于是对这个函数加以利用，启发函数的返回值为当前节点到所有剩余食物中距离最近的值。

问题 8：定义一个优先吃最近的豆子函数是提高搜索速度的一个好的办法。首先补充完 AnyFoodSearchProblem 类中对于目标节点的测试部分。对当前剩余的所有食物进行遍历，找到距离当前状态最近的食物的曼哈顿距离，如果这个距离为 0，那么该节点即为目标节点。选取 A*算法搜索得到路径。

二． 算法介绍

2.1 所用方法的一般介绍

问题 1-4：搜索过程基本相同，唯一不同的是 open 表的数据结构，问题 1 是栈；问题 2 是队列；问题 3 是优先队列，权值为起点到当前节点路径和；问题 4 也是优先队列，不过权值为从 起点到 当前节点路径和与函数 h(x)之和，其中搜索

过程如下：

（1）把初始节点放入 open 表中，建立一个 close 表，置为空；

（2）检查 open 表是否为空表，若为空，则问题无解，失败退出；

（3）把 open 表的第一个节点取出放入 closed 表，并记该节点为 n；

（4）考察节点 n 是否为目标节点，若是则得到问题的解成功退出；

（5）若结点 n 不可扩展，则转第二步；

（6）扩展节点 n，将其子节点放入 open 表的首部，并为每个子节点设 置指向父节点的指针，转向第二步。

问题 5：定义问题的**状态为一个形如(coord, foods_statebool)的二元组，其中 coord 为吃豆人所在的位置坐标(x,y)foods_statebool 为四个角落食物的状态列表**，初始值为 [False, False, False, False], foods_statebool 与 self.corners((1,1), (1,top), (right, 1), (right, top))位置一一对应，若对应位置的食物未被吃掉，则 foods_statebool 中对应项为 False,若已经被吃则 foods_statebool 中对应项为 True。。GetStarState 函数返回这个二元组；isGoalState 函数返回一个布尔值，判断已经遍历过的顶点个数是否为 4，若为 4 则停止遍历；getSuccessor 函数从当前为位置开始朝四个方向移动，如果该位置不是墙且未遍历过那么将其加入返回的序列中。

问题 6：启发式函数值应是真实代价的下界。而对于迷宫而言，如果我们选择**在无墙的迷宫中从某位置到达目标状态的最小代价作为启发式函数值**，那么这个值一定是真实代价的下界。启发性函数的启发策略为从当前位置依次向四个角落中还未被遍历过的且曼哈顿距离最小的位置的开始。并继续重复上述动作，直至所有角落被遍历。最后将这些曼哈顿距离之和作为返回值。

问题 7：用之前 A*算法进行搜索，启发性函数启发函数使用 **mazeDistance(point1, point2, gameState)函数**,可计算在迷宫中从一个点到达另一个点所需要的真实最小代价。使用此距离函数可**比使用曼哈顿距离函数产生一个更紧的下界（下确界）**。对这个函数加以利用，启发函数的返回值为当前节点到所有剩余食物中距离最近的值。这样选择启发式函数值可以使节点更倾向于扩展到剩余食物的中心位置，使节点距离剩余食物中的每一个都不至于很远。其次，这种启发式函数值选取方法还可以防止对某个食物节点的临近位置的过度探索，不会被局限在迷宫的某个局部，而是全局考虑。最后可以把离当前位置最远的那个豆子与当前位置之间的距离作为当前位置的启发式距离，即可以看做把最远的豆子作为目标点，因为我们肯定是吃完当前位置附近的豆子，最后再吃最远的豆子。

问题 8：本问题的运行逻辑是不断的使用 findPathToClosestDot 寻找到最近的食物的路径并将该路径加入到总路径之中，直到所有的食物均被吃完。因而在 isGoalState 函

数中，要定义将吃掉一个食物作为目标状态，以使 findPathToClosestDot 函数在吃到最近的食物后结束并返回 action 列表。而在 findPathToClosestDot 函数中，我们要给出吃豆人到最近的食物的 actions 列表，因此决定使用一定会产生最优解的广度优先搜索算法（BFS）进行搜索。本函数同时还要把已经到达的食物标记为已经吃掉了的状态，具体为在 food 对应的 Grid 对象中将相应的位置置为 False。首先补充完 AnyFoodSearchProblem 类中对于目标节点的测试部分。对当前剩余的所有食物进行遍历，找到距离当前状态最近的食物的曼哈顿距离，如果这个距离为 0，那么该节点即为目标节点。选取 A*算法搜索得到路径。

2.2 算法伪代码

通用搜索算法伪代码如下图所示，不同的部分是 open 表使用的数据结构：问题一是栈；问题二是队列；问题三是优先队列，权值为起点到当前节点路径和；问题四也是优先队列， 不过权值为从起点到当前节点路径和与函数 h(x)之和。

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

三. 算法实现

3.1 实验环境与问题规模

实验环境：Windows10，Python2.7，Pycharm

问题规模：地图中坐标的个数

3.2 数据结构

栈：后进先出，作为问题 1 的 open 表

队列：先进先出，作为问题 2 的 open 表

优先队列：元素进入后，根据权值大小排序，权值最小的先出，作为问 的 open 表

问题 searchProblem：搜索问题的数据结构，可获得问题的初始状态，目标状态，后继节点，以及从起点到当前节点最小代价 Actions：搜索路径的列表，保存搜索路径

3.3 实验结果

问题 1：与其他算法结果对比后发现，使用深度优先算法可以获得到一个特定位置的路径，但不一定是最短路径；并且深度优先算法在搜索过程中，不考虑路径的代价，即两个节点之间距离可看作恒定。

问题 2：与代价一致，A*算法结果对比后发现，在任意两个节点距离相等的情况下，使用宽度优先算法可以得到到一个特定位置豆的最短路径；而在其他情况下，未必得到最短路径。

问题 3：观察并分析实验结果后发现，无论任意两点间距离是否相同，使用代价一致算法均可以获得到达一个特定位置的最短路径。但这个算法类似于暴力搜索，搜索代价较大，速度较慢。

问题 4：与之前三个实验结果对比后发现，有了合适的启发函数，A*算法不仅可以得到最短路径，还可以大大降低搜索最短路径的代价，速度较快。

问题 5：结果如 3.4 中问题 5&6 的图 1 所示，对于各个测试文件，均能找到到一条访问所有四个角落的最短的路径，但搜索代价较大，类似于暴力搜索。可见对于效率方面，算法还有很大的改进空间。

问题 6：按照 2.1 所述构建启发函数，结果如 3.4 节中问题 5&6 图 2 所示，可以看出，启发式搜索可以在保证 得到问题最优解的基础上，一定程度上提高算法的搜索效率。。

问题 7：此问题关键在于选取合适的启发函数，按照 2.1 中所述选择了相关启发函数；在确定了具有代表性的食物位置之后，还要确定使用何种距离：若使用曼哈顿距离，扩展了 9551 个节点，搜索代价过大。使用 mazeDistance，可以产生比曼哈顿距离更紧的下界，结果如 3.4 节问题 7&8 中图 1 所示，扩展了 4731 个节点，效果有了一定程度的改善。这也说明，使用 maze 距离可以产生此问题的最优解。

问题 8：实验结果如 3.4 中问题 7&8 的图 2 所示，与最优解的结果对比，算法得到的结果路径更长，说明每步选取最近食物的贪心算法对于此问题并不能产生最优解，而是次优解。但大大提升路径的搜索速度，使效率也有了较大的提高，在最优解和搜索效率之间形成了一个合理的权衡。

3.4 系统中间及最终输出结果（要求有屏幕显示）

问题 1&2：

```
Question q1
===========

*** PASS: test_cases\q1\graph_backtrack.test
***     solution:       ['1:A->C', '0:C->G']
***     expanded_states:    ['A', 'D', 'C']
*** PASS: test_cases\q1\graph_bfs_vs_dfs.test
***     solution:       ['2:A->D', '0:D->G']
***     expanded_states:    ['A', 'D']
*** PASS: test_cases\q1\graph_infinite.test
***     solution:       ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:    ['A', 'B', 'C']
*** PASS: test_cases\q1\graph_manypaths.test
***     solution:       ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***     expanded_states:    ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases\q1\pacman_1.test
***     pacman layout:      mediumMaze
***     solution length: 130
***     nodes expanded:     146

### Question q1: 3/3 ###
```

图 1

```
Question q2
===========

*** PASS: test_cases\q2\graph_backtrack.test
***     solution:       ['1:A->C', '0:C->G']
***     expanded_states:    ['A', 'B', 'C', 'D']
*** PASS: test_cases\q2\graph_bfs_vs_dfs.test
***     solution:       ['1:A->G']
***     expanded_states:    ['A', 'B']
*** PASS: test_cases\q2\graph_infinite.test
***     solution:       ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:    ['A', 'B', 'C']
*** PASS: test_cases\q2\graph_manypaths.test
***     solution:       ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:    ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q2\pacman_1.test
***     pacman layout:      mediumMaze
***     solution length: 68
***     nodes expanded:     269

### Question q2: 3/3 ###
```

图 2

问题 3&4：

```
Question q3
===========

*** PASS: test_cases\q3\graph_backtrack.test
***     solution:       ['1:A->C', '0:C->G']
***     expanded_states:    ['A', 'B', 'C', 'D']
*** PASS: test_cases\q3\graph_bfs_vs_dfs.test
***     solution:       ['1:A->G']
***     expanded_states:    ['A', 'B']
*** PASS: test_cases\q3\graph_infinite.test
***     solution:       ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:    ['A', 'B', 'C']
*** PASS: test_cases\q3\graph_manypaths.test
***     solution:       ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:    ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q3\ucs_0_graph.test
***     solution:       ['Right', 'Down', 'Down']
***     expanded_states:    ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q3\ucs_1_problemC.test
***     pacman layout:      mediumMaze
***     solution length: 68
***     nodes expanded:     269
*** PASS: test_cases\q3\ucs_2_problemE.test
***     pacman layout:      mediumMaze
***     solution length: 74
***     nodes expanded:     260
```

图 1

```
Question q4
===========

*** PASS: test_cases\q4\astar_0.test
***     solution:       ['Right', 'Down', 'Down']
***     expanded_states:    ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q4\astar_1_graph_heuristic.test
***     solution:       ['0', '0', '2']
***     expanded_states:    ['S', 'A', 'D', 'C']
*** PASS: test_cases\q4\astar_2_manhattan.test
***     pacman layout:      mediumMaze
***     solution length: 68
***     nodes expanded:     221
*** PASS: test_cases\q4\astar_3_goalAtDequeue.test
***     solution:       ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:    ['A', 'B', 'C']
*** PASS: test_cases\q4\graph_backtrack.test
***     solution:       ['1:A->C', '0:C->G']
***     expanded_states:    ['A', 'B', 'C', 'D']
*** PASS: test_cases\q4\graph_manypaths.test
***     solution:       ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:    ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###
```

图 2

问题 5&6：

```
Question q5
===========

*** PASS: test_cases\q5\corner_tiny_corner.test
***     pacman layout:      tinyCorner
***     solution length:        28

### Question q5: 3/3 ###
```

图 1

```
Question q6
===========

*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
path: ['North', 'East', 'East', 'East', 'East', 'North', 'North',
path length: 106
*** PASS: Heuristic resulted in expansion of 741 nodes

### Question q6: 3/3 ###
```

图 2

问题 7&8：

```
Question q7
===========

*** PASS: test_cases\q7\food_heuristic_1.test
*** PASS: test_cases\q7\food_heuristic_10.test
*** PASS: test_cases\q7\food_heuristic_11.test
*** PASS: test_cases\q7\food_heuristic_12.test
*** PASS: test_cases\q7\food_heuristic_13.test
*** PASS: test_cases\q7\food_heuristic_14.test
*** PASS: test_cases\q7\food_heuristic_15.test
*** PASS: test_cases\q7\food_heuristic_16.test
*** PASS: test_cases\q7\food_heuristic_17.test
*** PASS: test_cases\q7\food_heuristic_2.test
*** PASS: test_cases\q7\food_heuristic_3.test
*** PASS: test_cases\q7\food_heuristic_4.test
*** PASS: test_cases\q7\food_heuristic_5.test
*** PASS: test_cases\q7\food_heuristic_6.test
*** PASS: test_cases\q7\food_heuristic_7.test
*** PASS: test_cases\q7\food_heuristic_8.test
*** PASS: test_cases\q7\food_heuristic_9.test
*** PASS: test_cases\q7\food_heuristic_grade_tricky.test
***     expanded nodes: 4137
***     thresholds: [15000, 12000, 9000, 7000]

### Question q7: 5/4 ###
```

图 1

```
Question q8
===========

[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_1.test
***     pacman layout:      Test 1
***     solution length:        1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_10.test
***     pacman layout:      Test 10
***     solution length:        1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_11.test
***     pacman layout:      Test 11
***     solution length:        2
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_12.test
***     pacman layout:      Test 12
***     solution length:        3
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_13.test
```

图 2

总结果：

```
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
```

四． 总结及讨论

本次四个搜索算法，由于仅仅是 Open 表中的排序方式和代价的计算有区别外，其他大部分结构都是一样的，因此将四个算法写在了一起，组合成了一个函数。后面四个问题是在前四个搜索算法已经写好的条件下计算的，根据不同的要求，填写代码不同的部分并选择合适的启发式函数，最后实现一个满足条件的算法。在实验中，我们实现了 DFS,BFS,UCS,A*等搜索算法，并且，深刻地领会到了三者的差异：DFS 不能得到问题最优解；BFS 只有在任意节点之间距离相等情况下，才能得到最优解；UCS 与 A*算法能得到任意情况下到特定位置的最优解，并且 A*搜索代价一般要小于 UCS。同时在找助教验收的过程中，也认识并学习到了启发式函数的选取要求和方法以及可用性、一致性的证明，对搜索算法有了进一步的理解。

另外，对实验的总结与思考：本次实验并不是从零开始，而是在已有的的代码上进行完善来完成实验要求。观之似易，但是实际上是有一定的难度的。因为在完善的过程中，需要对已经完成的程序进行阅读理解，如数据结构的定义、输入输出等。 我认为这种类型的实验模式十分合理，一举两得，既可以锻炼编程能力也可以锻炼阅读代码的能力，这两样能力对于计算机专业的学生来说是缺一不可的。

参考文献

[1] 人工智能实验大纲1.2

[2] 王万森.人工智能原理及其应用[M].北京：电子工业出版社，2012.09.01