



哈爾濱工業大學 (深圳)
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期: 2024 春季
课程名称: 人工智能 (实验)
实验名称: _____
实验性质: 综合设计型
实验学时: 4 地点: T2506
学生班级: 3 班
学生学号: 210110327
学生姓名: 兰锐
评阅教师: _____
报告成绩: _____

实验与创新实践教育中心制

2024 年 5 月

1、PositionSearchProblem: 问题 1-4

1.1 代码实现和说明

(1) 贴出代码截图，贴出对应的 autograder 测试结果截图，说明四种算法实现上的不同。后续问题 5-8 也贴出对应的代码截图和 autograder 测试结果截图。

```
def depthFirstSearch(problem: SearchProblem):  
    """  
    Search the deepest nodes in the search tree first.  
  
    Your search algorithm needs to return a list of actions that reaches the  
    goal. Make sure to implement a graph search algorithm.  
  
    To get started, you might want to try some of these simple commands to  
    understand the search problem that is being passed in:  
  
    print("Start:", problem.getStartState())  
    print("Is the start a goal?", problem.isGoalState(problem.getStartState()))  
    print("Start's successors:", problem.getSuccessors(problem.getStartState()))  
    """  
    """ YOUR CODE HERE """  
    closed = []  
    fringe = util.Stack()  
  
    current = (problem.getStartState(), [], [])  
    fringe.push(current)  
  
    while not fringe.isEmpty():  
        node, path, total = fringe.pop()  
        if problem.isGoalState(node):  
            return path  
        if node not in closed:  
            closed.append(node)  
            for successor, move, cost in problem.getSuccessors(node):  
                fringe.push((successor, path + [move], total + [cost]))
```

```
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 ###  
  
Finished at 2:26:52  
  
Provisional grades  
=====  
Question q1: 3/3  
-----  
Total: 3/3  
  
Your grades are NOT yet registered. To register your grades, make sure  
to follow your instructor's guidelines to receive credit on your project.
```

深度优先搜索优先探索离起始节点最远的节点，使用栈（后进先出）来实现，使用 `util.Stack()` 作为 fringe，每次将节点压入栈中。

```
3 usages
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    "*** YOUR CODE HERE ***"

    return general_search(problem, util.Queue(),
        lambda fringe, state, _ : fringe.push(state))
```

广度优先搜索优先探索离起始节点最近的节点，使用队列（先进先出）来实现，每次将节点加入队列末尾。

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:      442.0
Win Rate:    1/1 (1.00)
Record:      Win
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win
```

```

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 ###

Finished at 2:36:26

Provisional grades
=====
Question q2: 3/3
-----
Total: 3/3

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

```

代价一致搜索优先探索代价最小的节点，使用优先级队列来实现，每次将节点按照累计代价压入优先级队列中，代价越小的节点优先出队。

```

def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    "*** YOUR CODE HERE ***"

    return general_search(problem, util.PriorityQueue(),
        lambda fringe, state, cost: fringe.push(state, cost))

```

```

[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:      442.0
Win Rate:    1/1 (1.00)
Record:      Win

```

```

Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:      646.0
Win Rate:    1/1 (1.00)
Record:      Win

```

```

Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:      418.0
Win Rate:    1/1 (1.00)
Record:      Win

```

```

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
### Question q3: 3/3 ###

Finished at 2:46:11

Provisional grades
=====
Question q3: 3/3
-----
Total: 3/3

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

```

A*搜索优先探索估计总成本（路径成本 + 启发式估计）最小的节点，使用优先级队列来实现，每次将节点按照累计代价加上启发式函数值的和压入优先级队列中，总成本越小的节点优先出队。

```

def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    """ YOUR CODE HERE """

    return general_search(problem, util.PriorityQueue(),
        lambda fringe, state, cost: fringe.push(state, cost + heuristic(state[0], problem)))

```

```

[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win

```



```

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

Finished at 3:06:55

Provisional grades
=====
Question q4: 3/3
-----
Total: 3/3

```

(2) 请阅读代码，说明测试命令比如以下测试：

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
```

传入的搜索函数 bfs 是怎么被解析为 search.py 中对应的函数，又是在哪里调用的。

pacman.py 中使用 optparse 模块解析命令行参数。参数 -l mediumMaze 指定了迷宫布局，-p SearchAgent 指定了使用 SearchAgent，-a fn=bfs 指定了搜索函数名为 bfs。解析完参数后，pacman.py 会根据参数创建相应的 Agent 实例。在这个例子中，-p SearchAgent 会创建 SearchAgent 实例。

在 searchAgents.py 中，SearchAgent 的初始化会读取传入的搜索函数名称，并解析成具体的搜索函数

```

def __init__(self, fn='depthFirstSearch', prob='PositionSearchProblem', heuristic='nullHeuristic'):
    # Warning: some advanced Python magic is employed below to find the right functions and problems

    # Get the search function from the name and heuristic
    if fn not in dir(search):
        raise AttributeError(fn + ' is not a search function in search.py.')
    func = getattr(search, fn)
    if 'heuristic' not in func.__code__.co_varnames:
        print('[SearchAgent] using function ' + fn)
        self.searchFunction = func
    else:
        if heuristic in globals().keys():
            heur = globals()[heuristic]
        elif heuristic in dir(search):
            heur = getattr(search, heuristic)
        else:
            raise AttributeError(heuristic + ' is not a function in searchAgents.py or search.py.')
        print('[SearchAgent] using function %s and heuristic %s' % (fn, heuristic))
        # Note: this bit of Python trickery combines the search algorithm and the heuristic
        self.searchFunction = lambda x: func(x, heuristic=heur)

    # Get the search problem type from the name
    if prob not in globals().keys() or not prob.endswith('Problem'):
        raise AttributeError(prob + ' is not a search problem type in SearchAgents.py.')
    self.searchType = globals()[prob]
    print('[SearchAgent] using problem type ' + prob)

```

SearchAgent 的 `__init__` 方法中, 参数 `fn='depthFirstSearch'` 默认使用深度优先搜索。在执行 `python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs` 时, `fn` 参数被设置为 `bfs`。首先, 检查 `bfs` 是否在 `search` 模块的属性列表中。使用 `getattr(search, fn)` 从 `search` 模块中获取名为 `bfs` 的函数, 并将其赋值给 `self.searchFunction`。

```
# Abbreviations
bfs = breadthFirstSearch
dfs = depthFirstSearch
astar = aStarSearch
ucs = uniformCostSearch
```

```
def registerInitialState(self, state):
    """
    This is the first time that the agent sees the layout of the game
    board. Here, we choose a path to the goal. In this phase, the agent
    should compute the path to the goal and store it in a local variable.
    All of the work is done in this method!

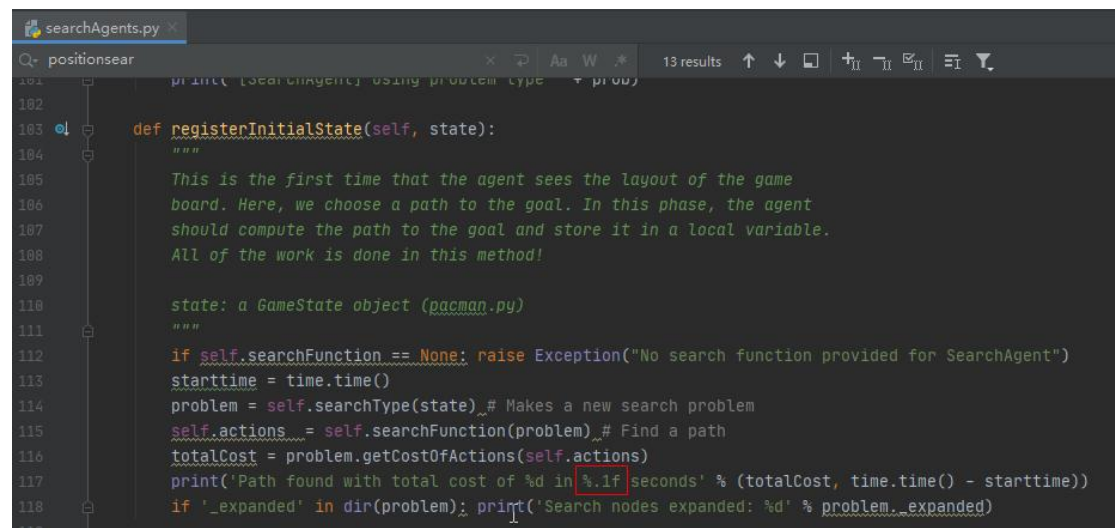
    state: a GameState object (pacman.py)
    """
    if self.searchFunction == None: raise Exception("No search function provided for SearchAgent")
    starttime = time.time()
    problem = self.searchType(state) # Makes a new search problem
    self.actions = self.searchFunction(problem) # Find a path
    totalCost = problem.getCostOfActions(self.actions)
    print('Path found with total cost of %d in %.1f seconds' % (totalCost, time.time() - starttime))
    if '_expanded' in dir(problem): print('Search nodes expanded: %d' % problem._expanded)
```

创建问题实例 `problem`, 使用 `self.searchFunction(problem)` 调用具体的搜索函数, 并获得解决方案 (即行动列表 `self.actions`)。

1.2 实验结果分析

以表格的形式, 列出四种算法在 `mediumMaze` 和 `bigMaze` 两种迷宫上的实验数据, 包括扩展节点数、路径 `cost`、耗时, 并总结对比四种算法在完备性、代价最优性的异同。

关于耗时数据, 需要改动计时的代码精确到小数点后 5 位, 在 `searchAgents.py` 中找到图中所示代码, 将红框中的 `%.1f` 改为 `%.5f`。



```
searchAgents.py
positionsearch
101 print('SearchAgent using problem type: %s' % problem)
102
103 def registerInitialState(self, state):
104     """
105     This is the first time that the agent sees the layout of the game
106     board. Here, we choose a path to the goal. In this phase, the agent
107     should compute the path to the goal and store it in a local variable.
108     All of the work is done in this method!
109
110     state: a GameState object (pacman.py)
111     """
112     if self.searchFunction == None: raise Exception("No search function provided for SearchAgent")
113     starttime = time.time()
114     problem = self.searchType(state) # Makes a new search problem
115     self.actions = self.searchFunction(problem) # Find a path
116     totalCost = problem.getCostOfActions(self.actions)
117     print('Path found with total cost of %d in %.1f seconds' % (totalCost, time.time() - starttime))
118     if '_expanded' in dir(problem): print('Search nodes expanded: %d' % problem._expanded)
119
```

迷宫类型	算法	扩展结点数	路径成本	耗时
mediumMaze	DFS	146	130	0.00400
mediumMaze	BFS	269	68	0.00606
mediumMaze	UCS	249	68	0.00606
mediumMaze	A*	221	68	0.00500
bigMaze	DFS	390	210	0.00700
bigMaze	BFS	620	210	0.01300
bigMaze	UCS	620	210	0.01195
bigMaze	A*	620	210	0.01305

完备性

DFS：不完备，因为它可能会陷入无限的分支而不找到解决方案，尤其在没有循环检测时。

BFS：完备，它会探索所有节点直到找到目标节点，因此总能找到最短路径。

UCS：完备，它总是选择代价最小的路径，最终会找到目标节点。

A*：完备，只要启发式函数是可容许的（不会高估实际成本），则能找到最优解。

代价最优性

DFS：不最优，因为它可能会找到一个较长或较贵的路径而忽略较短或较便宜的路径。

BFS：最优，因为它总是找到节点的最短路径，但仅在所有路径代价相同的情况下有效。

UCS：最优，它总是选择代价最小的路径，因此找到的是最代价最小的路径。

A*：最优，只要启发式函数是可容许的，它会找到代价最小的路径。

一般来说，DFS 不适合需要找到最优路径的情况，但在某些情况下它的搜索速度较快。BFS 适合寻找最短路径的问题，但在大规模问题上可能较慢。UCS 在找到最优路径的同时，处理时间和内存消耗可能较大。A*使用启发式函数优化搜索过程，通常在实际应用中表现最佳。

2、CornersProblem: 问题 5-6

2.1 问题 5 的代码实现和说明

```
class CornersProblem(search.SearchProblem):
    """
    This search problem finds paths through all four corners of a layout.

    You must select a suitable state space and successor function
    """

    def __init__(self, startingGameState):
        """
        Stores the walls, pacman's starting position and corners.
        """
        self.walls = startingGameState.getWalls()
        self.startingPosition = startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners = ((1,1), (1,top), (right, 1), (right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print('Warning: no food in corner ' + str(corner))
        self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
        # Please add any code here which you would like to use
        # in initializing the problem
        "*** YOUR CODE HERE ***"
        self.top = top
        self.right = right
```

```
def getStartState(self):
    """
    Returns the start state (in your state space, not the full Pacman state
    space)
    """
    "*** YOUR CODE HERE ***"
    FourCorners = (False, False, False, False)
    StartState = (self.startingPosition, FourCorners)
    return StartState
```

```
def isGoalState(self, state):
    """
    Returns whether this search state is a goal state of the problem.
    """
    "*** YOUR CODE HERE ***"
    isGoal = state[1] in [(True, True, True, True)]
    return isGoal
```

```

successors = []
for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
    # Add a successor state to the successor list if the action is legal
    # Here's a code snippet for figuring out whether a new position hits a wall:
    # x,y = currentPosition
    # dx, dy = Actions.directionToVector(action)
    # nextx, nexty = int(x + dx), int(y + dy)
    # hitsWall = self.walls[nextx][nexty]

    """ YOUR CODE HERE """
    x, y = state[0]
    Corners = state[1]
    dx, dy = Actions.directionToVector(action)
    nextx, nexty = int(x + dx), int(y + dy)
    hitsWall = self.walls[nextx][nexty]
    nextState = (nextx, nexty)

    if not hitsWall:
        nextCorners = list(Corners)
        if nextState == (1, 1):
            nextCorners[0] = True
        elif nextState == (1, self.top):
            nextCorners[1] = True
        elif nextState == (self.right, 1):
            nextCorners[2] = True
        elif nextState == (self.right, self.top):
            nextCorners[3] = True
        successors.append((nextState, tuple(nextCorners), action, 1))

self._expanded += 1 # DO NOT CHANGE
return successors

```

`__init__` 方法

初始化迷宫的墙壁信息和 Pacman 的起始位置。

获取迷宫的高度和宽度，确定四个角落的位置。

检查每个角落是否有食物，如果没有则发出警告。

初始化了 `_expanded` 变量以记录扩展的节点数。

`getStartState` 方法

返回起始状态，包括 Pacman 的起始位置和一个表示四个角落是否被访问的布尔元组 `FourCorners`，初始时四个角落都未被访问 (`False, False, False, False`)。

`isGoalState` 方法

检查当前状态是否是目标状态，即四个角落都被访问 (`True, True, True, True`)。

`getSuccessors` 方法

生成当前状态的所有合法后继状态。

对于每个可能的移动方向（北、南、东、西），计算新的位置并检查是否碰壁。

如果新位置不是墙壁，则更新访问的角落状态。

将新的状态、相应的行动和成本（1）添加到后继列表中。

更新扩展节点数 `_expanded`。

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:      512.0
Win Rate:    1/1 (1.00)
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:      434.0
Win Rate:    1/1 (1.00)
Record:      Win
```

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

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

Finished at 3:34:59

Provisional grades
=====
Question q2: 3/3
Question q5: 3/3
-----
Total: 6/6

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

2.2 问题 6 的启发函数设计和代码实现

- (1) 请尝试两种不同的启发函数实现
- (2) 给出启发函数可纳性、占优性的证明或说明

```

"""
corners = problem.corners # These are the corner coordinates
walls = problem.walls # These are the walls of the maze, as a Grid (game.py)

*** YOUR CODE HERE ***
position = state[0]
stateCorners = state[1]
corners = problem.corners
node = []
# node列表: 所有未到的角落
for i in range(0, len(corners)):
    if not stateCorners[i]:
        node.append(corners[i])
# 计算一个状态节点的启发值, 类比贪心思想
current_Position = position
cost = 0
# 从输入的位置开始, 计算与node列表的manhattanDistance, 选出manhattanDistance最小的角落
while len(node) > 0:
    # 把该manhattanDistance加进cost中, 并将该角落从node列表删除, 接着以该角落作为输入, 计算与node
    dist = []
    # 一直操作, 直至node列表为空
    for i in node:
        distance = util.manhattanDistance(current_Position, i)
        dist.append(distance)
    minimum_dist = min(dist)
    cost += minimum_dist
    minimum_dist_IN = dist.index(minimum_dist)
    current_Position = node[minimum_dist_IN]
    del node[minimum_dist_IN]
return cost

```

```

9 usages (8 dynamic)
def cornersHeuristic(state, problem):
    """
    A heuristic for the CornersProblem that you defined.

    state:      The current search state
                (a data structure you chose in your search problem)

    problem:    The CornersProblem instance for this layout.

    This function should always return a number that is a lower bound on the
    shortest path from the state to a goal of the problem; i.e. it should be
    admissible (as well as consistent).
    """
    corners = problem.corners # These are the corner coordinates
    walls = problem.walls # These are the walls of the maze, as a Grid (game.py)

    *** YOUR CODE HERE ***
    position = state[0]
    stateCorners = state[1]
    position = state[0]
    stateCorners = state[1]

    # List of unvisited corners
    unvisited_corners = [corners[i] for i in range(len(corners)) if not stateCorners[i]]

    if not unvisited_corners:
        return 0

```

```

(base) PS C:\Users\86156\Desktop\作业\大三下\人工智能\AI-2024\lab1\search\search> python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 692
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores: 434.0
Win Rate: 1/1 (1.00)
Record: Win

```

```

Question q0
=====
*** 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', 'West', 'West', 'West', 'West', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'East',
'East', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'West', 'West', 'East', 'East', 'North', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East',
'East', 'East', 'South', 'South', 'East', 'East', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'East', 'South', 'South', 'South',
'East', 'East', 'North', 'North', 'East', 'East', 'South', 'South', 'South', 'South', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'North', 'North', 'East', 'East', 'North', 'North']
path length: 106
*** PASS: Heuristic resulted in expansion of 692 nodes

### Question q0: 5/5 ###

Finished at 3:37:45

Provisional grades
=====
Question q0: 5/5
Question q0: 5/5
-----
Total: 6/6

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

```

在 `cornersHeuristic` 中，计算了从当前状态到所有未访问角落的曼哈顿距离之和，并且每次都选择距离当前状态最近的角落，逐步更新当前状态。这种方式下，启发函数是可纳的，因为曼哈顿距离的性质：曼哈顿距离是从当前位置到目标位置的一种直线距离（无障碍情况下的直线距离），它不会考虑实际路径上的障碍物。因此，曼哈顿距离总是小于等于真实路径成本。采用贪心算法，每次选择最近的角落进行计算并更新当前状态。这种方式确保了每一步计算都是一种下界估计，即总是考虑到最小的可能路径，而不会高估成本。

因此 `cornersHeuristic` 中每一步计算的曼哈顿距离都是对实际路径的下界估计，因此启发函数是可纳的。

占优性：在 `cornersHeuristic` 中，对于任意的状态转换，启发函数计算的曼哈顿距离也是一致的：

单步转移的曼哈顿距离：从状态 n 到状态 n' 的单步转移的曼哈顿距离是 1。因此，对于任意的状态 n 和 n' ，有 $c(n, n') = 1$

曼哈顿距离的三角不等式性质：曼哈顿距离满足三角不等式，即对于任意的状态 n ， n' ，和目标状态 g ，有： $h(n) \leq c(n, n') + h(n')$

这是因为在曼哈顿距离的计算中，从当前状态到任意未访问角落的距离加上从未访问角落到目标状态的距离不会超过实际路径的距离。

因此可以证明 `cornersHeuristic` 的启发函数是占优的。

2.3 实验结果分析

以表格的形式列出两种启发函数在 `mediumCorners` 上的实验数据，包括扩展节点数、路径 `cost`、耗时等，并说明哪个启发函数更好。

启发函数	扩展节点数	路径成本	耗时
贪心曼哈顿距离	692	106	0.01000
最大曼哈顿距离	1136	106	0.01200

启发函数 2 扩展的节点数显著多于启发函数 1。两种启发函数找到的路径成本是相同的，这表明两者在解决这个问题上都是最优的。启发函数 1 耗时更少，这与其扩展节点数较少是一致的。

因此在 `mediumCorners` 迷宫上，启发函数 1（贪心曼哈顿距离）表现更好。尽管两种启发函数最终找到的路径成本相同，但启发函数 1 扩展的节点数更少且耗时更短。因为不同的迷宫布局可能会对启发函数的表现产生影响。某些迷宫布

局可能更适合贪心曼哈顿距离启发函数，而另一些迷宫布局可能更适合最大曼哈顿距离启发函数。如果最大曼哈顿距离启发函数的计算复杂度比贪心曼哈顿距离启发函数高，那么在某些情况下贪心曼哈顿距离启发函数可能会更快地生成解。

3、FoodSearchProblem: 问题 7-8

3.1 问题 7 的启发函数设计和代码

需给出启发函数可纳性或一致性的证明或说明。

```
position, foodGrid = state
""" YOUR CODE HERE """
foods_position = []
food_mutual_dist = ((0, 0), (0, 0), 0)

# 获取豆子的坐标
for i in range(0, foodGrid.width):
    for j in range(0, foodGrid.height):
        if (foodGrid[i][j] == True):
            foods_position.append((i, j))

# 豆子已经被吃完
if (len(foods_position) == 0):
    return 0

# 还有豆子，则找到manhattanDistance最远的两个豆子，作为一个豆子组
for current_food in foods_position:
    for select_food in foods_position:
        if (current_food != select_food):
            distance = util.manhattanDistance(current_food, select_food)
            if (distance > food_mutual_dist[2]):
                food_mutual_dist = (current_food, select_food, distance)

# 以当前位置到该豆子组的manhattanDistance的较小值加上与该豆子组的manhattanDistance作为启发函数
if (food_mutual_dist[0] == (0, 0) and food_mutual_dist[1] == (0, 0)):
    result = util.manhattanDistance(position, foods_position[0])
else:
    dist1 = util.manhattanDistance(position, food_mutual_dist[0])
    dist2 = util.manhattanDistance(position, food_mutual_dist[1])
    result = min(dist1, dist2) + food_mutual_dist[2]

return result
```

```
(base) PS C:\Users\86156\Desktop\作业\大三下\人工智能\AI-2024\AI-2024\lab1\search\search> python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 1.2 seconds
Search nodes expanded: 7553
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:      570.0
Win Rate:    1/1 (1.00)
Record:      Win
```



```

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
*** FAIL: test_cases\q7\food_heuristic_grade_tricky.test
***     expanded nodes: 7553
***     thresholds: [15000, 12000, 9000, 7000]

### Question q7: 4/4 ###

Finished at 3:41:28

Provisional grades
=====
Question q4: 3/3
Question q7: 4/4
-----

```

获取所有未吃的食物位置：我们首先将食物网格中的所有食物位置存入一个列表 `foods_position`。找到两个最远的食物位置：通过计算所有食物位置对之间的曼哈顿距离，找到两个最远的食物位置，存入 `food_mutual_dist`。

计算启发值：启发值为当前状态位置到这两个最远食物位置中的较小曼哈顿距离，加上这两个食物位置间的曼哈顿距离。

启发函数的可纳性证明：

曼哈顿距离的性质：曼哈顿距离是从当前位置到目标位置的一种直线距离（无障碍情况下的直线距离），它不会考虑实际路径上的障碍物。因此，曼哈顿距离总是小于等于真实路径成本。最远食物位置对的选择：我们选择了两个最远的食物位置，并计算当前状态位置到这两个食物位置的较小曼哈顿距离。这个距离是从当前状态到食物位置的下界估计。

启发值的计算：我们将当前状态位置到最远食物位置对的较小曼哈顿距离，加上这两个食物位置间的曼哈顿距离，作为启发值。这确保了启发值是从当前状态到目标状态的下界估计。

因此，`foodHeuristic` 的启发函数是可纳的。

在 `foodHeuristic` 中，对于任意的状态转换，启发函数计算的曼哈顿距离也是一致的：单步转移的曼哈顿距离：从状态 n 到状态 n' 的单步转移的曼哈顿距离是 1。因此，对于任意的状态 n 和 n' ，有 $c(n, n') = 1$ 。

曼哈顿距离的三角不等式性质：曼哈顿距离满足三角不等式，即对于任意的状态 n ， n' ，和目标状态 g ，有： $h(n) \leq c(n, n') + h(n')$ 这是因为在曼哈顿距离的计算中，从当前状态到任意食物位置的距离加上从该食物位置到目标状态的距离不会超过实际路径的距离。

可以证明 `foodHeuristic` 的启发函数是占优的。

3.2 问题 8 的代码实现和说明

```
2 usages (1 dynamic)
def findPathToClosestDot(self, gameState):
    """
    Returns a path (a list of actions) to the closest dot, starting from
    gameState.
    """
    # Here are some useful elements of the startState
    startPosition = gameState.getPacmanPosition()
    food = gameState.getFood()
    walls = gameState.getWalls()
    problem = AnyFoodSearchProblem(gameState)

    """ YOUR CODE HERE """
    return search.breadthFirstSearch(problem)
```

```
2 usages (2 dynamic)
def isGoalState(self, state):
    """
    The state is Pacman's position. Fill this in with a goal test that will
    complete the problem definition.
    """
    x,y = state

    """ YOUR CODE HERE """
    return (True if (self.food[x][y]) else False)
```

在上述代码中 ClosestDotSearchAgent 搜索代理，它的任务是找到吃豆人（Pacman）到最近食物点的路径。这个代理使用广度优先搜索（BFS）算法来解决这个问题。findPathToClosestDot(self, gameState)：该方法的目的是找到从当前状态到最近食物点的路径。首先获取当前吃豆人位置、食物分布和墙壁位置。创建一个 AnyFoodSearchProblem 实例，并调用 search.breadthFirstSearch 方法执行搜索，返回路径。在每个循环中，调用 findPathToClosestDot 方法，创建一个 AnyFoodSearchProblem 实例，并使用 BFS 找到到最近食物点的路径。将路径上的动作追加到 self.actions 中，并更新 currentState。

路径验证：对每个动作进行合法性验证，确保返回的路径是合法的。

终止条件：当所有食物都被吃完后，打印总路径成本并结束循环。

```
(base) PS C:\Users\86156\Desktop\作业\大三下\人工智能\AI-2024\lab1\search\search> python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:      2360.0
Win Rate:    1/1 (1.00)
Record:      Win
```

```
***      solution length:          1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_6.test
***      pacman layout:           Test 6
***      solution length:          2
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_7.test
***      pacman layout:           Test 7
***      solution length:          1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_8.test
***      pacman layout:           Test 8
***      solution length:          1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_9.test
***      pacman layout:           Test 9
***      solution length:          1

### Question q8: 3/3 ###

Finished at 3:44:47

Provisional grades
=====
Question q8: 3/3
-----
Total: 3/3
```

4、总结

通过这次实验，我对不同搜索算法在吃豆人迷宫问题中的应用和性能有了深入的理解。选择合适的搜索算法应根据具体问题的需求，如是否需要最优解、状态空间的大小和启发函数的有效性等因素进行综合考虑。