同济大学计算机系

人工智能课程设计实验报告

学　　号 2152809

姓　　名 曾崇然

专　　业 计算机科学与技术

授课老师 武妍老师

一. 问题概述

1. 问题的直观描述

总体：在一个吃豆人游戏中，通过补充各种关于搜索的算法和游戏模式的相关内容，帮助糖豆人有效率的到达某个地方或者吃掉豆子。

具体：

问题 1：实现一个深度优先搜索算法帮助糖豆人从开始位置到达指定的位置

问题 2：实现一个宽度优先搜索算法帮助糖豆人从开始位置到达指定的位置

问题 3：实现一个一致代价搜索算法帮助糖豆人从开始位置到达指定的位置

问题 4：实现一个 A*搜索算法帮助糖豆人从开始位置到达指定的位置

问题 5：完善另一种游戏模式，在这种模式中糖豆人能够利用问题 2 中实现的宽度优先搜索算法使得糖豆人用最短的路径将迷宫的四个角落都经过一遍

问题 6：为问题 5 中实现的游戏模式设计一个启发式函数，再利用问题 4 中实现的 A*搜索算法帮助糖豆人更加高效的获得遍历迷宫四个角落的最短途径

问题 7：现有另一种游戏模式，在该模式中糖豆人要实现吃掉所有迷宫中食物的目标，要求为这种游戏模式设计一个启发式函数，并利用问题 4 中实现的 A*搜索算法高效的找到达成目标的最优途径

问题 8：为问题 7 中的游戏模式设计一个贪婪搜索算法，使糖豆人总是以最近的一个食物为目标前进直到吃掉所有的食物

2. 已有代码的阅读理解

searchAgent.py:这个文件里包含了游戏各种模式对应的类和类的实现，以及辅助实现这些类的一些函数和类

class SearchAgent(Agent):使用指定的搜索算法解决一个指定的搜索问题

class PositionSearchProblem(search.SearchProblem):包含从起始点到终点的游戏模式所需的各种属性和函数

manhattanHeuristic(position, problem, info={}):实现游戏模式 1 的一个启发式函数

euclideanHeuristic(position, problem, info={}):实现游戏模式 1 的另一个启发式函数

class CornersProblem(search.SearchProblem): 包含遍历迷宫四个顶点的游戏模式所需的各种属性和函数

cornersHeuristic(state: Any, problem: CornersProblem): 实现游戏模式 2 的一个启发式函数

class FoodSearchProblem: 包含吃掉迷宫里所有食物的游戏模式所需的各种属性和函数

foodHeuristic(state: Tuple[Tuple, List[List]], problem: FoodSearchProblem): 实现游戏模式 2 的一个启发式函数

class ClosestDotSearchAgent(SearchAgent):游戏模式 3 的贪婪搜索算法

class AnyFoodSearchProblem(PositionSearchProblem): 包含吃掉迷宫里任何指定食物的游戏模式所需的各种属性和函数

mazeDistance(point1: Tuple[int, int], point2: Tuple[int, int], gameState: pacman.GameState) -> int:计算任意两个点之间的距离

search.py:包含完成各种游戏模式的搜索算法

depthFirstSearch(problem: SearchProblem):深度优先搜索算法

breadthFirstSearch(problem: SearchProblem):宽度优先搜索算法

uniformCostSearch(problem: SearchProblem):一致代价搜索算法

aStarSearch(problem: SearchProblem, heuristic=nullHeuristic):A*搜索算法

util.py:包含了实现各种搜索算法可能用到的几种数据结构

class Stack:包含栈压入弹出判断为空的函数

class Queue:包含队列压入弹出判断为空的函数

class PriorityQueue:包含优先队列的压入弹出更新判断为空的函数

3. 解决问题的思路和想法

问题 1：通过状态的入栈和出栈来实现深度优先搜索的扩展顺序

问题 2：通过状态的入队和出队来实现广度优先搜索的扩展顺序

问题 3：将到达某个状态的路径长度作为权值来将某个状态压入优先队列，再利用优先队列的弹出来实现一致代价搜索的扩展顺序

问题 4：将到达某个状态的路径长度加上启发函数值作为权值来将某个状态压入优先队列，再利用优先队列的弹出来实现一致代价搜索的扩展顺序

问题 5：模仿 PositionSearchProblem 完善 CornersProblem 的函数，需要将位置和四个角落的状态同时作为状态空间来进行搜索

问题 6：将四个角落里和当前位置的最大曼哈顿距离作为启发函数值

问题 7：将最远食物和最近食物的曼哈顿距离和最近食物和当前位置的曼哈顿距离作为启发函数值

问题 8：将最近食物作为目标调用搜索算法即可

二. 算法设计

问题 1：

算法功能：根据深度优先搜索的顺序对吃豆人的状态空间进行转移

设计思路：分析发现深度优先搜索的顺序和栈的压入弹出的顺序之间存在着某种联系，因此采用栈的压入弹出来实现深度优先搜索，将起始状态压入栈，之后循环操作弹出，将弹出状态标记为已扩展，将弹出状态的可扩展状态压入栈，直到栈为空或者弹出状态为所寻找状态为止，若为空则返回 None，若不为空，则返回记录下来的路径

问题 2：

算法功能：根据宽度优先搜索的顺序对吃豆人的状态空间进行转移

设计思路：分析发现广度优先搜索的顺序和队列的入队和出队顺序之间存在着一定联系，不停的将出队的状态标记为已扩展，同时将该状态的可扩展状态入队，直到对空或者找到目标状态为止，状态出队的顺序就是宽度优先搜索的顺序，在找到目标状态后返回记录下来的对应路径即可

问题 3：

算法功能：根据一致代价搜索的顺序对吃豆人的状态空间进行转移

设计思路：一致代价搜索和宽度优先搜索有着很紧密的联系，宽度优先搜索可以看作是路径耗散均为 1 的一致代价搜索；一致代价搜索的设计就是在宽度优先的基础上将路径的耗散作为扩展的顺序，优先队列可以实现这个功能。将宽度优先搜索入队时的操作改为优先队列以到该状态的路径值为权值的入队即可实现一致代价搜索

问题 4：

算法功能：根据一个给定的启发函数，根据对应的 A*搜索的顺序对吃豆人的状态空间进行转移

设计思路：一致代价搜索是将路径耗散作为权值，而 A*搜索是将启发式函数值和路径耗散的和作为扩展的依据，因此将一致代价搜索中的路径耗散作为权值改为路径耗散和启发函数的值的和作为权值来进行优先队列的入队操作就能实现 A*搜索算法

问题 5：

算法功能：为之前完成的几种宽度优先算法实现一个新的游戏模式：要遍历过迷宫的四个顶点才算完成目标

设计思路：因为在遍历四个顶点的过程中，几乎不可避免的会出现前后经过相同位置的情况，

因此如果再将位置作为状态空间，则不好处理是否已经扩展的问题，另外由于游戏是否完成还取决于四个角落是否被经过，因此考虑将四个角落是否被经历过作为状态的一部分，既利于判断是否到达结束状态，又巧妙的避开了状态空间重复的问题。之后利用宽度优先搜索就能完成遍历四个顶点的目标了

问题 6：

算法功能：为 Corners Problem 设计一个启发式函数，使之能够更高效的扩展到结束状态

设计思路：已知在上一个使用 A*搜索的问题中使用曼哈顿距离作为启发函数值是可接受和一致的，因此这里模仿上一题的思路，采用四个顶点中曼哈顿距离最大的作为启发式函数的值

问题 7：

算法功能：为 Eating All The Dots 问题设计一个启发式函数

设计思路：可以大致将吃豆人吃完所有食物的过程分为两个部分，一是吃掉最近的食物，二是吃掉最近和最远之间的食物，所以将当前位置到最近食物的曼哈顿距离和最近食物和最远食物的曼哈顿距离的和作为启发函数的值，这样既考虑了最近的食物（一个大致走向），又合理的估计了到终点的路程

问题 8：

算法功能：实现总是依次吃掉距离吃豆人最近食物的行走顺序

设计思路：利用给出函数找出距离最近的食物，将其作为目标，使用搜索算法将其吃掉，循环直到食物被吃完

三. 算法实现

问题 1：

代码：

```
def depthFirstSearch(problem: SearchProblem):
            from game import Directions
    S=util.Stack()
    parent={}
    visited={}
    init=problem.getStartState()
    p=(init,Directions.STOP,0)
    S.push(p)
    parent[p[0]]=None
    while 1:
        if S.isEmpty():
```

```
            return None
        p=S.pop()
        if parent[p[0]]==None:
            visited[p[0]]=[]
        else:
            midlist=visited[parent[p[0]]].copy()
            midlist.append(p[1])
            visited[p[0]]=midlist#标记为访问同时记录路径
        if problem.isGoalState(p[0]):
            break
        successors=problem.getSuccessors(p[0])
        for x in successors:
            if x[0] in visited:
                continue
            else:
                S.push(x)
                parent[x[0]]=p[0]
    return visited[p[0]]
```

数据结构定义：栈，使用栈的压入弹出操作来实现深度优先搜索的顺序

实现细节：我使用了一个 visited 字典来存放弹出（已扩展）状态对应的路径，每次弹出时根据上一个状态的路径和到达当前状态的方向获得到达当前状态的路径，在搜索到终点状态时便于返回

问题 2：

代码：

```
def breadthFirstSearch(problem: SearchProblem):
        from game import Directions

    visited=[]
    Q=util.Queue()
    start=problem.getStartState()
    Q.push((start,[]))
    while 1:
        if Q.isEmpty():
            return None
        state,action=Q.pop()
        if problem.isGoalState(state):
            break
        if state not in visited:#屏蔽掉已经访问过的
            visited.append(state)
            successors=problem.getSuccessors(state)
            for x in successors:
```

```
                if x[0] in visited:
                    continue
                else:
                    Q.push((x[0],action+[x[1]]))
    return action
```

数据结构定义：队列，使用队列的入队和出队来实现宽度优先扩展的顺序

实现细节：①在实现深度优先搜索时，我感觉到将路径存储在一个单独的字典中有些不方便，其在与总是要去寻找器父状态的路径，因此我在宽度优先搜索里将路径同状态一起入队和出队，这样就能很方便的得到父状态的路径②在弹出时判断是否已经扩展过了，这是至关重要的，因为在弹出和选择可扩展状态时，并不能判断可扩展状态是否已经在队列里了，因此队列里可能出现两个相同的状态，所以需要在扩展时进行判断，是否已经扩展过了

问题 3：

代码：

```
def uniformCostSearch(problem: SearchProblem):
        from game import Directions

    P=util.PriorityQueue()
    store={}
    visited={}
    parent={}
    init=problem.getStartState()
    parent[init]=None
    P.push(init,0)
    store[init]=[Directions.STOP,0]
    while 1:
        if P.isEmpty():
            return None
        loc=P.pop()
        if parent[loc]==None:
            visited[loc]=[]
        else:
            midlist=visited[parent[loc]].copy()
            midlist.append(store[loc][0])
            visited[loc]=midlist
        if problem.isGoalState(loc):
            break
        successors = problem.getSuccessors(loc)
        for x in successors:
            if x[0] in visited:
                continue
```

```
            else:
                P.update(x[0], x[2]+store[loc][1])
                if x[0] in store:
                    if x[2]+store[loc][1]<store[x[0]][1]:
                        store[x[0]][1]=x[2]+store[loc][1]
                        store[x[0]][0]=x[1]
                        parent[x[0]]=loc
                else:
                    store[x[0]]=[Directions.STOP,0]
                    store[x[0]][1] = x[2] + store[loc][1]
                    store[x[0]][0] = x[1]
                    parent[x[0]] = loc#存储到达该状态的耗散
    return visited[loc]
```

数据结构设计：优先队列，一致代价搜索需要在宽度优先搜索的基础上根据状态的路径耗散进行有顺序的扩展，因此使用优先队列来实现这一功能

实现细节：①注意到优先队列当中有一个 update 函数，能够对根据不同的情况对队列进行更新，不用在队列里放重复的状态和判断是否重复解决掉重复状态入队的问题，因此在这个问题中我尝试使用②为了使优先队列判断是否两个状态是重复的，我只将位置放入优先队列中（如果放入方向和耗散则无法判断）用单独的字典对路径，耗散，方向进行存储

问题 4：

代码：

```
def aStarSearch(problem: SearchProblem, heuristic=nullHeuristic):
        visited = []
    Q = util.PriorityQueue()
    start = problem.getStartState()
    Q.push((start, []),0)
    while 1:
        if Q.isEmpty():
            return None
        state, action = Q.pop()
        if problem.isGoalState(state):
            break
        if state not in visited:
            visited.append(state)
            successors = problem.getSuccessors(state)
            for x in successors:
                if x[0] in visited:
                    continue
                else:
                    F=heuristic(x[0],
```

```
problem)+problem.getCostOfActions(action + [x[1]])#将耗散值和估计距离的和作为权值
                    Q.push((x[0], action + [x[1]]),F)
    return action
```

数据结构设计：因为 A*搜索只是将权值从一致代价搜索的路径耗散转变为了路径耗散和启发函数值的和，所以基于和一致代价搜索算法同样的考虑，这里同样使用优先队列作为主句结构

实现细节：①在尝试使用了优先队列的 update 函数来解决状态重复的问题时，仍然感到有许多不方便的地方，比如需要寻找父状态，得到路径比较繁琐等问题，因此在写 A*算法时，在宽度优先搜索的基础上修改，使用判断是否扩展的方法来解决（允许队列里的状态重复）②将耗散值和启发函数的和作为权值入队

问题 5：

代码：

```
def getStartState(self):
    corners=[False,False,False,False]
    return (self.startingPosition,corners)#将位置和四个角落是否被访问作为状态

def isGoalState(self, state: Any):
        if state[1][0] and state[1][1] and state[1][2] and state[1][3]:#四个顶点均访问过则完成
        return True
    else:
        return False


def getSuccessors(self, state: Any):
    successors = []
    for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
        x, y = state[0]
        corners=state[1]
        nextcorners=()
        dx, dy = Actions.directionToVector(action)
        nextx, nexty = int(x + dx), int(y + dy)
        if not self.walls[nextx][nexty]:#是否到达角落
            if (nextx,nexty) in self.corners:
                if (nextx,nexty)==self.corners[0]:

nextcorners=(True,corners[1],corners[2],corners[3])
```

```
                elif (nextx,nexty)==self.corners[1]:

nextcorners=(corners[0],True,corners[2],corners[3])
                elif (nextx,nexty)==self.corners[2]:

nextcorners=(corners[0],corners[1],True,corners[3])
                elif (nextx,nexty)==self.corners[3]:

nextcorners=(corners[0],corners[1],corners[2],True)
                nextState = ((nextx, nexty),nextcorners)
            else:
                nextState = ((nextx, nexty), corners)
            cost = 1
            successors.append((nextState, action, cost))
    self._expanded += 1 # DO NOT CHANGE
    # if state not in self._visited:
    #     self._visited[state] = True
    #     self._visitedlist.append(state)
    return successors
```

设计细节：不能按照上一个游戏模式里那样将吃豆人的位置作为状态，这样无法处理重复经过某个位置的情况，因此将四个角落是否被访问过野作为状态，这样既避免状态重复，又能够很方便的判断是否到达结束状态；设计好后，按照类似上一题的方法调用已经完成的宽度优先搜索就能看到四个角落被以最优路径访问

问题 6：

代码：

```
def cornersHeuristic(state: Any, problem: CornersProblem):
    corners = problem.corners # These are the corner coordinates
    walls = problem.walls # These are the walls of the maze, as a
Grid (game.py)

    xy = state[0]
    list = state[1]
    S=0
    if list[0]==False:
        mid=(abs(xy[0] - corners[0][0]) + abs(xy[1] - corners[0][1]))
        if S<mid:
            S=mid
    if list[1] == False:
        mid = (abs(xy[0] - corners[1][0]) + abs(xy[1] -
corners[1][1]))
        if S < mid:
```

```
            S = mid
    if list[2] == False:
        mid = (abs(xy[0] - corners[2][0]) + abs(xy[1] -
corners[2][1]))
        if S < mid:
            S = mid
    if list[3] == False:
        mid = (abs(xy[0] - corners[3][0]) + abs(xy[1] -
corners[3][1]))
        if S < mid:
            S = mid#将四个角落距离当前位置的曼哈顿距离最大值作为启发
式函数的值

    return S # Default to trivial solution
```

设计细节：采用四个顶点中曼哈顿距离最大的作为启发式函数的值，既能够保证该启发式函数是可接受的并且具有一致性，又能对距离进行一个合理的估计

问题 7：

代码：

```
def foodHeuristic(state: Tuple[Tuple, List[List]], problem:
FoodSearchProblem):
    position, foodGrid = state
    count=0
    Food = state[1].copy()
    loc = state[0]
    max=0
    max_loc=[0,0]
    min_loc=[0,0]
    min=100000
    h=Food.height
    w=Food.width
    for i in range(0,w):
        for j in range(0,h):
            if Food[i][j]==True:
                count=count+1
                mid=abs(i-loc[0])+abs(j-loc[1])
                if mid>max:
                    max=mid
                    max_loc[0]=i
                    max_loc[1]=j
                if mid<min:
                    min=mid
                    min_loc[0] = i
```

```
                    min_loc[1] = j
    if count==0:
        return 0
    elif count==1:
        D=abs(max_loc[0]-loc[0])+abs(max_loc[1]-loc[1])
    else:
        D = abs(min_loc[0] - loc[0]) + abs(min_loc[1] -
loc[1])+abs(min_loc[0] - max_loc[0]) + abs(min_loc[1] - max_loc[1])

    return D
```

设计细节：将当前位置到最近食物的曼哈顿距离和最近食物和最远食物的曼哈顿距离的和作为启发函数的值，我认为这有几个好处，使其能够作为该问题的一个合适的启发式函数：①它对当前状态到终点状态的距离估计是合理的，符合常识：吃豆人先到最近的食物处，吃掉食物，再沿着食物的分布一直吃到最远的食物②保证其是可接受的，这两个曼哈顿距离之和一定是小于等于所需到达终点的距离的，可以很容易的证明③这个启发式考虑了最近的食物，如果只考虑最远，可能会出现近在眼前的食物周转很大才被扩展的情况，降低效率

问题 8：

代码：

```
def findPathToClosestDot(self, gameState: pacman.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.bfs(problem)
def isGoalState(self, state: Tuple[int, int]):
    """
    The state is Pacman's position. Fill this in with a goal test
that will
    complete the problem definition.
    """
    x,y = state
    if self.food[x][y]:
        return True
    else:
        return False
    "*** YOUR CODE HERE ***"
```

实现细节：①由于是任意食物，所以判断的标准是是否在食物里②在 findPathToClosestDot

函数中调用即可不断吃掉最近的食物

四． 实验结果

问题 1：

结果展示：

```
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 21:04:03

Provisional grades
==================
Question q1: 3/3
------------------
Total: 3/3
```

描述说明：

在这几个测试用例中，前一项指标是指得出的路径（深度优先遍历未必最优），比如第一个测试的结果指从 A 到 C 再到 G

第二项指标指扩展的的状态，扩展的状态越少效率越高，比如第一个测试就扩展了 ADC 三个状态

问题 2：

结果展示：

```
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 21:06:47

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

描述说明：和深度优先搜索一样，第一项指标表示得出的结果路径，第二项表示扩展的结点，可以看出在结果相同的情况下，宽度优先搜索的扩展结点更多。

但是宽度优先搜索能够保证找到最优解

问题 3：

结果展示：

```
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
*** PASS: test_cases\q3\ucs_3_problemW.test
***     pacman layout:          mediumMaze
***     solution length: 152
***     nodes expanded:         173
*** PASS: test_cases\q3\ucs_4_testSearch.test
***     pacman layout:          testSearch
***     solution length: 7
***     nodes expanded:         14
*** PASS: test_cases\q3\ucs_5_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']

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


Finished at 21:07:21

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

描述说明：

指标的含义同上，不过一致代价搜索的扩展时根据路径的耗散值进行选择的

问题 4：

结果展示：

```
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 ###


Finished at 21:07:48

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

描述说明：指标的含义同上，不过 A*搜索的扩展时根据路径的耗散值和启发式函数的值的和进行选择的

问题 5：

结果展示：

```
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 ###


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

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


Finished at 21:08:18

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

描述说明：前两项指标的含义同上，pacman layout：表示地图的类型，solution length：表示所走过的路径长度；该题需要将地图的四个顶点以最短路径遍历完

问题 6：

结果展示：

```
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 ###


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', 'West', 'West', 'West', 'West', 'West',
 'West', 'South', 'South', 'South', 'West', 'West', 'North', 'East', 'East', 'North', 'North', 'North',
North', 'East', 'East', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'We
st', 'South', 'South', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'Sout
h', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', 'East', 'East', 'Ea
st', 'North', 'North', 'East', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East'
, 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'North', 'North', 'East', 'East', '
South', 'South', 'South', 'South', 'South', 'North', 'North', 'North', 'North', 'North', 'North', 'North'
, 'West', 'West', 'North', 'North', 'East', 'East', 'North', 'North']
path length: 106
*** PASS: Heuristic resulted in expansion of 1136 nodes

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

描述说明：

heuristic value less than true cost at start state：这句话表明在开始状态时进行验证，启发式函数可接受

path:[……]：这是搜索得出的最优路径

PASS: Heuristic resulted in expansion of 1136 nodes：这句话指使用启发式函数后扩展了 1136 个状态，A*搜索相比于宽度优先搜索扩展的结点更少，效率更高

问题 7：

结果展示：

```
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: 8617
***     thresholds: [15000, 12000, 9000, 7000]

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


Finished at 21:09:05

Provisional grades
==================
Question q4: 3/3
Question q7: 4/4
------------------
Total: 7/7
```

描述说明：

expanded nodes: 8617：这句话表明在使用我写的启发式函数的情况下，扩展的状态数量为 8617 个

问题 8：

结果展示：

```
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
***     pacman layout:          Test 13
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_2.test
***     pacman layout:          Test 2
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_3.test
***     pacman layout:          Test 3
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_4.test
***     pacman layout:          Test 4
***     solution length:                3
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_5.test
***     pacman layout:          Test 5
***     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
```

```
*** PASS: test_cases\q8\closest_dot_2.test
***     pacman layout:          Test 2
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_3.test
***     pacman layout:          Test 3
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_4.test
***     pacman layout:          Test 4
***     solution length:                3
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_5.test
***     pacman layout:          Test 5
***     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 ###
```

描述说明：solution length 依次描述了各个问题解决的路径的长度

五.　总结分析

1.　搜索算法的选择

在解决寻路问题时，选择合适的搜索算法非常重要。深度优先搜索和宽度优先搜索是最基本的搜索算法，它们分别具有深度优先和广度优先的特点。一致代价搜索和 A*搜索是更高级的搜索算法，它们可以使用启发式函数（一致代价搜索看作启发函数值为 0）来指导搜索方向。在实际应用中，要根据具体问题的特点来选择合适的搜索算法。

2.　启发式函数的设计

启发式函数可以提供搜索方向的指引，对搜索效率有重要的影响。设计好的启发式函数应该尽可能地接近实际问题的情况，可以根据问题的不同特点来设计不同的启发式函数。

3. 实现算法的细节

在实现算法时，需要注意算法的细节。例如，在一致代价搜索和 A*搜索中，需要给每个状态赋一个代价值，并在更新代价值时进行优化。还需要注意避免重复搜索同一个状态，以及避免搜索无效的状态。

4. 算法的优缺点

不同的搜索算法具有不同的优缺点。例如，深度优先搜索和宽度优先搜索在空间和时间上的复杂度差异很大，一致代价搜索和 A*搜索需要选择合适的启发式函数来达到最优解。

总之，这个实验让我深入了解了搜索算法和启发式函数的基本原理和应用，了解了如何将这些算法和函数应用到寻路问题中。通过实现和调试这些算法，我更好地理解了这些算法的优缺点，掌握了这些算法实现的细节，并提高了自己的算法设计能力。