

**人工智能实验报告**

题 目 搜索策略

专 业 计算机科学与技术

学　　 号 1171000410

学 生 强文杰

指 导 教 师 李钦策

同 组 人 员 陈鋆、张亚博、束魏琦、王家琪

1. **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。

问题1：应用深度优先算法找到一个特定的位置的豆；

问题2：宽度优先算法；

问题3：代价一致算法；

问题4：A\*算法；

问题5：找到所有的角落；

问题6：角落问题（启发式）；

问题7：吃掉所有的豆子；

问题8：次最优搜索。

1.2 问题的形式化描述

给定一个迷宫，问题1-4分别要求完成深度优先搜索算法、广度优先搜索算法、代价一致搜索算法和A\*算法以帮助吃豆人规划一条合适的路径进行吃豆；问题5要求在四个角上有四个豆的迷宫中找到一条访问所有四个角落的最短的路径；问题6要求构建合适的启发函数以完成角落搜索问题；问题7要求用尽可能少的步数吃掉所有的豆子；问题8要求定义一个优先吃最近的豆子函数。

1.3 解决方案介绍（原理）

·问题1：深度优先搜索算法。

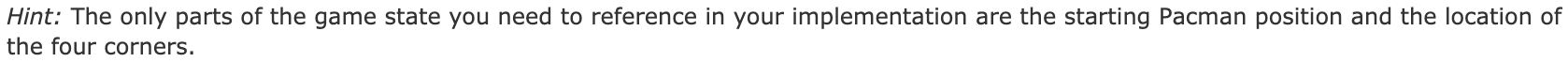
·问题2：广度优先搜索算法。

·问题3：代价一致搜索算法。

·问题4：A\*算法。

·问题5：

根据网站上的提示：



用二元组(currentPosition, list)来表示该问题中吃豆人的状态,其中currentPosition为吃豆人当前所在迷宫的位置，list存储吃豆人在当前位置时已访问的角落。每到达一个角落就将该角落添加进list中，则该问题的目标状态就是吃豆人已访问所有四个角落（list中保存了所有四个角落）。故起始状态可表示为(self.startingPosition, [])，getStartState(self)方法返回该起始状态即可，isGoalState(self, state)方法只需判断state状态二元组中的list是否保存了所有四个角落即可，getSuccessors(self, state)方法与PositionSearchProblem中的类似，只不过后继状态用上述定义的二元组来表示。

·问题6：

首先统计吃豆人在当前位置时还未到达的那些角落，由于这些角落的位置固定且数量较少，可以通过广度优先搜索枚举吃豆人从当前位置遍历剩余未访问角落的所有可能路径，利用searchAgents.py文件中提供的mazeDistance方法计算这些路径的真实代价，从中选出最小的路径代价作为当前位置的启发式代价。该代价几乎等于真实代价，因此包含较多的启发式信息，使得算法拓展的节点数量非常少。

·问题7：

当启发式函数越接近真实最优代价时，包含的启发式信息越多，效果越好，但得到该启发式函数所需要的代价越大；当启发式函数越远离真实最优代价时，包含的启发式信息越少，效果越差。由于计算吃豆人吃掉所有食物的启发式路径距离是一件代价很大的事，故考虑从所有剩余食物中选择具有代表性的食物，计算吃豆人到该食物的距离并以此为吃豆人完成目标所需要的最优代价下界。如果选择距吃豆人最近的食物，类似于贪心思想，则容易对某个食物的临近位置进行过度探索从而局限在迷宫的某个局部，没有考虑全局（局部最优不一定得到全局最优）。故考虑使用从吃豆人当前位置到距其最远豆子的距离来作为当前位置的启发式距离，这样可让吃豆人更倾向于扩展到剩余食物的中心位置，使得吃豆人距离剩余食物中的每一个都不至于很远。如果使用曼哈顿距离来计算这个启发式代价，效果不太理想（扩展的节点数量仍旧较多），因为曼哈顿距离比真实最优代价小太多，于是可采用searchAgents.py文件中提供的mazeDistance方法，该方法能计算从吃豆人当前位置到最远食物位置的真实距离，比曼哈顿距离大，是一个更紧的下界，效果更好。

·问题8：

在AnyFoodSearchProblem中，判断当前状态是否为目标状态只需判断吃豆人当前位置上是否有食物即可，如果有食物则为目标状态，反之。

在ClosestDotSearchAgent中，为找到一条到达最近食物的路径，只需利用广度优先搜索算法解决AnyFoodSearchProblem即可。因为广度优先搜索算法一定优先遍历距当前吃豆人位置最近的位置，而且该算法得出的当前吃豆人位置到其他每个位置的路径一定是到该位置的最短路径，故当算法第一次搜索到有食物的位置时，该食物一定是距当前吃豆人位置最近的。

1. **算法介绍**

2.1 所用方法的一般介绍

·深度优先搜索：

在深度优先搜索中，首先扩展最新产生的(即最深的)节点，是后生成的节点先扩展的策略。从初始节点开始扩展，若没有得到目标节点，则选择最新产生的子节点进行扩展，若还是不能到达目标节点，则再对刚才最新产生的子节点进行扩展，一直如此向下搜索。当到达某个子节点，且该子节点既不是目标节点又不能继续扩展时，才选择其兄弟节点进行考察。Open表是一种栈结构，最先进进入的节点排在最后面，最后进入的节点排最前面。

算法步骤如下：

(1) 把初始节点放入Open表，建立一个Closed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) 若节点n不可扩展，则转第（2）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（2）步。

深度优先搜索不一定能找到解，更不一定能找到最优解（在本问题中能找到）。按树形结构进行扩展，当每个父节点拥有b个子节点、树的深度为m层时，深度优先搜索扩展的节点数将达到个，Open表中存放的节点数有个。

·广度优先搜索：

又称为宽度优先搜索，是一种先生成的节点先扩展的策略。宽度优先搜索的基本思想是：从初始节点开始，逐层地对节点进行扩展并考察它是否为目标节点，在第n层的节点没有全部扩展并考察之前，不对第n＋1层的节点进行扩展。Open表是一种队列结构，表中的节点总是按进入的先后顺序排列，先进入的节点排在前面，后进入的排在后面，即先进先出（FIFO）。

算法步骤如下：

(1) 把初始节点放入Open表，建立一个Closed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) 若节点n不可扩展，则转第（2）步；

(6) 扩展节点n，将其子节点放入Open表的尾部，并为每个子节点设置指向父节点的指针, 转向第（2）步。

广度优先搜索一定能找到解且找到的一定是最优解。按树形结构进行扩展，当每个父节点拥有b个子节点、解位于树的第m层时，广度优先搜索扩展的节点数将达到个，Open表中存放的节点数也将达到个。

·代价一致搜索：

代价一致搜索是宽度优先搜索的一种推广，不是沿着等长度路径断层进行扩展，而是沿着等代价路径断层进行扩展。代价一致搜索的基本思想是：在代价一致搜索算法中，把从起始节点到任一节点i的路径代价记为g(i)。从初始节点开始扩展，若没有得到目标节点，则优先扩展最少代价g(i)的节点，一直如此向下搜索。

算法步骤如下：

(1) 把初始节点放入Open表,设g()=0，建立一个Closed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) 若节点n不可扩展，则转第（2）步；

(6) 扩展节点n， 生成子节点（i=1,2,……），将其子节点放入Open表，并为每个子节点设置指向父节点的指针，计算各个节点的代价g()，将Open表内的节点按g()从小到大排序, 转向第（2）步。

·A\*算法：

A\*算法是对A算法的估价函数f(n)=g(n)+h(n)加上某些限制后得到的一种启发式搜索算法。对g(n)和h(n)的限制如下：第一，g(n)是对最小代价g\*(n)的估计，且g(n)>0；第二，h(n)是最小代价h\*(n)的下界，即对任意节点n均有h(n)≤h\*(n)。其中，g\*(n)是从出发到达n的最小代价,h\*(n)是n到的最小代价。

算法步骤如下：

(1) 把初始节点放入Open表中，f()=g()+h()；

(2) 如果Open表为空，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，则找到了问题的解，成功退出；

(5) 若节点n不可扩展，则转第(2)步；

(6) 扩展节点n，生成其子节点(i=1, 2, …)，计算每一个子节点的估价值f()(i=1, 2, …)，并为每一个子节点设置指向父节点的指针，然后将这些子节点放入Open表中；

(7) 根据各节点的估价函数值，对Open表中的全部节点按从小到大的顺序重新进行排序；

(8) 转第(2)步。

2.2 算法伪代码

**def** generalSearch(***problem***, ***open***, ***priorityOption***, ***heuristic***):  
 *"""General search algorithm."""* ***closed*** an empty set  
 *# Actions which would be taken from start state to goal state* ***actions*** an empty list  
 **if *priorityOption*** **is True**:  
 *# uniformCostSearch* ***priority*** = 0  
 *# aStarSearch* **if *heuristic*** **is not None**:

use the function ***heuristic*** to calculate ***h***[the start state of ***problem***]  
 ***priority*** += ***h***[the start state of ***problem***]  
 push the tuple (the start state of ***problem***, ***actions***) to ***open*** sorted by ***priority***  
 **else**:  
 *# DFS & BFS*

push the tuple (the start state of ***problem***, ***actions***) to ***open***  
 **while *open*** **is not empty**:  
 pop the first tuple (***currentState***, ***actions***) from ***open***  
 **if *currentState*** is **in *closed***:  
 **continue** add ***currentState*** to ***closed***  
 **if *currentState*** is the goal state of ***problem***:  
 **break  
 for** every successor ***s*** of ***currentState***:

***a*** is the action taken from ***currentState***to ***s***  
 **if *s*** is **not in *closed***:  
 **if *priorityOption*** **is True**:  
 *# uniformCostSearch*

***g***[***s***]= the cost of ***actions*** + [**a**]***priority*** = ***g***[***s***]  
 *# aStarSearch* **if *heuristic*** **is not None**:

use the function ***heuristic*** to calculate ***h***[***s***]  
 ***priority*** += ***h***[***s***]  
 push the tuple (***s***, ***actions*** + [***a***]) to ***open*** sorted by ***priority***  
 **else**:  
 *# DFS & BFS* push the tuple (***s***, ***actions*** + [***a***]) to ***open***  
 **return *actions***

1. **算法实现**

3.1 实验环境与问题规模

实验环境：MacOS，Python3.7，PyCharm；

问题规模：整个地图大小。

3.2 数据结构

由文件util.py定义，包括栈、队列和优先队列。

3.3 实验结果

利用测试文件autograder.py进行打分，结果如下：

3.4 系统中间及最终输出结果（要求有屏幕显示）

1. **总结及讨论**

通过本次实验，我对4种搜索算法——深度优先搜索、广度优先搜索、代价一致搜索和A\*算法有了更深入的理解和掌握，对A\*算法中启发式函数的可纳性和一致性有了进一步的认识。启发式函数的设计与选择具有较高的难度，其效果的好坏遵循以下原则：启发式函数越接近真实最优代价，启发效果越好，但得到这样的启发式函数更困难；启发式函数越远离真实最优代价，启发效果越差，但容易计算。

**参考文献**