软件工程分析报告

开发任务

编程实现Last Remaining Cell策略和Possible Number策略

函数定义

```
从棋盘现有的确定值出发,如果一个区域(行\列\宫格)当前单元格之外,其余单元格均不能填写某个候选值,则可以确定当前单元格的确定值是此值
@param {Array<Array<number>>} grid 数独棋盘,元素为0-9的数字,0表示未填写
@returns {Array<Array<Array<number>>>} 数独推理结果,9*9*9的三维数组,每个元素为当前单元格的候选值(集)
const lastRemainingCellInference = (grid) => {
}

未确定单元格的候选值应当排除其同行\列\宫格的确定值,据此可以推断出每个未确定单元格的候选值(集)
@param {Array<Array<number>>} grid 数独棋盘,元素为0-9的数字,0表示未填写
@returns {Array<Array<Array<number>>>} 数独推理结果,9*9*9的三维数组,每个元素为当前单元格的候选值(集)
const possibleNumberInference = (grid) => {
}
```

开发要求

- 实现推理策略,不限语言\工具
- 设计测试用例,验证\确保函数正确

分析报告

分类	任务	时间	详情	难点	改进
分析	阅读题目,撰写基本分析文档	0:00-3:47	阅读与课堂上的题目要求,整理开发任务,函数定义,开发要求,为开发建立基本流程	理解任务要求,建立合理的开发流程	完成基本流程后,也可以让ai根据已有流程提出改进建议
分析	了解数独的基本知识	3:47-5:07	编写提示词,向DeepSeek询问数独的基本知识,同时在搜索引擎内查找数独知识,互相验证	确认ai生成信息的正确性	将ai的结果与搜索引擎结果互相对照,尽可能验证信息正确性
分析	了解LRC策略	5:07-7:20	编写提示词,向DeepSeek询问LRC策略,同时在搜索引擎内查找,与题目信息互相验证	确认ai生成信息的正确性,理解LRC策略	将ai的结果与搜索引擎结果互相对照,尽可能验证策略信息正确性,确认对策略的理解无误
分析	了解PN策略	7:20-9:00	编写提示词,向DeepSeek询问PN策略,同时在搜索引擎内查找,与题目信息互相验证	确认ai生成信息的正确性,理解LRC策略	将ai的结果与搜索引擎结果互相对照,尽可能验证策略信息正确性,确认对策略的理解无误
编码	初始化项目	9:20-9:26	初始化项目,配置虚拟python环境,建立解决文件和测试文件	合理构建项目,便于开发	使用git进行版本管理,便于项目拓展与修改
编码	编写代码文件框架	9:26-10:48	编写代码文件类框架,将两个功能函数作为一个sudoku类的两个方法,在类下共用辅助函数		
编码	编写测试文件框架	10:48-13:47	安装pytest库,编写测试文件框架	组织测试形式	使用第三方库,有以文件的形式区分功能实现和测试,从函数名上直接反映测试项目
编码	编写功能函数	13:47-18:20	编写提示词,向DeepSeek获取功能函数	组织提示词,向ai正确传达项目与函数信息	在使用总结通用提示词,在前文咨询数独信息的同意对话中继续补全代码
编码	编写测试用例	18:20-20:37	编写提示词,向DeepSeek获取测试用例	即使ai能正确讲解数独规则,但生成的数独策略用例并非完全正确	通过其他手段(如搜索引擎)获取测试用例,或者开发过程中人工修正测试用例
测试	第一次测试	20:37-20:42	测试获取的测试用例		

分类	任务	时间	详情	难点	改进
调试	更正代码,第二次测试	20:42-21:25	修改代码,重新测试获取的测试用例	测试代码中引入了sudoku类的一个实例,但在 `sudoku.py` 中未创建该实例	关注IDE的提示
测试	添加新测试样例测试	21:25-22:51	从ai生成的测试样例中选择新的测试样例以测试功能函数的正确性		
测试	修正样例结果	22:51-25:12	修正ai生成的样例结果	即使ai能正确讲解数独规则,但生成的数独策略用例并非完全正确	通过其他手段(如搜索引擎)获取测试用例,或者开发过程中人工修正测试用例
调试	查看功能函数逻辑	25:12-28:21	初步更改样例结果测试仍不对,开始检查LRC逻辑是否有误	嵌套数组结构叫较复杂,vscode调试功能难以直观呈现所有信息	通过打印数据的方式进行追踪,或者考略换用功能更强大的IDE
调试	修正样例结果	25:12-31:44	根据调试信息修正样例结果		
测试	添加测试样例	31:44-34:36	从ai生成的测试样例中选择新的测试样例以测试功能函数的正确	即使ai能正确讲解数独规则,但生成的数独策略用例并非完全正确	通过其他手段(如搜索引擎)获取测试用例,或者开发过程中人工修正测试用例
调试	更改LRC逻辑	34:36-37:40	更改LRC逻辑,让LRC在PN的基础上进行确认,以返回候选数集		
调试	修正样例结果	37:40-44:12	根据调试信息修正样例结果	嵌套数组结构叫较复杂,vscode调试功能难以直观呈现所有信息	通过打印数据的方式进行追踪,或者考略换用功能更强大的IDE
测试	确认测试结果	44:12-44:16	修正样例结果后重新进行测试,测试通过		

代码实现

`sudoku.py` 功能实现代码

```python
class sudoku:
    def __init__(self):
        self.size = 9
        self.subgrid_size = 3

    def _is_valid_cell(self, row: int, col: int) -> bool:
        """检查坐标是否有效"""
        return 0 <= row < self.size and 0 <= col < self.size

    def _get_subgrid_range(self, row: int, col: int) -> tuple:
        """获取指定位置所在的3x3子网格的范围"""
        start_row = (row // self.subgrid_size) * self.subgrid_size
        start_col = (col // self.subgrid_size) * self.subgrid_size
        return (start_row, start_col)

    def _get_used_numbers(self, grid: list[list[int]], row: int, col: int) -> set:
        """
        获取指定位置所在行、列和子网格中已使用的数字
        返回: 已使用数字的集合
        """
        used = set()

        # 检查行
        used.update(num for num in grid[row] if num != 0)

        # 检查列
        used.update(grid[r][col] for r in range(self.size) if grid[r][col] != 0)

        # 检查3x3子网格
        start_row, start_col = self._get_subgrid_range(row, col)
        for r in range(start_row, start_row + self.subgrid_size):
            for c in range(start_col, start_col + self.subgrid_size):
                if grid[r][c] != 0:
                    used.add(grid[r][c])
```

```
        return used

    def lastRemainingCellInference(
        self, grid: list[list[int]]
    ) -> list[list[list[int]]]:
        """
        应用最后剩余格策略推断候选数
        先通过PN获取候选数集，如果有能确定的数(候选数集长度为1)，
        则将该位置集合内只有该数，否则为PN获取的候选数集
        返回：每个位置的候选数列表(3D数组)
        """
        # 首先获取所有位置的候选数
        candidates = self.possibleNumberInference(grid)

        # 创建一个新的候选数矩阵用于存储结果
        result = [[[] for _ in range(self.size)] for _ in range(self.size)]

        for num in range(1, 10):
            # 检查每一行
            for row in range(self.size):
                possible_cols = [
                    col
                    for col in range(self.size)
                    if grid[row][col] == 0 and num in candidates[row][col]
                ]
                if len(possible_cols) == 1:
                    col = possible_cols[0]
                    result[row][col] = [num]

            # 检查每一列
            for col in range(self.size):
                possible_rows = [
                    row
                    for row in range(self.size)
                    if grid[row][col] == 0 and num in candidates[row][col]
                ]
                if len(possible_rows) == 1:
                    row = possible_rows[0]
                    result[row][col] = [num]

            # 检查每个3x3子网格
            for sg_row in range(0, self.size, self.subgrid_size):
                for sg_col in range(0, self.size, self.subgrid_size):
                    possible_cells = []
                    for r in range(sg_row, sg_row + self.subgrid_size):
                        for c in range(sg_col, sg_col + self.subgrid_size):
                            if grid[r][c] == 0 and num in candidates[r][c]:
                                possible_cells.append((r, c))
                    if len(possible_cells) == 1:
                        r, c = possible_cells[0]
                        result[r][c] = [num]

        # 合并结果：如果LRC找到了确定数字则使用，否则保留PN的候选数
        for row in range(self.size):
            for col in range(self.size):
                if grid[row][col] == 0:
                    if result[row][col]:  # LRC找到了确定数字
                        pass  # 保留LRC的结果
                    else:
                        result[row][col] = candidates[row][col]  # 使用PN的候选数
        return result

    def possibleNumberInference(self, grid: list[list[int]]) -> list[list[list[int]]]:
        """
        应用候选数策略推断每个位置可能的数字
```

```
        返回: 每个位置的候选数列表(3D数组)
        """
        candidates = [[[] for _ in range(self.size)] for _ in range(self.size)]

        for row in range(self.size):
            for col in range(self.size):
                if grid[row][col] == 0:
                    used_numbers = self._get_used_numbers(grid, row, col)
                    possible = [num for num in range(1, 10) if num not in used_numbers]
                    candidates[row][col] = possible
        return candidates


sudoku_ins = sudoku()
```

`test_sudoku.py` 功能测试代码

```
from sudoku import sudoku_ins

test_PN_grid = [
    [[0] * 9 for _ in range(9)],
    [
        [1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
    ],
]
test_PN_answer = [
    [[list(range(1, 10)) for _ in range(9)] for _ in range(9)],
    [
        [[] for _ in range(9)],  # 第一行已填满
        [
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 4, 5, 6],
            [1, 2, 3, 4, 5, 6],
            [1, 2, 3, 4, 5, 6],
        ],
        [
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 4, 5, 6],
            [1, 2, 3, 4, 5, 6],
            [1, 2, 3, 4, 5, 6],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
```

```
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8],
        ],
    ],
]

test_LRC_grid = [
    [
        [1, 2, 3, 4, 5, 6, 7, 8, 0],  # 最后一格必须是9
```

```
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
        [0] * 9,
    ],
]
test_LRC_answer = [
    [
        [[], [], [], [], [], [], [], [], [9]],
        [
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 9],
            [1, 2, 3, 4, 5, 6, 9],
            [1, 2, 3, 4, 5, 6, 9],
        ],
        [
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 9],
            [1, 2, 3, 4, 5, 6, 9],
            [1, 2, 3, 4, 5, 6, 9],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8,9],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8,9],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
```

```python
            [1, 2, 3, 4, 5, 6, 7, 8,9],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8,9],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8,9],
        ],
        [
            [2, 3, 4, 5, 6, 7, 8, 9],
            [1, 3, 4, 5, 6, 7, 8, 9],
            [1, 2, 4, 5, 6, 7, 8, 9],
            [1, 2, 3, 5, 6, 7, 8, 9],
            [1, 2, 3, 4, 6, 7, 8, 9],
            [1, 2, 3, 4, 5, 7, 8, 9],
            [1, 2, 3, 4, 5, 6, 8, 9],
            [1, 2, 3, 4, 5, 6, 7, 9],
            [1, 2, 3, 4, 5, 6, 7, 8,9],
        ],
    ]
]


def test_possibleNumberInference():
    for gird, answer in zip(test_PN_grid, test_PN_answer):
        assert sudoku_ins.possibleNumberInference(gird) == answer


def test_lastRemainingCellInference():
    for grid, answer in zip(test_LRC_grid, test_LRC_answer):
        assert sudoku_ins.lastRemainingCellInference(grid) == answer
```