跳马实验报告

这是作者的 **「2025秋 数据结构及其算法」** 其中一个实验内容。

---

1 需求分析

在 n * n 棋盘上，使“马”从给定起点出发，按照国际象棋马的走法遍历所有格子且只到达一次；若能覆盖全部格子即为一条完整巡游路径。需求包括：

- 输入: 棋盘规模 n (典型 n>=5) 与起始坐标 (x_0,y_0)。
- 输出: 跳马可行的完整路径。
- 性能: 在合理时间内给出一个解乃至全部解。
- 健壮性: 非法输入检测；无解时返回明确提示。
- 可扩展: 支持附加特殊规则（如排除的格子，一般矩形棋盘等），可视化功能（跳马路径回放与手动跳马等）。

2 概要设计

2.1 设计思想/算法

对于求解部分，先给出能正确工作的朴素深度优先搜索（DFS 回溯）实现，再引入 Warnsdorff 启发式搜索算法来显著降低回溯量并快速得到大棋盘上的解。

1. 普通 DFS（由浅入深）

    - 基本思路：从起点出发，在搜索的每一个位置尝试所有合法马步，走到步数 n * n 则成功；若某一步无法继续则回溯并撤销标记。
    - 优点：实现简单、能枚举所有解。
    - 缺点：搜索树巨大，甚至随 n^2 指数增长，在大棋盘上回溯开销极高且求解速度极慢。

    伪代码（核心）：

```
函数 JumpingSearch(stack, visited, size, depth, solutionCount):
    若 depth == size * size:
        输出 "found:"
        DrawChessboard(stack, size)
        solutionCount ← solutionCount + 1
        返回

    currentPos ← stack[depth - 1].pos
```

```
    对 dir 从 0 到 7:
        nextPos.x ← currentPos.x + moveDirections[dir].x
        nextPos.y ← currentPos.y + moveDirections[dir].y
        若 ValidPos(nextPos, size, visited):
            stack[depth].pos ← nextPos
            stack[depth].direction ← dir
            visited[nextPos.x][nextPos.y] ← true
            JumpingSearch(stack, visited, size, depth + 1, solutionCount)
            visited[nextPos.x][nextPos.y] ← false   // 回溯撤销
// 说明:
// stack: 数组，元素含 pos 与 direction；depth 为当前步序 (1..size*size)
// visited: 布尔矩阵标记已访问格
// moveDirections: 8 个马步偏移常量
// ValidPos: 越界与未访问检测
// 回溯保证穷举所有可能路径；找到完整路径即输出并计数
```

2. Warnsdorff 启发式（用于快速求解大棋盘）

- 基本思路：在每一步选择下一个位置时，优先选择后继度（即该位置可继续走的合法马步数）最小的格子，从而减少未来回溯概率。
- 优点：大幅降低回溯次数，能在大棋盘上快速找到解。
- 缺点：实现复杂度增加；某些棋盘和起点仍可能无解或需较多回溯。

伪代码（核心）：

```
函数 WarnsdorffSearch(stack, visited, size, depth, solutionCount):
    若 depth == size * size:
        输出 "found:"
        DrawChessboard(stack, size)
        solutionCount ← solutionCount + 1
        返回

    currentPos ← stack[depth - 1].pos
    nextMoves ← GenerateNextMoves(currentPos, visited, size) // 按后继度排序
    对 move in nextMoves:
        nextPos ← move.pos
        stack[depth].pos ← nextPos
        stack[depth].direction ← move.direction
        visited[nextPos.x][nextPos.y] ← true
        WarnsdorffSearch(stack, visited, size, depth + 1, solutionCount)
        visited[nextPos.x][nextPos.y] ← false   // 回溯撤销
// GenerateNextMoves: 生成所有合法后继并按后继度升序排序
```

对于展示部分，设计一个解法回放模块，读取保存的路径文件并逐步在棋盘上显示马的移动过程，验证路径合法性。在求解完毕后直接执行。

2.2 数据结构

- 棋盘: 使用二维布尔数组 `board[n][n]`，未访问用 0。
- 方向集合: 固定长度数组 `moves[8] = {(2,1),(1,2),(-1,2),(-2,1),(-2,-1),(-1,-2),(1,-2),(2,-1)}`。
- 访问标记: 直接依据 `board` 对应元素是否为 0。
- 回溯**栈**: 记录从起点到当前点的路径并用作下一个步选择依据。此外还有搜索算法中递归调用栈。
- 算法实现辅助结构: 如 Warnsdorff 算法中的后继列表等。
- 统计信息: 尝试步数、回溯次数、用时等。

3 详细设计

使用 Warnsdorff 算法的 Jumping 模块划分:

- 输入校验: 检查 n 合法范围与起点坐标边界。
- 数据初始化: 分配 `board`，清零并置起点。
- 可行性检测函数 `isValid(x,y)`: 越界 & 已访问判断。
- 后继度计算函数 `degree(x,y)`: 遍历 8 方向统计未访问合法格。
- 下一步生成函数 `nextMoves(x,y)`: 过滤合法后继，附带度数计算，按度数升序返回。
- 主搜索 `search(step,x,y)`:
  - 若 `step == n * n` 成功，按需输出解法。
  - 获取 `nextMoves`；循环尝试，递归；失败则复原 `board`。
- 结束处理: 输出路径或无解或未找到解等信息，统计用时与回溯次数。
- 统计记录: 在每次递归进入、回退更新 `counters`。

边界与异常:

- 对于较小的棋盘（如 n=5），将给出全部解；更大的棋盘找到一个解即跳出搜索，进入回放模块。
- 极大棋盘可能导致性能问题，需合理限制 n 上限。
- 起点不合法立即报错。

解法回放模块：

- 读取保存的路径文件并检查合法性
- 解析路径数据
- 逐步在棋盘上显示马的移动过程
  - 由于过程在控制台进行，棋盘规模相对较小（设定最大棋盘大小为 12 * 12），且步骤间 pause 时间较长，可以采用朴素循环方法实现

4 实验结果

测试环境: 本地普通笔记本 CPU 单线程。

普通 DFS 样例结果:

- n=5, 起点 (0,0): 在合理时间（秒级）内找到全部 304 条路径（未考虑对称重复等情况）。

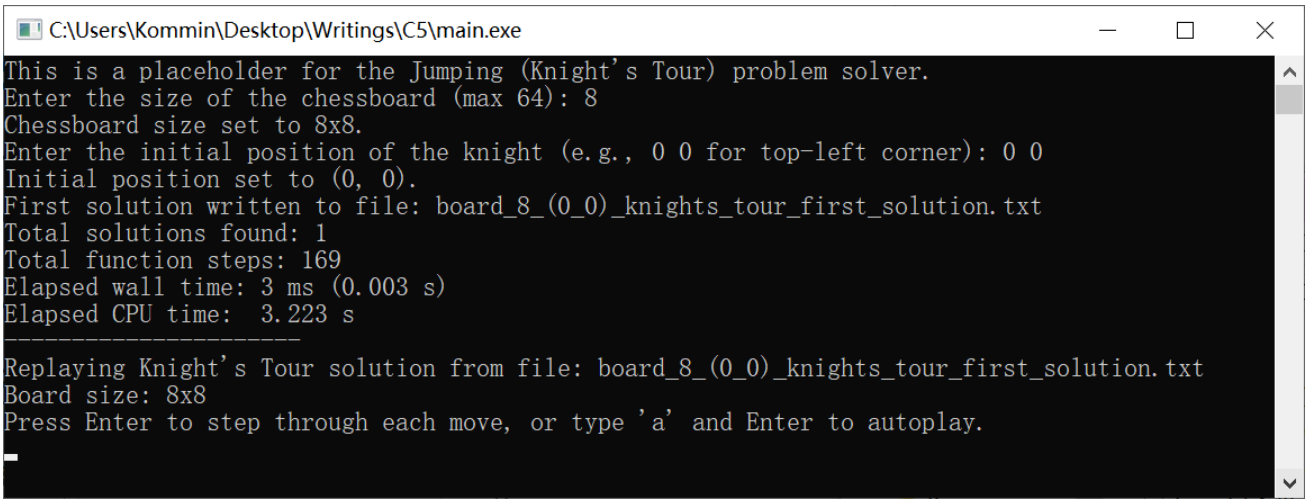
```
Solution #304 found.
Total solutions found: 304
Total function steps: 1735079
Elapsed wall time: 658 ms (0.658 s)
Elapsed CPU time:  19.076 s
----------------------
Replaying Knight's Tour solution from file: board_5_(0_0)_knights_tour_first_solution.txt
Board size: 5x5
Press Enter to step through each move, or type 'a' and Enter to autoplay.
```

- n=8, 起点 (0,0): 在合理时间内找到至少一条完整路径。但是在 5 * 10^7 次搜索内仅找到 4 条路径。

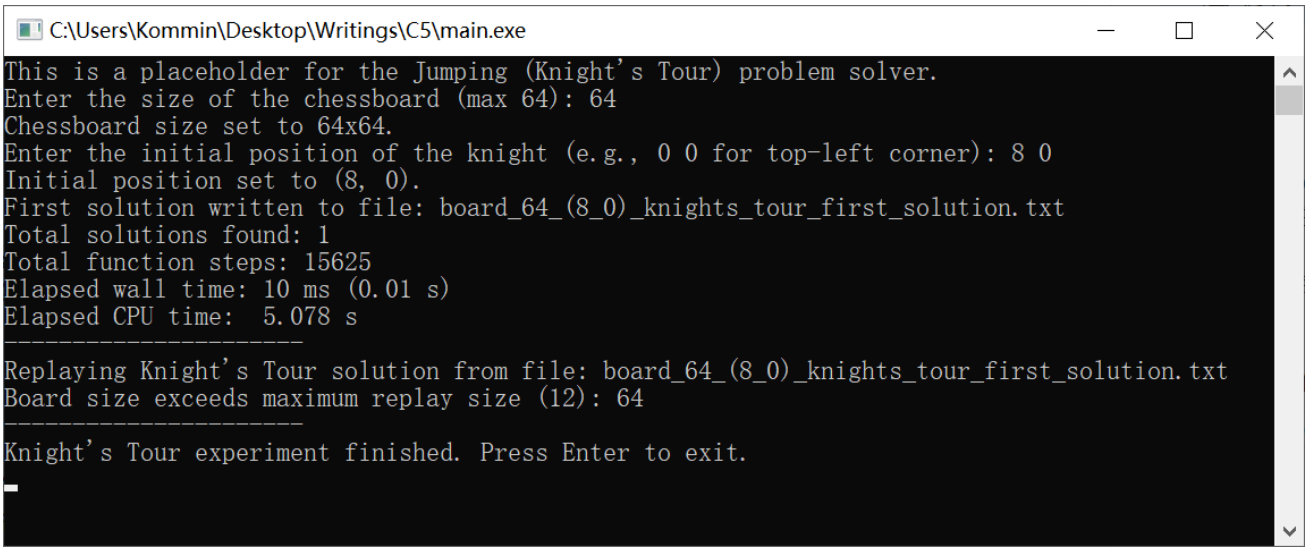
```
This is a placeholder for the Jumping (Knight's Tour) problem solver.
Enter the size of the chessboard (max 8): 8
Chessboard size set to 8x8.
Enter the initial position of the knight (e.g., 0 0 for top-left corner): 0 0
Initial position set to (0, 0).
A complete tour is found:
   1   60   39   34   31   18    9   64
  38   35   32   61   10   63   30   17
  59    2   37   40   33   28   19    8
  36   49   42   27   62   11   16   29
  43   58    3   50   41   24    7   20
  48   51   46   55   26   21   12   15
  57   44   53    4   23   14   25    6
  52   47   56   45   54    5   22   13
Function steps: 10000000, Current depth: 53, Solutions found: 1
```

Warnsdorff 样例结果:

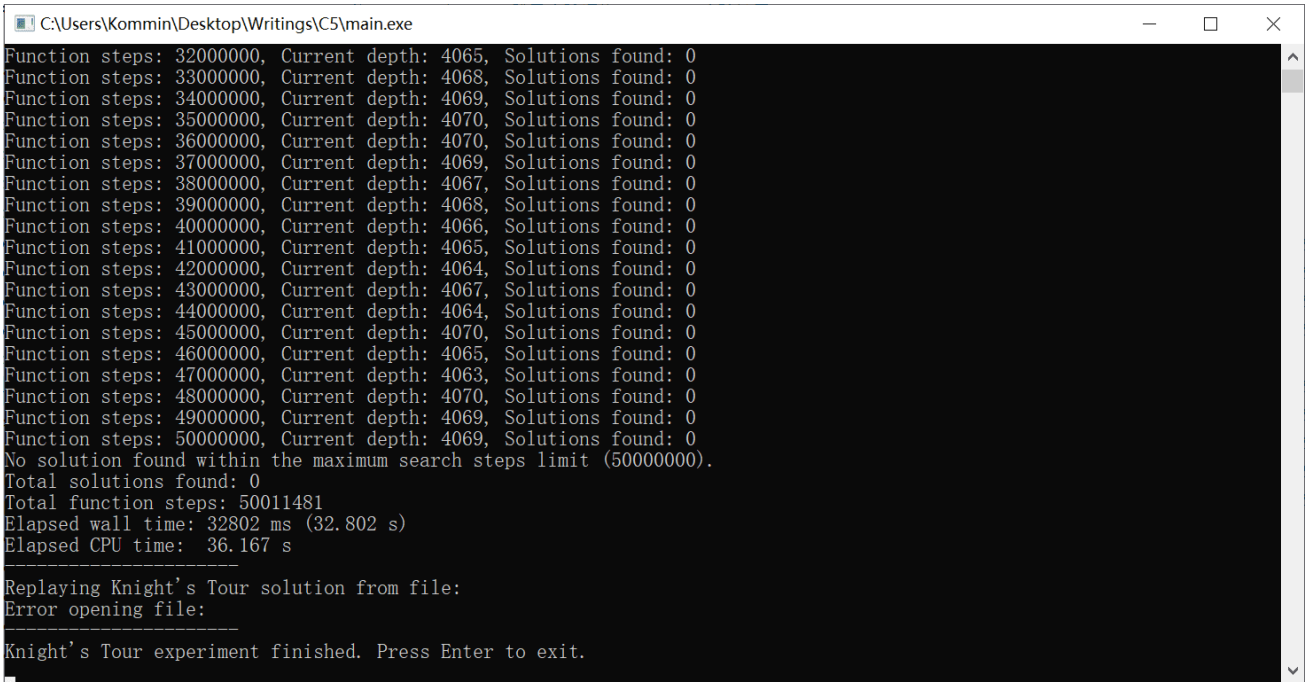
- n=8, 起点 (0,0): 成功生成一条完整路径，用时 3 ms，搜索执行次数 169（作为对比，路径长度 64）。



- n=64, 起点 (8,0): 第一条完整路径用时 10 ms，搜索执行次数 15625（路径长度 4096）。



- n=64, 起点 (0,0): 在 5 * 10^7 次搜索内找不到完整路径，表明某些起点仍较难。



对于极大棋盘，启发式显著提升效率，但仍存在起点敏感性。以上 Warnsdorff 解法均成功输出到文件。

跳马解法回放结果：

- n=8, 起点 (0,0): 成功读取并回放 Warnsdorff 算法生成的路径，逐步显示马的移动过程，验证路径合法性。