proj1 解释文档

陈永鸿

2025 年 11 月 7 日

笔者首先将会对解题思路进行简要概述，介绍使用的核心方法，随后对梳理算法流程，接着介绍各个函数的作用和使用的数据结构，最后说明实现难点和创新点。

1 解题思路概要

核心思路可以分为两大部分：

首先，利用约束规则，确定出一定可以放灯和一定不能放灯的地方。

然后，当上述方法无法推进时，使用带有优先级的 DFS 进行暴力求解。笔者设置了两套优先级分数，首先将采用数字格优先策略，通过模仿人类思维（笔者是这样玩这个游戏的）来提高搜索速度，此时使用第一套优先级分数；如果该策略失败，则采用另一种优先级分数，完成剩下的搜索任务。

2 算法流程梳理

算法流程如下：

2.1 题目读取与初始化

1. 读取行、列信息，将输入字串转化为二维矩阵 board 存储。
2. 初始化状态矩阵 state(其中 U 表示未知是否可放置，B 表示静止放置，L 表示已放置灯)
3. 初始化状态矩阵 light, 被照亮 +1，未被照亮 0

2.2 利用约束规则推进

这一阶段的目的是利用约束规则，尽可能确定出一定可以放灯和一定不能放灯的位置，从而减少后续 DFS 的搜索空间。

1. 数字格约束：对于每一个数字格，统计它的四邻：
   - 若已放灯 $>$ 需要灯，则返回 0，当前路径失败。
   - 若已放灯 $=$ 需要灯，标记周围周围的 U 格为 B，不能放灯
   - 若已放灯 $+$ 未知 $=$ 需要灯，标记周围的 U 格为 L
   - 若已放灯 $+$ 未知 $<$ 需要灯，则返回 0，当前路径失败。
2. 光照约束：对于任意一个白格，如果它是黑暗的，那么检查它的四条线，如果有不被阻挡的灯，那么它应该得是光明的
   - 如果在方向上有空白可以放灯的位置，那么那个位置是可能的灯的位置
   - 如果在方向上没有可以放灯的位置，那么它那个位置必须放灯

3. 不能放灯约束：如果一个白格它的四条线上有没有被遮挡的灯，那么它应该是不能放灯的

4. 灯互相不可见约束：主要用于检查

2.3 带优先级的 DFS 暴力求解

当约束规则无法推进时，进入 DFS 暴力求解阶段。此时，笔者设计了两套优先级分数，分别对应两种不同的搜索策略。

1. 数字格优先策略：优先选择与数字格相关的未知格进行放灯尝试，优先级分数计算公式为：

$$\text{score} = 1000 \cdot \text{residual} + 100 \cdot |\text{unknown}| + \begin{cases} 0, & \text{if can\_put\_lamp}(u_k) \\ 10, & \text{otherwise} \end{cases} + \begin{cases} 0, & \text{if light}(u_k) = 0 \\ 1, & \text{otherwise} \end{cases}$$

其中 $u_k = \text{unknown}[k]$，$\text{residual} = \text{need} - \text{num\_lamp}$。分数设计原理是优先看数字格剩余需求，其次看未知格数，最后考虑该格当前是否可放灯和是否被照亮。分数越小越优先尝试。

2. 普通优先策略：优先选择未知格数最少的行或列进行放灯尝试

$$\text{score} = 1000 \cdot \begin{cases} 1, & \text{if can\_put\_lamp}(i,j) \\ 0, & \text{otherwise} \end{cases} + 100 \cdot \begin{cases} 0, & \text{if light}(i,j) = 0 \\ 1, & \text{otherwise} \end{cases}$$

这里只看该格当前是否可放灯和是否被照亮。分数越小越优先尝试。

2.4 检查与结果输出

当 check 函数检查得到一个可行解时，输出结果矩阵 board。

3 主要函数介绍与数据结构说明

3.1 主要数据结构

- 全局维度：`int row, column`，网格行列数。
- 输入与棋盘视图：`string input`，`vector<string> board`。
  其中 `board[i][j]` 取值：`'.'` 白格、`'X'` 无数字黑格、`'0'..'4'` 数字黑格；黑格均阻光。
- 方向数组：`int dr[4]={-1,1,0,0}; int dc[4]={0,0,-1,1};` 用于四连通遍历。
- 光照矩阵：`vector<vector<int>> light`。记录每格受到的光照计数（包含灯自身与四向直线可见的灯，遇黑格停止）。
- 状态矩阵：`vector<vector<char>> state`。搜索状态：`'U'` 未定可放，`'B'` 禁放，`'L'` 已放灯。
- 坐标结构：`struct pos { int r; int c; };` 辅助收集数字格四邻中的未知白格坐标。
- 分支选择：`struct Choice { int r; int c; bool can_put_lamp; };` DFS 分支点与“是否可放灯”标记。
- 回溯暂存：`temp_state`、`temp_light`（`vector<vector<char>>` 与 `vector<vector<int>>` 的拷贝），用于 DFS 回溯。

约定：`board` 只读；`state` 与 `light` 随搜索动态维护；`dr/dc` 简化四向扫描。

3.2 函数说明

首先是一些小函数，用于实现一些基本操作与判断：

1. `bool in_boarder(int r, int c)`：判断坐标 (r,c) 是否在棋盘内
2. `bool is_black(int r, int c)`：判断坐标 (r,c) 是否为黑格
3. `bool is_white(int r, int c)`：判断坐标 (r,c) 是否为白格
4. `bool is_number(int r, int c)`：判断坐标 (r,c) 是否为数字格
5. `int get_number(int r, int c)`：获取数字格的数字
6. `bool find_lamp(int r, int c)`：判断坐标 (r,c) 处的四个方向是否有灯
7. `void adjust(int r, int c, int delta)`：调整坐标 (r,c) 处的光照状态，delta 为 +1 或-1
8. `bool put_light(int r, int c)`：在坐标 (r,c) 处放置灯
9. `bool remove_light(int r, int c)`：移除坐标 (r,c) 处的灯 (注：实际没用使用到)
10. `bool cannot_put_light(int r, int c)`：将坐标 (r,c) 处标记为不能放灯

其次是利用约束规则推进的函数：

1. `bool single_work(bool &changed)`：该函数主要实现了如算法流程梳理中所述的四条约束规则，返回值分别表示：
   (a) changed = 1, return 1: 表示有改动，且未发现冲突，仍需要继续推进。
   (b) changed = 0, return 1: 表示无改动，且未发现冲突，推进结束，进入 DFS 阶段或 check=1 输出结果。
   (c) return 0: 表示发现冲突，当前路径失败。
2. `bool work()`：该函数循环调用 single_work，直到 single_work 返回无改动或失败。

接着是优先级函数：

1. `bool num_first(Choice &ch)`：通过算法流程所示的分数公式，计算所有未知格的优先级分数，选择分数最低的格子作为下一步尝试放灯的位置。
2. `bool no_number_first(Choice &ch)`:当无法从数字格导出有效分支时,面向全盘所有未定白格（`state=='U'`）选取一个备选分支点。

然后是 DFS 函数: **bool dfs() :**

1. 首先，调用 work() 利用约束规则推进，work 会反复调用 single_work 直到无改动或失败。
2. 若 work 这一步直接失败了，那么回溯至 dfs 的上一层（具体哪个上一层在第六点的举例说明）。
3. 然后，进行 DFS 搜索，首先调用优先级函数选取一个分支点，尝试放灯与不放灯两种选择。
4. 优先选择放灯，如果没有矛盾，递归调用 dfs()；如果出现矛盾，回溯至放灯这一步，尝试不放灯。
5. 现在尝试不放灯，如果没有矛盾，递归调用 dfs()；如果出现仍然矛盾，回溯至当前 dfs() 的上一层。
6. 这里单纯文字理论说明比较难说清楚，采用举例说明：假设先选择了 A 点放灯，接下来 B 点放灯与不放灯都矛盾，那么回溯至 A 点放灯这一步，选择 A 点不放灯。

最后是检查与输入输出函数：

1. `bool check()`：检查当前状态是否为完整解，若是则返回真。
2. `vector<string> cut(string s, int r, int c)`：将输入的一维向量变成二维的。
3. 输出直接就是二维矩阵的输出，没有另外写函数。

4 难点与创新点说明

该问题是个 NP 问题，暴力搜索的时间复杂度是指数级别的，因此如何优化搜索策略和剪枝是实现的关键。笔者认为以下几点是针对该难点的创新点：

1. 利用约束规则推进：在每次 DFS 递归前，先进行合理利用规则，确定一定能放灯和不能放灯的格子，尽可能减少搜索空间。

2. 数字格优先策略与打分函数：通过模仿人类思维，先去解决数字格相关的未知格，通过打分函数设计，优先选择更有可能影响整体解的格子进行尝试。

5 流程图

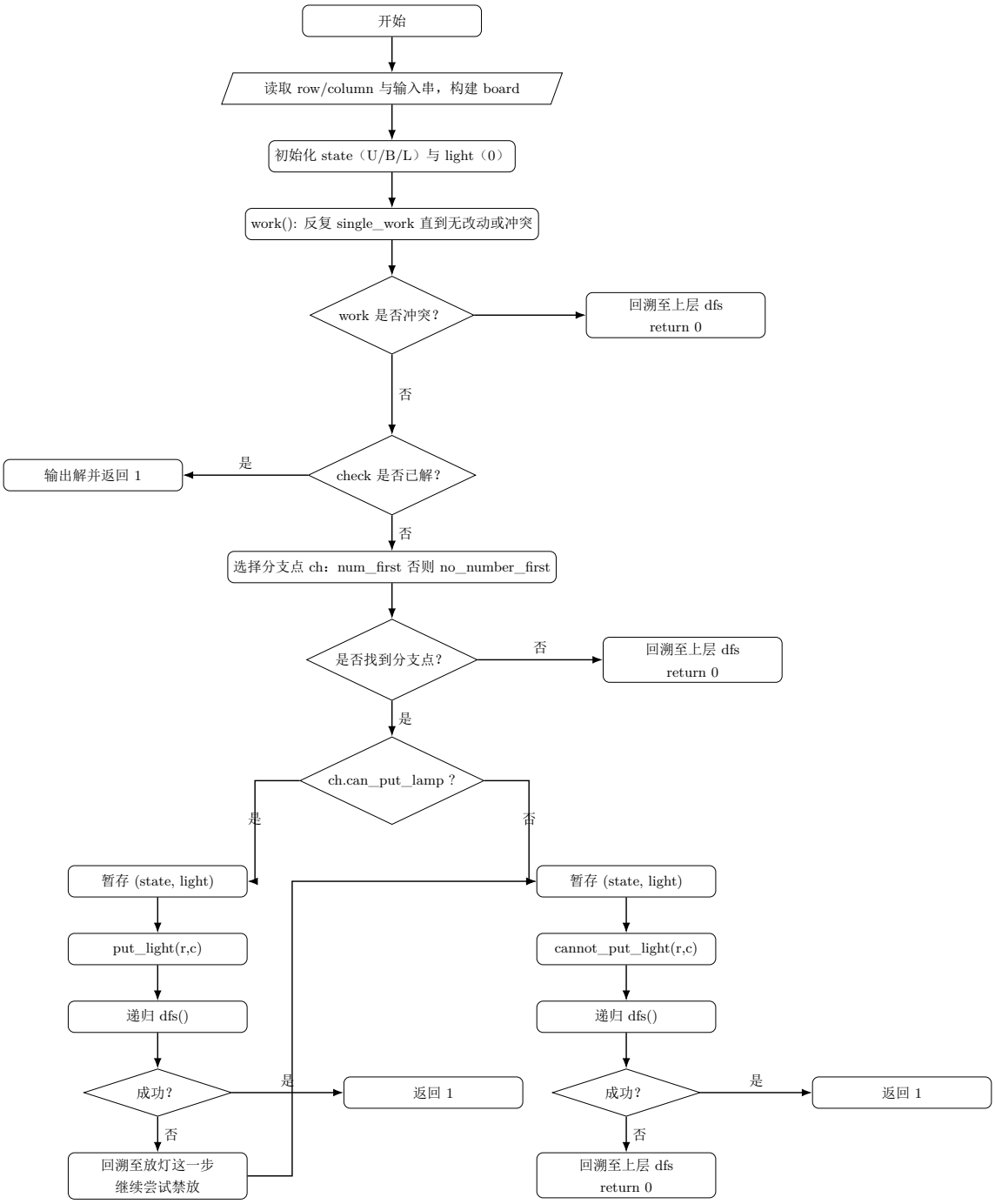

图 1: 流程图