

# 迷宫问题

作者：柳絮源

日期：2025.04.10

本项目旨在探索和实现不同的图搜索算法，以解决带有特殊元素（如传送门、沼泽、加速器）的迷宫问题。我们实现了以下四种经典的寻路算法：

1. 深度优先搜索 (DFS) (对应 `maze_DFS.py`)
2. 广度优先搜索 (BFS) (对应 `maze_BFS.py`)
3. Dijkstra 算法 (对应 `maze_dijkstra.py`)
4. A\* 算法 (对应 `maze_A_star.py`)

所有算法实现均考虑了增强的迷宫环境，并支持八个方向的移动（上、下、左、右及四个对角线方向）。为了更直观地理解算法的工作原理，我们利用 `matplotlib` 库对搜索过程和最终找到的路径进行了动态可视化展示。

## 新迷宫元素

- 0: 空地: 基础移动代价（直行 1, 斜行  $\sqrt{2}$ ）。
- 1: 墙壁: 不可通行。
- 2: 沼泽: 可通行，移动代价是相应空地的 2 倍。
- 3: 加速器: 可通行，移动代价是相应空地的 0.5 倍。
- 4, 5, ...: 传送门: 成对出现，进入一个传送门会零代价瞬间移动到同 ID 的另一个传送门位置。

## 算法思路

### 1. 深度优先搜索 (DFS - `maze_DFS.py`)

- 核心思想: DFS 采用“一路走到黑”的策略，沿着一条路径深入探索，直到遇到死路或找到终点才回溯。它不保证找到步数或代价最优的路径，但能找到一条从起点到终点的可行路径（如果存在）。
- 实现细节:
  - 通常使用栈（代码中采用递归或显式栈的迭代方式）来管理待访问节点。
  - 使用 `visited` 集合防止在单次搜索路径中重复访问同一节点，避免无限循环。
  - `parent` 字典记录路径来源，用于找到路径后回溯。
  - 探索邻居时检查 8 个方向。

- **传送门处理:** 当访问到一个传送门节点时, 如果其配对的传送门目标尚未被访问, 则将其加入待探索列表 (栈顶), 并将当前传送门设为其父节点。传送本身代价为 0 (体现在路径总代价计算中)。
- **代价计算:** DFS 本身不直接使用代价来指导搜索, 但找到路径后, 会根据路径经过的单元格类型 (空地、沼泽、加速器) 和移动方向 (直/斜) 计算总代价。
- **可视化:**
  - 蓝色点 ( `visited_scatter` ) 按 DFS 探索和回溯的顺序动态展示访问过的节点。
  - 红色路径 ( `path_line` ) 在找到解后, 展示最终的回溯路径。

## 2. 广度优先搜索 (BFS - `maze_BFS.py` )

- **核心思想:** BFS 从起点开始, 按层级、逐圈向外扩展搜索。它使用队列来管理待访问节点, 保证了最先访问距离起点**步数**最近的节点。因此, BFS 找到的路径一定是**步数最少**的路径。但在有不同代价的迷宫中, 步数最少不一定意味着总代价最低。
- **实现细节:**
  - 使用 `deque` 作为先进先出 (FIFO) 队列。
  - `visited` 集合记录已加入队列或已处理的节点, 防止重复。
  - `parent` 字典用于路径回溯。
  - 探索邻居时检查 8 个方向。
  - **传送门处理:** 当从队列中取出一个传送门节点时, 检查其配对目标。如果目标未被访问过, 则将其加入 `visited` 集合, 设置父节点, 并加入队列。传送本身不计步数 (在路径回溯时不体现为一步), 代价为 0。
  - **代价计算:** BFS 自身不考虑代价, 寻路基于步数。找到路径后, 同样可以计算其总代价。
- **可视化:**
  - 蓝色点 ( `visited_scatter` ) 按 BFS 的层级访问顺序动态展示已探索的节点。
  - 红色路径 ( `path_line` ) 在搜索结束后, 展示回溯得到的最短步数路径。

## 3. Dijkstra 算法 ( `maze_dijkstra.py` )

- **核心思想:** Dijkstra 算法旨在寻找图中从单一源点到所有其他可达点的**最低累计代价**路径。它使用优先队列 (最小堆) 来管理待访问节点, 每次总是选择当前已知距离起点代价最小的节点进行扩展。
- **实现细节:**
  - 使用 `heapq` (最小堆) 作为优先队列, 存储 (当前累计代价, 节点)。
  - `dist` (或 `g_score` ) 字典存储从起点到各节点的当前已知最低代价。
  - `parent` 字典用于路径回溯。
  - `processed` 集合记录已确定最低代价的节点 (从优先队列中取出并处理过的)。
  - 探索邻居 (8 方向): 计算到达邻居的**新代价** = 当前节点代价 + 移动到邻居的单步代价。单步代价根据目标单元格类型 (沼泽\*2, 加速器\*0.5) 和移动方向 (直行 1, 斜行  $\sqrt{2}$ ) 计算。

- **代价更新:** 如果新代价低于邻居当前的已知最低代价, 则更新 `dist`, 记录 `parent`, 并将 (新代价, 邻居节点) 加入优先队列。
- **传送门处理:** 当处理一个传送门节点时, 计算到达其配对目标的代价 (当前代价 + 0)。如果这个代价更低, 则更新目标的 `dist` 和 `parent`, 并将 (新代价, 目标节点) 加入优先队列。
- **可视化:**
  - 蓝色点 (`visited_scatter`) 按 Dijkstra 算法处理节点 (即确定其最低代价) 的顺序动态展示。这通常是代价大致递增的顺序。
  - 红色路径 (`path_line`) 在搜索结束后, 展示回溯得到的全局最低代价路径。

## 4. A\* 算法 ( `maze_A_star.py` )

- **核心思想:** A\* 算法是 Dijkstra 算法的一种优化, 属于启发式搜索。它在选择下一个要探索的节点时, 不仅考虑从起点到该节点的实际代价  $g(n)$  (同 Dijkstra), 还考虑一个**启发式函数**  $h(n)$  估算的从该节点到终点的代价。A\* 优先探索  $f(n) = g(n) + h(n)$  值最低的节点, 这使得搜索更具方向性, 通常能更快地找到最优路径。
- **实现细节:**
  - 使用 `heapq` 作为优先队列, 存储 (`f_score`, 节点) 或 (`f_score`, `g_score`, 节点)。
  - `g_score` 存储实际代价, `parent` 用于回溯, `processed` 记录已处理节点。
  - 探索邻居 (8 方向): 计算实际代价 `new_g` 到达邻居 (同 Dijkstra)。
  - **启发式函数:** 使用**对角距离 (Chebyshev距离)**  $h(n) = \max(\text{abs}(dx), \text{abs}(dy))$  作为启发式, 它适用于 8 方向移动且是“可接受的”(admissible, 即不高于实际最低代价), 保证 A\* 找到最优解。
  - **代价与入队:** 计算  $f_{\text{new}} = \text{new\_g} + h(\text{邻居节点})$ 。如果 `new_g` 比邻居已知的 `g_score` 更低, 则更新 `g_score`, 记录 `parent`, 并将 (`f_new`, 邻居节点) 加入优先队列。
  - **传送门处理:** 当处理一个传送门节点时, 计算到达其配对目标的 `new_g` (当前 `g_score` + 0)。如果更低, 更新目标的 `g_score` 和 `parent`, 计算目标的  $f_{\text{new}} = \text{new\_g} + h(\text{目标节点})$ , 并将 (`f_new`, 目标节点) 加入优先队列。
- **可视化:**
  - 蓝色点 (`visited_scatter`) 按 A\* 算法处理节点的顺序动态展示。由于启发式的引导, 探索范围通常比 Dijkstra 更聚焦于目标方向。
  - 红色路径 (`path_line`) 在搜索结束后, 展示回溯得到的最优 (最低代价) 路径。