Report：Pacman

2016010766 林逸晗

Finding a Fixed Food Dot

1、Implement DFS algorithm in the depthFirstSearch function in search.py (2 points)

数据结构：栈存储，使用栈同时存储action和state，使用DFS生成图的一张支撑树

算法实现：DFS较为简单的写法是使用递归进行。为了使用递归，定义了一个新的函数DFS(probblem, root)，将action和state的list用global推到全局。选择一个节点作为根节点进行搜索，将其展开，对其子节点进行迭代，并且选其中一个子节点作为新的root，同时将action和state同时压入栈，然后调用DFS函数进行更深一层的搜索。当搜索到goal时停止并且返回action_list，而如果调用DFS直至没有后继或者后继被迭代完了，在子树中也没有找到goal，action和state则出栈一层，返回上层寻找。

优点：可以很快地找到一条合理的路线。

缺点：很难找到最优解。

表现：

```
***    pacman layout:        mediumMaze
***    solution length:      246
***    nodes expanded:       269
```

分析：找到了解，但明显不是最优解

2、Implement BFS algorithm in the breadthFirstSearch function in search.py (2 points)

数据结构：队列存储，使用队列同时存储action和state，使用BFS生成图的一张支撑树

算法实现：BFS从toor节点开始，将离root距离最近的节点入队，然后是第二近的一批，同时最开始入队的一批出队。同时，和节点相关的action也一同入队。在从队头弹出较近的节点时，获取它的不重复后继，能保证一定是离root更远的节点，将这些后缀以及何其相关的action压入栈中，以备之后调用检查。

在这里，由于之后存在多goal的问题，一开始我采取的策略是将isGoalState(state)函数的输出改为多态而非二态，并且存储了每一个节点对应的路径上的所有的已访问状态为一个list，将这些list伴随着节点也存入一个队列里和state同时维护。这样，在访问完一个节点之后，可以重置已访问的节点列，并且不至于使其他路径受影响。问题比较大的是在于容易出现大量重复计算，这一问题在后面的A*和UCS中得以解决。

优点：确保在无权图中能够有最优解

缺点：对多目标只能找到最近解（修改之后得以解决），总体时间较为稳定，但对于较深层的节点而言无关的计算太多

表现：

```
***    pacman layout:          mediumMaze
***    solution length:        68
***    nodes expanded:         269
```

分析：找到了最优解，但是展开的点的数目依然很大

3、Implement the uniform-cost graph search algorithm in the uniformCostSearch function in search.py (2 points)

数据结构：优先级队列，使用走到当前节点所需的cost作为priority进行建堆。

算法实现：UCS和BFS的实现很像，但是使用了root节点到当前节点的耗费cost左右priority进行小顶堆维护。由于没有发现utik中的priorityqueue有update接口，导致程序写得较为繁琐，这个问题在A*中得以解决（由于重构代码时间话费太大，故暂时没有对素有问题进行修改）。算法流程和之前的BFS很像，但是在出栈节点比较时有差别，出栈的节点需要比较是否是访问过的节点，如果是的话，比较是否比原节点的开支更小（实际上这一步可以在入栈时就完成，这一优化在A*中写了）。如果找到了目标节点，也要比较是否时是开支更小的路径，如果是的话更新最终的决策。

优点：和BFS类似，但是可以在有权图中求解，也可以在多个goal中找到最小开支路径

缺点：和BFS类似.

表现：

```
*** PASS: test_cases/q3/ucs_1_problemC.test
***    pacman layout:          mediumMaze
***    solution length: 68
***    nodes expanded:         269
```

分析：和BFS相同，但是可以适用于有权图

4、 Implement A^* graph search in the empty function aStarSearch in search.py (3 points)

数据结构：使用优先级队列维护最小堆，权重是当前节点的cost+H（估计值）。

算法实现：由于该算法需要适用于多目标物的情况，需要在获得目标物之后得以掉头，而启发函数可以很简洁地实现这一效果。

原本的单目标物的情况是通过记录已访问节点的list来防止陷入掉头往复的死循环，但是对于多目标物，需要有一个外力来促使获得一个目标之后能够掉头覆盖原有已经访问过的状态，一个一致的、便捷的方法就是使用到最近goal距离的启发函数作为priority。在达到目标之后，将目标删除，此时的G已经变为0，并且这一次入栈允许掉头，此时，当下一次访问到这个节点的时候，原目标物的位置对于它就是F值更大的节点，这促使着它向着新的节点

运动，同样的，由于G相当于以已经访问的goal起算，所以这样的相当于刷新地图，执行新的寻找任务，但又存储着之前的路径

A*的实现和UCS完全一样，只不过将cost改为了cost+H

优点：启发函数的引入使得大量无意义的计算消失，尽管很多情况下无法改变复杂度的阶数，但常数优化在超大规模搜索中很有意义

缺点：H函数的好坏直接决定了A*的好坏，而H函数需要因地制宜，就事论事，还是十分tricky的。

表现：

*** PASS: test_cases/q4/astar_2_manhattan.test
*** pacman layout: mediumMaze
*** solution length: 68
*** nodes expanded: 224

分析：在找到最优解的同时可以降低展开的点的数目，具体表现取决于启发函数的好坏，如果在一定程度上有感性的认识：启发函数的贪婪程度越高，展开的点数目越低，偏离最优解的可能越大

Basic Tasks

1、 Implement the CornersProblem search problem in searchAgents.py (3 points)

算法实现：将state由(x,y)改为((x,y),[havn't visited])，列表中存储着还没有访问过的节点，在上层中调用这个list进行最小曼哈顿路径搜索，并且修改其余几个接口使其能够和这个格式统一，调用上述已经优化过的A*算法即可

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

2、 Implement a non-trivial, consistent heuristic for the CornersProblem in cornersHeuristic (3 points)

实现：使用曼哈顿距离，不考虑地图特征，使用$dis = |\Delta x| + |\Delta y|$，其中可以使用max，sum和min，便利所有goal。

经过观察和调试，dis是当前节点到剩余目标的距离列表，下述写法能达到尽量优，又满足启发函数性质。

```
if len(dis)==3:
    return sum(dis)
else:
    return min(dis)
```

结果：

path length: 106
*** PASS: Heuristic resulted in expansion of 726 nodes

Bonus

Eating All The Dots

the FoodSearchProblem (1 points)

使用BFS直接得到最好的启发函数。
为了减少expand的node的数量，使用一个字典记录已经展开过的节点的后继，因为在BFS中需要考虑的东西只是一次性对某个已知目标寻路，所以这种做法很合适

最终结果：
*** expanded nodes: 9923
*** thresholds: [15000, 12000, 9000, 7000]

失分原因
A*有冗余的展开等等。

Suboptimal Search

Implement the function findPath ToClosestDot in searchAgents.py (1points)

解答：利用BFS的特性可以直接写出本题

Finished at 15:57:14

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: 3/4
Question q8: 3/3

Total: 24/25

```
Finished at 15:57:14

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: 3/4
Question q8: 3/3
---
Total: 24/25
```