迷宫问题

作者：柳絮源
日期：2025.04

1. **迭代加深深度优先搜索 (IDDFS)** (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`)

我撰写的代码不仅实现了算法逻辑，还利用 `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。所有算法均支持在迷宫中进行八个方向的移动（上、下、左、右以及四个对角线方向）。

算法思路

1. 迭代加深深度优先搜索 (IDDFS - `maze_DFS.py`)

- **核心思想:** IDDFS 结合了深度优先搜索 (DFS) 的空间效率和广度优先搜索 (BFS) 的完备性（在找到路径层面上）。它通过限制搜索深度，并逐步增加这个深度限制来进行多次 DFS。从深度 0 开始，每次迭代深度加 1，直到找到目标节点或遍历完所有可达节点。
- **实现细节:**
  - 使用递归实现核心的 DFS 逻辑 (`dfs` 函数)。
  - 外部循环 (`while True`) 控制搜索深度的增加 (`max_depth`)。
  - 每次迭代都会重置 `visited` 集合和 `lst` (用于路径回溯)。
  - 允许 8 方向移动，直行代价为 1，斜行代价为 $\sqrt{2}$。
  - 找到的路径是步数最少的路径之一（因为它是按深度逐层搜索的）。
  - 确保找到从起点到终点的总代价最小的路径。
- **可视化:**
  - 蓝色点 (`scatter`) 显示每次迭代中访问过的节点。由于迭代会重复访问节点，可视化会展示所有迭代中累积访问过的节点。
  - 红色路径 (`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，逐层向外扩展搜索。它使用一个队列来存储待访问的节点，保证了首先访问距离起点最近（按步数计算）的节点。因此，BFS 找到的路径一定是步数最少的路径。

- **实现细节:**
  - 使用 `deque` 作为队列。
  - `visited` 集合用于记录已访问节点，防止重复搜索。
  - `lst` 字典用于存储每个节点的父节点，以便找到路径后进行回溯。
  - 允许 8 方向移动，直行代价为 1，斜行代价为 $\sqrt{2}$。
  - 确保找到从起点到终点的总代价最小的路径。
- **可视化:**
  - 蓝色点 (`scatter`) 按 BFS 的访问顺序动态展示已探索的节点。
  - 红色路径 (`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 算法用于查找图中从源节点到所有其他节点的最短路径（按路径总代价计算）。它使用优先队列来维护待访问的节点，每次选择距离源节点累计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 `(cost, node)`。
  - `g_score` 字典存储从起点到当前节点的实际最小代价。
  - `lst` 字典用于路径回溯。
  - `visited` 集合用于记录已处理过的节点（已找到其最短路径）。
  - 允许 8 方向移动，直行代价为 1，斜行代价为 $\sqrt{2}$。
  - 确保找到从起点到终点的总代价最小的路径。
- **可视化:**
  - 蓝色点 (`scatter`) 按 Dijkstra 算法处理节点的顺序（通常是按代价递增）动态展示。
  - 红色路径 (`line`) 在搜索结束后，展示回溯得到的最低代价路径。

4. A* 算法 (`maze_A_star.py`)

- **核心思想:** A* 算法是一种启发式搜索算法，它结合了 Dijkstra 算法（考虑实际代价 `g(n)`) 和启发式信息（估计从当前节点到目标的代价 `h(n)` ）。它优先探索具有最低 `f(n) = g(n) + h(n)` 值的节点，从而更有效地引导搜索方向，通常比 Dijkstra 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, g_score, node)`。
  - `g_score` 存储实际代价， `f_score` 存储估计总代价。
  - `lst` 用于路径回溯。
  - `visited` 集合记录已访问节点（从优先队列中取出并处理过的节点）。
  - 允许 8 方向移动，直行代价为 1，斜行代价为 $\sqrt{2}$。
  - 使用**曼哈顿距离**作为启发式函数 `h(n)`： `abs(x1 - x2) + abs(y1 - y2)`。这是一个可接受的启发式（不高估实际代价），保证了 A* 算法找到最优路径。

- **可视化:**
  - 蓝色点（`scatter`）按 A* 算法处理节点的顺序动态展示。通常 A* 探索的节点数会少于 Dijkstra。
  - 红色路径（`line`）在搜索结束后，展示回溯得到的最优（最低代价）路径。