

Project-1 Pacman 实验报告

陈笑宇 21340246003 林子开 21307110161

目录

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 一致代价函数

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 推入优先队列中，也即一

个三元组 $((nextState, new_path), new_cost)$

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：

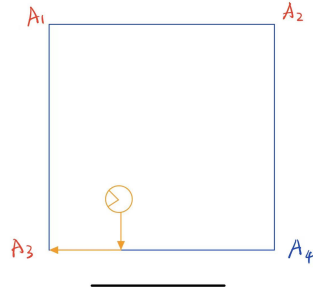


图 1: 所采用的启发式的示意图

在示意图中??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??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 $S_3 < S_1 < S_2$ ，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 $X_i > S_i - c, i = 1, 2, 3$ 。若仍然距离 A3 最近，则显然启发值最多能降低 c ，因为 $c > S_3 - X_3$ 。若距离其他角落更近，我们不妨设距离 A1 更近，那么 $S_3 - X_1 < S_3 - S_1 + c < c$ ，也即启发值最多降低 c 。因此，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。

对我们使用的启发式函数的分析：

7 吃掉所有的点

8 次优搜索