Project-1 Pacman 实验报告

陈笑宇 21340246003　林子开 21307110161

1 使用深度优先搜索找到固定的食物点

在本题中，使用深度优先搜索（DFS），并且使用 util 文件中的 Stack 作为维护“后进先出”的数据结构，将从起点到该点的路径加上到达下一个点的 action 作为 path，与下一个点一起压入 Stack 中。此外，我们使用 explored 列表，来保证不再访问那些已经访问过的节点。

请注意！**在我们以下的所有的算法中，我们都规定在某个节点弹出 frontier 的时候，才对其判断是否为目标节点。因此，我们的算法可能比其他先判断是否为 goal，然后再入 frontier 的算法多扩展一些节点。**

我们的实验结果如下：

表 1: DFS 实验结果

迷宫类型	扩展的节点	代价
TinyMaze	15	10
MediumMaze	146	130
BigMaze	390	210

2 广度优先搜索

在本题中，使用广度优先搜索（BFS），并且使用 util 文件中的 Queue 作为维护“先进先出”的数据结构，同样将到达下一个点的 path 和下一个点一起推入 Queue 中。类似的，我们也使用 explored 列表来防止重复访问。我们的实验结果如下：

表 2: BFS 实验结果

迷宫类型	扩展的节点	代价
TinyMaze	15	8
MediumMaze	269	68
BigMaze	620	210

3 一致代价搜索

在本题中，我们使用 util 中的优先队列 PriorityQueue 来保证最低代价节点先被访问。我们将下一个节点、到达下一个节点的 path，以及这条 path 的 cost 作为一个三元组推入优先队列中。

其中，path 的 cost 通过 getCostOfActions 函数计算得到。我们的实验结果如下：

表 3: UCS 实验结果

迷宫类型	扩展的节点	代价
MediumMaze	270	68
MediumDottedMaze	187	1
MeidumScaryMaze	108	68719479864

4 A* 搜索

在本题中，我们在构建 A* 搜索时，我们将从起点到下一个点的 cost 与启发式函数的估计值相加，将其作为权重，推入优先队列 PriorityQueue 中。在本题中，我们统一使用曼哈顿距离启发式进行测试。我们在 bigMaze 上测试了 DFS，BFS，UCS 和 A*，实验结果如下：

表 4: 在 bigMaze 上测试 DFS，BFS，UCS 和 A* 的实验结果

迷宫类型	扩展的节点	代价
dfs	390	210
bfs	620	210
ucs	621	210
astar(with manhattanHeuristic)	549	210

可以看出，A* 比 UCS 扩展的节点略少一点。现在，我们继续在 openMaze 测试上述四种搜索策略：

表 5: 在 openMaze 上测试 DFS，BFS，UCS 和 A* 的实验结果

迷宫类型	扩展的节点	代价
dfs	576	298
bfs	682	54
ucs	683	54
astar(with manhattanHeuristic)	535	54

可以发现，DFS 在 openmaze 上并没有找到最优解。BFS，UCS，A* 都找到了最优解，但是 A* 算法扩展的节点仍然比 BFS 和 UCS 更少。

5 找到所有角落

在本题中，我们将当前的坐标以及已经访问过的角落作为 state。其中，我们**用一个列表来存储已经访问过的角落**。在判断是否为目标状态的 isGoalState 函数中，以及得到下一个状态的 getSuccessors 函数中，我们用相同的方法来判断和更新已访问角落的信息。

我们首先判断当前点的坐标是否为角落；如果该坐标是角落，再判断是否为一个不曾访问过的角落；如果确实是一个不曾访问过的角落，则将信息更新，也即向记录已访问角落的列表中加入该坐标。**当该列表的长度到达 4，也即所有的角落都访问过时，我们认为已经达到目标状态**。

现在，使用 BFS 策略，我们已经能够解决找到所有角落问题。实验结果如下：

表 6: 使用 BFS 找到所有角落的实验结果

迷宫类型	扩展的节点	代价
tinyCorners	435	28
mediumCorners	2448	106

6 使用带启发式的 A* 策略找到所有角落

我们使用的启发式**基于问题松弛**。也即，假设没有围墙，到达距离吃豆人最近角落的曼哈顿距离。示意图如下：

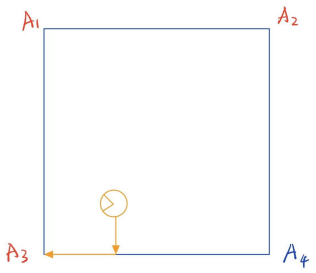

图 1: 所采用的启发式的示意图

在示意图中1中，我们假设没有围墙，且已经访问过 A4 角落，未访问过 A1，A2 和 A3 角落，那么，启发式即等于吃豆人前往角落 A3 的曼哈顿距离。

这个启发式显然是**可采纳的**，因为无论如何，吃豆人最后都会到达这个目前距离自己最近的角落，且在任何有围墙的情况下的路程 cost，必定会大于或等于松弛问题下直接前往最近角落的 cost。

此外，我们的启发式也是**一致的**，我们仍然以图1为例进行说明。假设当前位置到 A1，A2，A3 角落的曼哈顿距离分别为 S_1, S_2, S_3，那么，并且有 $S_3 \leq S_1 \leq S_2$，在进行曼哈顿距离为 c 的移动之后，到达三个角落的距离分别为 X_1, X_2, X_3，并且应该会满足 $X_i \geq S_i - c$, $i = 1, 2, 3$。若仍然距离 A3 最近，则显然启发值最多能降低 c，因为 $c \geq S_3 - X_3$。若距离其他角落更近，我们不妨设距离 A1 更近，那么 $S_3 - X_1 \leq S_3 - S_1 + c \leq c$，也即启发值最多降低 c。至此，我们证明了该启发式是一致的。

我们的实验结果如下：

表 7: 使用 UCS 和 A* 找到所有角落的实验结果

迷宫类型	搜索策略	扩展的节点	代价
mediumCorners	A*(with our own cornersHeuristic)	1745	106
mediumCorners	UCS	2449	106

注意到，使用 A* 所得到的路径长度与使用 UCS 所得到的路径长度相同，都是 106，这也进一步验证了我们给出的启发式是可采纳的且一致的。

7 吃掉所有的点

我们使用的启发式为：以当前位置作为起点，再分别以所有剩余食物作为终点，利用 PositionSearchProblem 定义一个新的搜索问题，用 BFS 寻找到最短路径。然后我们将到达各个食物的最短路径中的最大值（也即到最远食物的距离）作为启发值。

这个启发式是**一致的**，证明如下：假设当前吃豆人所在位置是 X，距离最远（在最短路径意义下）的食物的位置是 Y，那么，X 位置的启发值即为从 X 到 Y 的路径长度 S_{XY}。当吃豆人移动了 c 的路径程度到达了新的位置 Z，此时 Z 到食物 Y 的距离满足 $S_{ZY} \geq S_{XY} - c$。如果此时距离位置 Z 最远的食物仍然是 Y 处的食物，那么 Z 处的启发值显然满足 $S_{XY} - S_{ZY} \leq c$。若此时距离位置 Z 最远的不是 Y，而是 W 处的食物，那么应该满足 $S_{ZY} < S_{ZW}$，显然也有 $S_{XY} - S_{ZW} \leq c$。综上，不论吃豆人在移动后距离最远食物是 Y 还是 W，启发值最多都只可能下降 c。至此，我们证明了我们给出的启发式是一致的。

我们的实验结果如下：

表 8: 带启发式的 A* 搜索在 trickySearch 上的实验结果

迷宫类型	扩展的节点	代价	探索时间
trickySearch	4110	60	21.5

8 次优搜索

在本题中，我们总是使用 BFS 找到距离吃豆人最近的食物。我们对 bigSearch 进行搜索，迅速得到了一条 cost 为 350 的路径。但是，在这条路径中，吃豆人忽略了右侧的一个食物，最后还要回到迷宫右侧，这严重导致了该路径的次优性，如下图所示：

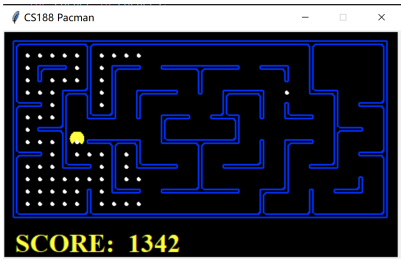

图 2: 次优路径忽略了右侧的一个点

下面，我们给出一个简单的一维空间上例子，说明重复前往最近的点，并不会找到吃掉所有点的最短路径，如下所示：

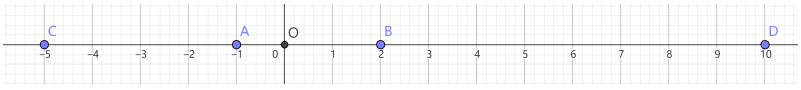

图 3: 一个说明贪婪策略无法找到全局最优解的一维例子

在这个例子中，如果使用贪心算法，不断前往最近的点，则吃豆人的路径应该为 $O \to A \to B \to C \to D$，这条路径的长度为 $1 + 3 + 7 + 15 = 26$。但实际上，这条路径缺乏对全局信息的掌握，反复在中间部分来回行走，导致其不是最优。如果使用 BFS 或 UCS 或 A* 等算法，可以发现最优路径应该是 $O \to A \to C \to B \to D$，相应的总路径长度是 $1 + 4 + 7 + 8 = 20$。