

华中科技大学

课程设计报告

题目: 基于 SAT 的百分号数独游戏求解程序

课程名称: 程序设计综合课程设计

专业班级: 2401

学 号: U202490042

姓 名: 马俊豪

指导教师: 李剑军

报告日期: 2025.9.23

计算机科学与技术学院

目录

任务书.....	III
1 引言.....	4
1.1 课题背景与意义.....	4
1.1.1 问题概述与重要性.....	4
1.1.2 广泛的研究与应用背景.....	4
1.1.3 DPLL 算法：核心方法与历史贡献.....	4
1.1.4 本研究的目标与意义.....	5
1.2 国内外研究现状.....	5
1.3 课程设计的主要研究工作.....	6
2 系统需求分析与总体设计.....	8
2.1 系统需求分析.....	8
2.1.1 SAT 求解器.....	8
2.2 系统总体设计.....	10
2.2.1. 主控与界面管理模块（Display）.....	10
2.2.2. CNF 解析处理模块（CNFParser）.....	10
2.2.3. DPLL 求解引擎模块（DPLL-Solver）.....	10
2.2.4. 百分号数独游戏模块（Percent Sudoku）.....	10
3 系统详细设计.....	12
3.1 有关数据结构的定义.....	12
3.1.1 CNF 文件读取和处理相关结构体.....	12
3.1.2 DPLL 过程.....	13
3.1.3 百分号数独.....	13
3.2 主要算法设计.....	13
3.2.1 CNF 文件解析及处理.....	13
3.2.2 DPLL 算法处理.....	14
3.2.3 百分号数独算法处理.....	19
4 系统实现与测试.....	22
4.1 系统实现.....	22
4.1.1 软硬件环境.....	22
4.1.2 存储结构定义.....	22
4.1.3 函数声明及其关系.....	22

4.2 系统测试	24
4.2.1 交互系统展示	24
4.2.2 CNF 文件处理及求解模块测试	25
4.2.3 数独交互界面及功能模块测试	28
4.2.4 算例测试总结	31
5 总结与展望	32
5.1 全文总结	32
5.2 工作展望	32
6 体会	34
参考文献	36
附录	37
SAT. hpp	37
main. cpp	39
display. hpp	39
cnfparser. hpp	43
solver. hpp	46
%_sudoku. hpp	59

任务书

□ 设计内容

SAT 问题即命题逻辑公式的可满足性问题 (satisfiability problem)，是计算机科学与人工智能基本问题，是一个典型的 NP 完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于 DPLL 算法实现一个完备 SAT 求解器，对输入的 CNF 范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

□ 设计要求

要求具有如下功能：

- (1) **输入输出功能：**包括程序执行参数的输入，SAT 算例 cnf 文件的读取，执行结果的输出与文件保存等。(15%)
- (2) **公式解析与验证：**读取 cnf 算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)
- (3) **DPLL 过程：**基于 DPLL 算法框架，实现 SAT 算例的求解。(35%)
- (4) **时间性能的测量：**基于相应的时间处理函数（参考 time.h），记录 DPLL 过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)
- (5) **程序优化：**对基本 DPLL 的实现进行存储结构、分支变元选取策略^[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为： $[(t-t_0)/t]*100\%$ ，其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t₀ 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)
- (6) **SAT 应用：**将数独游戏^[5]问题转化为 SAT 问题^[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为 SAT 问题的具体方法可参考文献[3]与[6-8]。(15%)

1 引言

1.1 课题背景与意义

1.1.1 问题概述与重要性

布尔可满足性问题（Propositional Satisfiability Problem, SAT）是逻辑与计算机科学中的一个核心难题，其本质是判定是否存在一组对变量的赋值，使得给定的布尔公式的最终结果为真。该问题不仅在理论计算机科学中占据基础性地位，更在硬件验证、软件测试、人工智能等诸多实际应用领域中发挥着不可或缺的关键作用。随着信息技术的持续发展，对 SAT 问题进行高效求解的需求变得愈发迫切。

1.1.2 广泛的研究与应用背景

SAT 问题之所以长期吸引研究者的广泛关注，在于其极强的表达能力和规约特性。大量复杂的组合问题，例如图着色、顶点覆盖和团检测等，均可转化为命题逻辑公式，并通过调用 SAT 求解器得以有效解决。在自动定理证明领域，SAT 也扮演着重要角色，其成功案例包括著名数学猜想——凯勒猜想的证明。

在工业界，SAT 求解技术已成为许多关键流程的支柱。它被广泛应用于集成电路设计中的有界模型检查、配置管理、等效性检查，以及逻辑综合等任务。事实上，许多高效的 SAT 求解器正是为满足这些特定应用场景的需求而专门设计的。因此，对 SAT 问题的研究不仅具有理论价值，更能为工业界带来显著的经济效益与效率提升。

1.1.3 DPLL 算法：核心方法与历史贡献

鉴于 P 与 NP 问题尚未解决，研究者在充分认识到 SAT 问题计算复杂性的同时，始终致力于开发高效的求解算法。其中，Davis - Putnam - Logemann - Loveland (DPLL) 算法因其开创性的贡献和基础性地位而在 SAT 求解史上具有里程碑意义。该算法通过递归分解、系统性回溯搜索，并巧妙地结合了“单位传播”和“分裂启发式”等关键技术，极大地缩减了搜索空间和计算复杂度。面对日益复杂和大规模的 SAT 实例，研究人员对 DPLL 算法进行了持续且深入的优化，这些工作奠定了现代 SAT 求解技术发展的坚实基础。

1.1.4 本研究的目标与意义

本课题致力于研究与优化基于 DPLL 算法的 SAT 求解器，具有重要的理论意义和实际应用价值。从理论意义方面来看，DPLL 算法作为 SAT 求解的基础性框架，其性能直接决定了解决各类 SAT 问题的效率。通过深入探究该算法在处理大规模实例时的内在机理与性能表现，能够为算法的进一步改进与创新提供坚实的理论支撑。

从实践意义方面来看，对 DPLL 算法的深入研究与优化，不仅直接推动了 SAT 求解技术本身的进步，其成功经验更可迁移至硬件设计验证、自动推理、形式化方法等众多工程应用领域，为实现更高效、更准确的问题解决方案提供核心动力。

1.2 国内外研究现状

SAT 问题自 20 世纪 60 年代起逐渐成为研究热点，因其在理论与应用方面的重要意义，持续受到国内外学术界的广泛关注。自 2002 年开始，国际 SAT 竞赛定期举办，旨在推动高性能求解器的研发，并通过公开源代码促进技术交流与共享，显著加速了 SAT 问题的研究进展。以下是 SAT 问题研究的主要发展历程：

1960 年，Davis 和 Putnam 提出了 DPLL 算法，这是首个用于求解 SAT 问题的完备算法，至今仍是大多数精确 SAT 求解器的核心基础。

1962 年，Martin Davis、George Logemann 和 Donald W. Loveland 共同提出 DPLL 算法的改进版本，确立了基于树搜索和回溯的变量赋值枚举框架。

1971 年，Stephen Arthur Cook 从计算理论层面证明了 SAT 是 NP 完全问题，为其复杂性奠定了重要理论基础。

1992 年，Bart Selman 等人提出 GSAT 算法，将贪心局部搜索策略引入 SAT 求解中。

1997 年，李初民教授开发出 satZ 求解器，首次在分支启发策略中系统应用单子句传播技术。

1998 年，梁东敏提出带子句加权的 WSAT 算法，进一步改进了局部搜索性能。

1999 年，João Marques Silva 等人在 GRASP 算法中首次引入冲突学习和非

时序回溯机制，显著提升 DPLL 框架的效率。

2000 年，金人超与黄文奇提出并行 Solar 算法；2002 年，张德富提出将模拟退火算法应用于 SAT 求解。

2003 年，Niklas Eén 开发出 MiniSAT 求解器。该求解器结构简洁、性能高效，成为后来许多求解器的重要基础。

2005 年，Niklas Eén 等人提出 SatElite 预处理工具，能够有效简化问题规模、降低求解复杂度。

2007 年，Bart Selman 和 Henry Kautz 对 SAT 研究现状进行了系统性总结与展望。

2009 年，Simon 与 Laurent 等人提出冲突驱动子句学习（CDCL）机制，极大推动了现代 SAT 求解器的发展。同年，Audemard 等人开发出 Glucose 求解器，并提出以文字块距离（LBD）衡量学习子句质量，成为后续重要评价指标。

2014 年，Oh 基于 Glucose 2.3 开发出 SWDiA5BY 求解器。

2016 年，Liang 等人提出的 MapleCOMSPS 求解器在当年 SAT 竞赛主赛道中荣获第一名。

2018 年，Devriendt 研发的 InIDGlucose 求解器重点关注初始变量选择策略的优化。

2022 年，华中科技大学何琨教授团队提出将随机游走策略与决策模型相结合的新方法，通过根据算例特征自适应调整搜索策略，显著提升了求解器的鲁棒性和泛化能力。

至今，SAT 竞赛仍在持续推动各类新算法的涌现，不断促进 SAT 问题研究向前发展。

1.3 课程设计的主要研究工作

本课程设计围绕布尔可满足性问题（SAT）展开，主要研究工作包括理论学习和实践应用两个层面：一方面系统研究 SAT 问题的基础理论与求解算法，另一方面探索将实际游戏问题转化为 SAT 问题并借助自研求解器进行求解。具体研究内容

如下：

- (1) SAT 问题的理论研究：系统梳理 SAT 问题的基本概念、理论背景及其在学术与工业界的研究意义，总结当前主流研究现状与发展趋势，为后续算法实现与应用奠定理论基础。
- (2) DPLL 算法的理解与实现：深入理解 DPLL 算法的基本原理与关键技术，包括单位传播和变量分支启发式策略等，在此基础上自主设计与实现一个具备基本功能的 DPLL 求解器。
- (3) 算法性能测试与优化：通过多组不同规模与复杂度的 SAT 实例对求解器进行测试，从求解正确率与运行效率等方面评估其性能。根据测试结果分析算法瓶颈，提出并实施相应改进策略，以提升求解能力与适用范围。
- (4) 百分号线数独的 SAT 建模与求解：针对百分号线数独这一约束满足问题，建立其 SAT 表示模型。通过为每个单元格的可能取值定义布尔变量，将数独中的行、列、宫及两条主百分号线约束转化为合取范式（CNF）形式的布尔公式。最终将生成的问题实例输入自研 SAT 求解器，通过 DPLL 算法求解并获得数独的有效解。

2 系统需求分析与总体设计

2.1 系统需求分析

2.1.1 SAT 求解器

SAT 求解器应具有以下功能：

- (1) 输入交互与容错处理：系统支持用户通过参数选择功能操作，包括加载 CNF 文件、显示公式内容、执行求解及返回等。用户需指定 CNF 文件路径，程序将其解析并存入结构化的 CNF 表达式模型中。系统集成输入校验机制，可识别并阻止非法操作，例如在未成功载入文件时尝试调用求解功能。
- (2) 结果输出与性能报告：求解完成后，系统将 SAT 求解结果（可满足性判定结果、变量赋值情况）同时写入与原 CNF 文件同名的结果文件，并实时在界面中显示。输出内容还包括关键性能指标，如求解耗时、优化策略的有效性统计等，为用户提供清晰的性能反馈。
- (3) 可配置的 DPLL 求解模块：系统提供基于 DPLL 算法的核心求解功能，允许用户在执行求解前动态选择变元分支策略，如随机选择、最多出现频次等启发式方法，以适配不同结构的问题实例。
- (4) 用户界面与交互体验：系统配备控制台交互界面，通过菜单驱动和功能模块的清晰划分实现用户操作引导。借助定时清屏、格式化输出和操作状态提示，提供流畅、直观且稳定的用户体验。

2.1.2 百分号数独游戏设计

1. 数独生成模块

采用**挖洞算法**随机生成符合规则的百分号数独初盘。生成过程中需确保布局满足百分号数独的特定约束条件，即除常规九宫格、行、列约束外，还需满足两个百分号窗口内以及副对角线的数字不重复。

2. 用户交互与游戏控制

支持用户通过指定坐标（行号和列号）在允许的空格中填入数字，并提供多项操作功能：

- (1) 游玩数独，包括：插入及删除数字，并最终提交游戏；
- (2) 重置游戏至初始状态；
- (3) 请求查看标准答案；
- (4) 调用 SAT 求解器对当前谜题进行求解。

3. 输入校验机制

对用户输入进行多维度实时检查，包括：

- (1) 验证填入数字是否在 1 到 9 的有效范围内；
- (2) 判断输入位置是否为初始生成的可填空格（禁止覆盖初始数字）；
- (3) 检测是否违反百分号数独的多重约束规则，包括行、列、九宫格及两个百分号窗口内以及副对角线上的数字唯一性。

4. 图形化控制台界面

设计清晰美观的文本图形界面，功能包括：

- (1) 通过符号和格式优化（如分块显示、百分号路径视觉强化）清晰呈现数独盘面；
- (2) 集成 system 指令与返回上级菜单选项，提升界面整洁度与操作流畅性。

5. SAT 集成求解功能

实现将百分号数独问题自动转化为 SAT 问题的能力：

- (1) 将数独中所有数字约束（包括常规规则和百分号特殊约束）编码为合取范式（CNF），并生成对应的 CNF 文件；
- (2) 调用已实现的 DPLL 求解器对 CNF 文件进行求解；
- (3) 将求解结果（变量赋值）反向映射为数字填入矩阵，并以可读形式输出最终解。

2.2 系统总体设计

本系统需实现一个基于菜单驱动的 SAT 求解器集成开发环境，并包含百分号数独游戏功能模块。系统采用模块化设计，除主控模块外，各功能模块均通过.h 头文件实现文件进行封装。具体模块划分如下：

2.2.1. 主控与界面管理模块（Display）

实现文件：display.hpp

核心功能：

本系统通过 while 循环和 switch-case 结构构建多级菜单系统，集成 system("pause") 和 system("cls")实现界面控制，根据用户输入参数调度各功能模块执行，并维护全局状态变量和系统运行环境。

2.2.2. CNF 解析处理模块（CNFParser）

实现文件：cnfparser.hpp

核心功能：

读取并解析 DIMACS 格式的 CNF 文件，将其转换为内存中的二级链表结构（子句链表+文字链表），并提供 CNF 公式打印与内存释放的配套操作。

2.2.3. DPLL 求解引擎模块（DPLL-Solver）

实现文件：solver.hpp

核心功能：

实现了完整的 DPLL 算法框架，包含单元传播、子句简化、四种变量选择启发式策略（随机选择、频次统计、最小子句优先和 MOMS 策略）、递归回溯求解以及带超时检测的 SAT 求解功能。

2.2.4. 百分号数独游戏模块（Percent Sudoku）

本模块由四个子模块构成，实现完整的数独游戏生命周期管理：

(1) 数独生成引擎：

数独生成引擎采用 DPLL 递归回溯算法生成初始完整盘面，然后再通过在左上和右下两个 3×3 的百分号窗口区域预置随机数字作为初始约束，再通过挖洞法移除指定数量的数字来生成可解的数独题目。

(2) SAT 问题归约模块：

该模块负责将数独游戏盘面转换为 CNF 格式，然后通过预定义的模板文件将数独的各类约束条件（包括常规规则和特殊百分号区域约束）编码为合取范式，并将转换结果输出保存至指定 CNF 文件。

(3) 数独求解接口：

该模块封装了 DPLL 求解器的调用接口，通过解析求解结果将布尔变量赋值映射为 1-9 的数字解，并优化输出仅显示有效的数独数字赋值。

(4) 游戏交互验证模块：

该模块实现了游戏交互验证功能，包括坐标有效性检查（1-9 范围）、初始数字保护（禁止修改预设好了的数字），以及即时验证行/列/九宫格唯一性规则和两个百分号窗口及副对角线的数字唯一性约束，而且提供了详细的错误提示和重新输入引导。

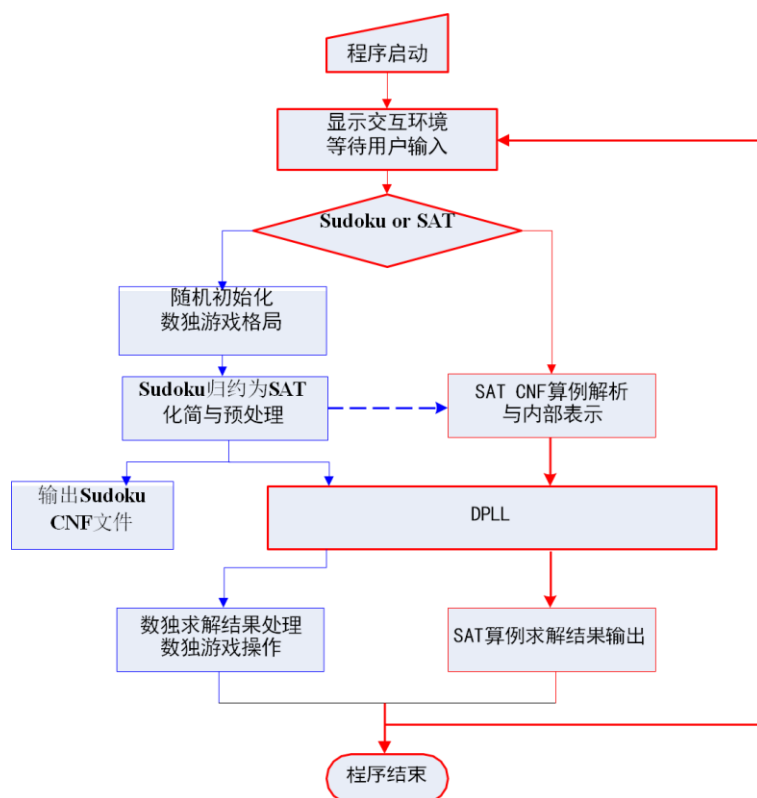


图 2-1 程序流程图

3 系统详细设计

3.1 有关数据结构的定义

3.1.1 CNF 文件读取和处理相关结构体

(1) 对于文字节点需要一个结构体 `literal_node`来定义: 数据项有 `int` 型的文字值 `literal`; `literal_node*`型的指向下一个文字节点的指针 `*next`。

(2) 对于子句节点需要一个结构体 `clause_node`来定义: 数据项有 `literal_node*`型的指向该子句第一个文字节点的指针 `head`; `clause_node*`型的指向下一个子句节点的指针 `*next`。

(3) 用一个结构体`cnf_node`来定义CNF 范式链表, 用于存储 CNF 文件信息: 数据项有 `int` 型的布尔变元个数 `bool_count`; `int` 型的子句个数 `clause_count`; `cnf_node*`型的指向第一个子句节点的指针 `root`。

表 3.1-1 对 CNF 文件读取和处理相关结构体

数据名称	作用	数据内容s
<code>literal_node</code>	文字节点	<code>int literal;</code> <code>struct literal_node* next;</code>
<code>clause_node</code>	子句节点	<code>literal_list head;</code> <code>struct clause_node *next;</code>
<code>cnf_node</code>	CNF 范式链表	<code>int bool_count;</code> <code>int clause_count;</code> <code>clause_list root;</code>

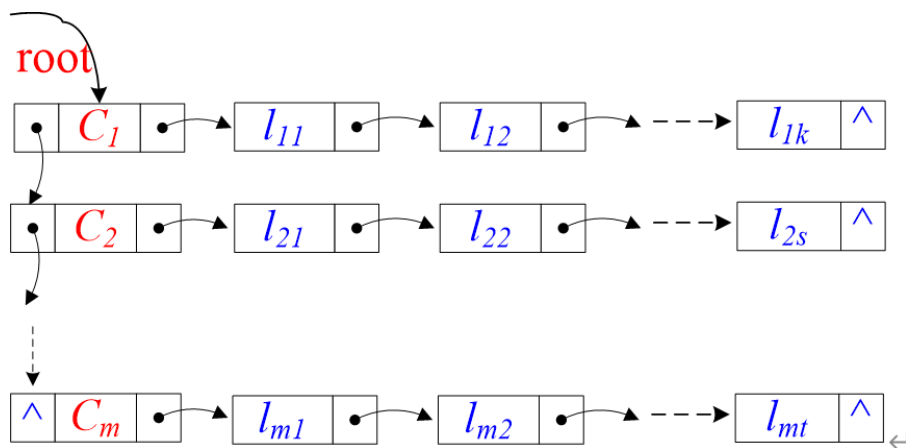


图3.1-1 CNF 文件存储结构图

3.1.2 DPLL过程

- (1) CNF（合取范式）：以链表形式存储子句（`clause_list`），每个子句包含一个文字链表（`literal_node`），表示逻辑公式的合取结构；
- (2) 布尔赋值数组（`value[]`）：记录每个文字的真值（`true/false`），用于跟踪 DPLL 搜索过程中的变量赋值状态。

3.1.3 百分号数独

这个部分需要一个 `int` 类型的数组来记录数独棋盘每个位置的值，同时需要布尔赋值数组记录每个文字的真值（`true/false`），用于跟踪 DPLL 搜索过程中的变量赋值状态。

3.2 主要算法设计

3.2.1 CNF 文件解析及处理

1. 根据 CNF 文件创建 CNF 范式链表

```
bool read_file(char file_name[], CNF &cnf);
```

参数: (CNF) &cnf: 指向 CNF 范式链表的指针

(char) file_name: 文件名称

返回值: 1: 创建成功

0: 创建失败

这段函数打开文件名为 fileName[]的文件后, 先读入文件中的无关信息, 再读入布尔变元个数和子句个数, 最后依次创建子句节点和文字节点。

2. 销毁范式链表

```
void destroy_cnf(CNF &cnf);
```

参数: (CNF) &cnf: 指向 CNF 范式链表的指针

这段函数用 while 循环销毁创建的范式链表, 首先销毁文字链表, 然后再销毁子句链表, 同时释放各节点存储空间。

3. 遍历范式链表

```
void print_cnf(CNF cnf);
```

参数: (CNF) cnf: 指向 CNF 范式链

这段函数首先检验 CNF文件的解析与处理是否正确, 然后输出布尔变元个数, 子句个数, 最后再打印每个子句及文字的信息。

3.2.2 DPLL 算法处理

1. 找到单子句并返回其文字节点的文字值

```
int find_unit_clause(clause_list cl);
```

参数: (clause_list) cl: 指向范式链表root子句节点的指针

返回值: 非 0: 找到单子句, 且返回值为单子句的文字内容

0: 没有找到单子句

这段函数首先遍历子句链表，检查每个子句是否只包含一个文字（即单子句），如果找到则立即返回该文字的值，否则在遍历完所有子句后返回0表示未找到单子句。

2. 根据选择的文字化简链表

```
void simplify(clause_list &cl, int lit);
```

参数: (clause_list) & cl: 指向范式链表root节点的指针

(int) lit: 要用来化简链表的文字

这段函数实现了CNF公式的简化处理，它会遍历子句链表，当遇到包含目标文字(lit)的子句时直接删除整个子句，遇到包含该文字否定形式(-lit)的子句则只删除对应文字，其他情况保持不变，从而完成公式的化简操作。

3. 判断当前链表是否满足 (DPLL 递归的出口)

```
bool is_satisfy(clause_list cl);
```

参数: (clause_list) cl: 指向范式链表root的指针

返回值: 1: 满足

0: 不满足

这段函数通过检查子句链表是否为空来判断CNF公式是否已被满足：当链表为空时返回1（表示所有子句已被满足），否则返回0（表示仍有未满足的子句）。

4. 判断当前链表是否有空子句 (DPLL 的另一个递归出口)

```
bool is_empty_clause(clause_list cl);
```

参数: (clause_list) cl: 指向范式链表root节点

返回值: 1: 有空子句

0: 没有空子句

这段函数通过遍历子句链表来检查是否存在空子句（head指针为空的子句），如果找到空子句则返回1（真），否则遍历完所有子句后返回0（假）。

5. 将当前链表复制一份(用于 DPLL 的一个分支)

`clause_list copy_cnf(clause_list cl);`

参数: (clause_list) cl: 指向范式链表root节点的指针

返回值: (clause_list): 指向复制的范式链表第一个子句节点的指针

这段代码实现了CNF公式的深拷贝功能，它会递归遍历原始子句链表和每个子句的文字链表，为每个节点创建新的内存空间并复制对应的文字值，最终返回一个结构与内容完全相同但内存独立的新CNF公式副本，属于一个深拷贝操作。

6. 没有单子句时选择变元的策略(朴素方法)

`int choose_literal_1(CNF cnf);`

参数: (CNF) cnf: 指向 CNF 范式链表的指针

返回值: (int)型文字

随机在当前链表(链表中剩下的文字都未被赋值)中选择一个文字。

7. 没有单子句时选择变元的策略(优化一)

`int choose_literal_2(CNF cnf);`

参数: (CNF) cnf: 指向 CNF 范式链表的指针

返回值: (int)型文字

这段函数实现了基于文字出现频率的选择策略，首先它会统计CNF公式中每个文字及其否定形式出现的总次数，然后再选择出现次数最多的文字作为分支变量，若正文字出现次数相同则优先选择正文字，最终返回出现频率最高的文字值。

8. 没有单子句时选择变元的策略(优化二: MOMS 算法)

`int choose_literal_3(CNF cnf);`

参数: (CNF) cnf: 指向 CNF 范式链表的指针

返回值: (int)型文字

这段函数实现了基于最小子句的文字选择策略, 它会先找出长度最短的子句, 然后统计该子句中各文字的出现频率, 最终选择在最小子句中出現次数最多的文字作为分支变量返回。

9. 没有单子句时选择变元的策略(优化三: MOMS 进阶算法)

`int choose_literal_4(CNF cnf);`

参数: (CNF) cnf: 指向 CNF 范式链表的指针

返回值: (int)型文字

这段函数会先找出所有长度最短的子句, 然后统计这些子句中每个变量及其否定形式的出现次数, 最终选择在最短子句中出现总次数最多的变量(优先选择正文字出现次数多的)作为分支文字返回。

10. 采用递归思路的 DPLL 算法

`status DPLL(CNF cnf, bool value[], int flag);`

参数: (CNF) cnf: 指向 CNF 范式链表的指针

(bool *) value: 用来存布尔变元真值的数组(true or false)

(int) flag: 用来控制选择哪种变元选取策略

(const std::chrono::steady_clock::time_point) &start

(double) timeout_seconds

返回值: 1: DPLL 成功, 该 SAT 问题有解

0: DPLL 失败, 该 SAT 问题无解

-1: DPLL超时，该SAT解法无效

这段函数实现了一个带超时控制的DPLL算法变体，它在递归求解过程中会持续检查是否超过预设的时间限制，若超时则立即返回-1终止计算，否则继续执行标准的DPLL流程（包括单元传播、分支选择等），并在每次关键操作前都进行超时检测以保证算法能在规定时间内返回结果。

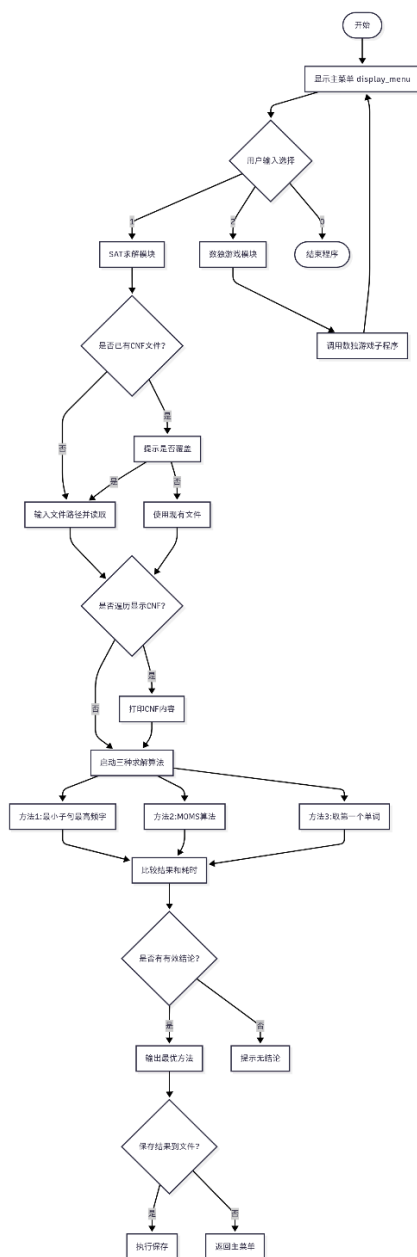


图3.2.2-1 DPLL 函数流程图

3.2.3 百分号数独算法处理

1. 生成初始数独

```
int fixed_board[SIZE + 1][SIZE + 1] = {0}; // 生成的数独
int answer_board[SIZE + 1][SIZE + 1] = {0}; // 答案数独
int play_board[SIZE + 1][SIZE + 1] = {0}; // 用来玩的数独
bool is_num[SIZE + 1][SIZE + 1]; // 判断数独某个位置存不存在提示数字
bool value[SIZE * SIZE * SIZE + 1] = {0}; // 存储数独变元真值（返回值：1：
生成成功；0：生成失败）
```

该函数首先生成一个包含特定初始模式的数独（在percentA和percentB位置填充随机排列的数字），然后使用DPLL算法求解出一个完整的数独答案。接着采用挖洞法，根据指定的提示数hint，均匀地在每行移除相应数量的数字，最终将结果分别保存在fixed_board（固定数字）、play_board（可玩棋盘）和answer_board（完整答案）三个棋盘中。

2. 判断填入的数字是否有效

```
bool is_valid(int hang, int lie, int value, int play_board[SIZE + 1][SIZE + 1]);
```

参数：(int) board[SIZE + 1][SIZE + 1]：初始数独

(int) hang：行号

(int) lie：列号

(int) value：插入的值

返回值：1：有效

0：无效

该函数用于检查在数独棋盘指定位置填入某个数字是否有效，它会依次检查该数字是否违反行、列、3×3宫格、副对角线（若在对角线上）以及两个特定3×3窗口（左上和右下）的重复规则，若发现任何冲突则返

回错误提示和false，否则返回true表示该数字可以合法填入。

3. 用户玩数独的交互界面

```
void play_sudoku(int answer_board[SIZE + 1][SIZE + 1], int
(&play_board)[SIZE + 1][SIZE + 1], bool is_num[SIZE + 1][SIZE + 1], int
fixed_board[SIZE + 1][SIZE + 1])
```

参数：int (&play_board)[SIZE + 1][SIZE + 1]：用来交互的数独

bool is_num[SIZE + 1][SIZE + 1]：是否为提示数

int fixed_board[SIZE + 1][SIZE + 1]：初始数独

int answer_board[SIZE + 1][SIZE + 1]：答案数独

这段代码实现了一个交互式数独游戏的主循环，它会持续显示当前棋盘状态并等待玩家输入，支持以下功能：玩家可以指定位置填入数字（会进行有效性检查）、查看完整答案、退出游戏或提交最终答案（会检查是否完成），并在每次操作后提供相应的提示信息，直到玩家选择退出或成功完成数独为止。

4. 将数独规约为 SAT 问题并写入文件

```
bool write_file(int (&board)[SIZE + 1][SIZE + 1], int num, char name[]);
```

参数：int (&board)[SIZE + 1][SIZE + 1]：初始数独

int num：提示数个数

char name[]：要写入的文件名

返回值：1：写入成功

0：写入失败

该函数将数独规则转化为CNF格式文件，首行声明变量数（729）和子句总数（初始提示数num加上固定值12654），随后依次写入包含初始提示

数字、每个格子必须且只能填一个数字、每行每列每宫必须包含1-9且不重复、副对角线需满足数独规则等约束条件，其中每个数独位置(i,j)填入数字k通过公式 $(i-1)81+(j-1)9+k$ 编码为1~729的连续变量

5. 求解数独

```
bool solve_sudoku(int (&board)[SIZE + 1][SIZE + 1], bool (&value)[SIZE * SIZE * SIZE + 1]);
```

参数: int (&board)[SIZE + 1][SIZE + 1]: 保存结果的数独

bool (&value)[SIZE * SIZE * SIZE + 1]: 存储数独变元真值

返回值: 1: 求解成功

0: 求解失败

这段代码的作用是根据一个布尔数组 value 的标记情况，将对应的数字填入数独棋盘 board 的相应位置。具体来说，代码会遍历 value 数组的每一个元素，当遇到某个位置的值为 true 时，就计算出该位置对应的数独棋盘的行号、列号以及要填入的数字，然后将这个数字写入棋盘。其中，行号的计算方式是将索引 i 减去 1 后除以 SIZE 的平方（即 $SIZE * SIZE$ ），再加 1；列号的计算方式是将索引 i 减去 1 后除以 SIZE，再对 SIZE 取模，最后加 1；数字的计算方式是将索引 i 减去 1 后对 SIZE 取模，再加 1。最终，函数返回 1，表示所有有效的数字都已经成功填入棋盘。

4 系统实现与测试

4.1 系统实现

4.1.1 软硬件环境

1. 硬件环境:

处理器: 12th Gen Intel Core i7-12700H

机带 RAM: 16GB

2. 软件环境

Windows11

IDE: Visual Studio Code

编译器: mingGW

项目管理: CMake

4.1.2 存储结构定义

```
typedef struct literal_node // 子句中的文字的链表
{
    int literal;
    struct literal_node *next;
} literal_node, *literal_list;
typedef struct clause_node // 每个子句的链表
{
    literal_list head;
    struct clause_node *next;
} clause_node, *clause_list;
typedef struct cnfNode // 总的 cnf 文件链表
{
    clause_list root;
    int bool_count;
    int clause_count;
} cnf_node, *CNF;
```

4.1.3 函数声明及其关系

```
// 显示主界面
```

```
void main_display();
```

```
// 显示菜单
```

```

void display_menu();
// 数独主流程
void main_sudoku();
// 显示数独界面
void display_sudoku();
// 生成数独题目
void generate_sudoku(int (&fixed_board)[SIZE + 1][SIZE + 1], int (&answer_board)[SIZE + 1][SIZE + 1], int (&play_board)[SIZE + 1][SIZE + 1], int hint, bool value[SIZE * SIZE * SIZE + 1], bool (&is_num)[SIZE + 1][SIZE + 1]);
// 打印棋盘
void print_board(int board[SIZE + 1][SIZE + 1]);
// 数独游戏交互
void play_sudoku(int answer_board[SIZE + 1][SIZE + 1], int (&play_board)[SIZE + 1][SIZE + 1], bool is_num[SIZE + 1][SIZE + 1], int fixed_board[SIZE + 1][SIZE + 1]);
// 检查数字是否合法
bool is_valid(int hang, int lie, int value, int play_board[SIZE + 1][SIZE + 1]);
// 写CNF文件
bool write_file(int (&board)[SIZE + 1][SIZE + 1], int num, char name[]);
// 打乱数组
void shuffle(int (&arr)[], int size);
// 用SAT解数独
bool solve_sudoku(int (&board)[SIZE + 1][SIZE + 1], bool (&value)[SIZE * SIZE * SIZE + 1]);
// 读取CNF文件
bool read_file(char file_name[], CNF &cnf);
// 释放CNF链表
void destroy_cnf(CNF &cnf);
// 打印CNF链表
void print_cnf(CNF cnf);
// 查找单子句
int find_unit_clause(clause_list cl);
// 移除子句
void remove_clause(clause_list &cl);
// 化简子句
void simplify(clause_list &cl, int lit);
// 判断是否满足
bool is_satisfy(clause_list cl);

```



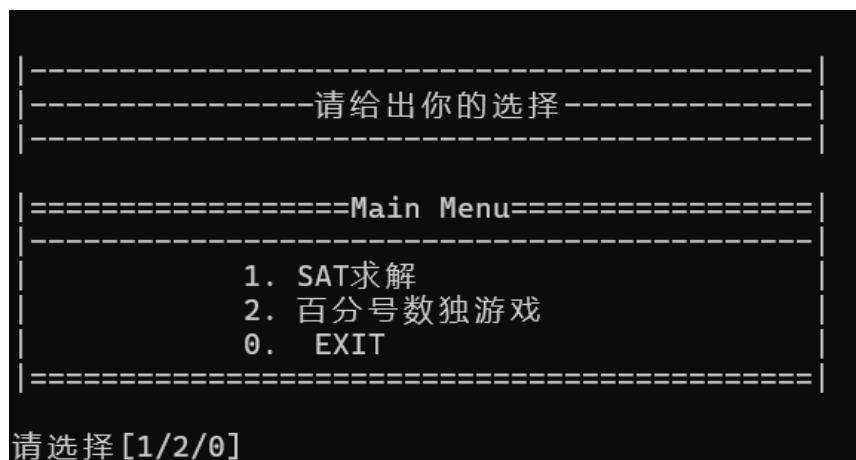
```
// 判断是否有空子句
bool is_empty_clause(clause_list cl);
// 选择分支变量1
int choose_literal_1(CNF cnf);
// 选择分支变量2
int choose_literal_2(CNF cnf);
// 选择分支变量3
int choose_literal_3(CNF cnf);
// 拷贝子句链表
clause_list copy_cnf(clause_list cl);
// 保存SAT解
bool save_file(int result, char file_name[], double time, bool value[],
    int bool_count, double time_);
// 基本DPLL算法
bool DPLL(CNF cnf, bool value[], int flag);
// 支持超时的DPLL
int DPLL_2(CNF cnf, bool value[], int flag, const std::chrono::steady_c
    lock::time_point &start, double timeout_seconds);
// 按flag和超时求解, 返回耗时
int solve_with_timeout_flag(CNF cnf, bool value[], int flag, double tim
    eout_seconds, double &elapsed_seconds);
```

4.2 系统测试

4.2.1 交互系统展示

主交互系统界面和百分号数独交互系统界面分别如图 4.2.1-1 和图

4.2.1-2 所示:



```
|-----|
|-----请给出你的选择-----|
|-----|
|=====Main Menu=====|
|-----|
|          1. SAT求解          |
|          2. 百分号数独游戏    |
|          0.  EXIT            |
|=====|
|
| 请选择 [1/2/0]
```

图4.2.1-1 主交互系统界面

```

*****Menu for Percent_Sudoku*****
-----
1. 生成数独
2. 查看答案
3. 游玩数独
0. EXIT
*****
请选择 [1/2/3/0]
    
```

图4.2.1-2 百分号数独交互系统界面

4.2.2 CNF 文件处理及求解模块测试

1. 读入 cnf 文件并询问是否需要遍历输出，若需要，则在主交互界面输入 1，并遍历打印cnf文件内容，然后开始使用三种方法对cnf文件进行求解，并计算打印出每个方法的求解时间，同时比较出最快方法，之后再询问是否需要保存文件，若需要则输入1，并保存文件。

```

-----请给出你的选择-----
=====Main Menu=====
1. SAT求解
2. 百分号数独游戏
0. EXIT
=====
请选择 [1/2/0]
1
请输入文件地址
test/1.cnf
读取成功！
需要我遍历一遍cnf文件吗 [1/0]
    
```

图4.2.2-1 文件读取与解析

```

需要我遍历一遍cnf文件吗[1/0] 28 50 -122 0
1 29 -103 176 0
The CNF is: -55 97 -146 0
boolCount:200 -2 44 70 0
clauseCount:1200 -55 -139 -159 0
108 113 158 0 70 103 -195 0
-108 158 165 0 140 143 -178 0
113 158 -165 0 64 -124 195 0
45 88 158 0 -90 -112 154 0
45 84 158 0 167 200 0
88 -113 158 0 61 -73 -198 0
-88 -113 158 0 -26 -59 100 0
45 133 -158 0 -6 -95 -136 0
100 -133 -158 0 1 -188 -198 0
45 -100 -133 0 -7 112 -127 0
-15 53 141 0 28 -52 -159 0
-15 -53 141 0 -90 94 157 0
-15 -53 141 0 4 29 -117 0
-15 -33 -53 0 14 52 102 0
-33 -53 -81 0 23 59 103 0
-15 -45 -141 0 84 -136 160 0
-2 -15 -141 0 52 -157 -180 0
15 -45 -57 0 -19 -101 -146 0
-45 -57 -184 0 -7 97 102 0
-8 14 -45 0 149 167 -186 0
-8 -45 59 0 46 52 -124 0
-8 -57 -59 0 -90 -122 148 0
-45 57 106 0 -4 46 100 0
47 57 -106 0 28 123 153 0
-13 -47 -106 0 -25 28 160 0
-13 -47 -106 0 167 -193 198 0
54 -93 -106 0 4 -46 52 0
-54 -93 -106 0 -74 183 -188 0
-54 -93 -106 0 48 -180 -199 0
15 -49 93 0 -7 -76 -199 0
13 49 169 0 64 -76 102 0
-27 49 169 0 -7 137 157 0
13 15 -191 0 4 136 0
13 -169 -175 0 32 64 130 0
15 57 -178 0 102 -126 -150 0
13 -16 93 0 -7 -86 157 0
51 -143 175 0 28 -96 -198 0
-51 -143 175 0 -7 130 147 0
13 -30 49 0 -111 157 0
30 144 178 0 106 112 -199 0
30 50 178 0 -7 -86 157 0
30 50 178 0

```

图4.2.2-2&4.2.2-3 cnf 文件遍历

```
s 1
v
1 -2 -3 -4 -5 -6 7 -8 9 10 11 -12 -13 14 -15 -16 -17 18 -19 20 21 -22 -23 -24 -25 -26
27 -28 -29 -30 -31 32 33 -34 35 -36 37 -38 -39 40 41 -42 -43 -44 45 -46 47 48 -49
50 -51 -52 53 54 -55 -56 -57 58 59 60 61 62 63 64 -65 66 67 -68 -69 -70 -71 72
73 -74 -75 -76 -77 -78 -79 80 -81 -82 -83 84 -85 -86 -87 -88 -89 90 91 -92 -93 94
95 -96 97 98 99 -100 101 -102 -103 -104 105 106 107 108
109 -110 -111 -112 -113 -114 115 -116 -117 118 119 -120 -121 122 -123 -124 125
126 -127 -128 -129 -130 131 -132 -133 -134 135 136
137 -138 -139 -140 -141 -142 -143 144 -145 146 147 148 149 -150 -151 -152
153 -154 -155 -156 -157 158 -159 -160 -161 -162 163 164 165 166 -167 168 169
170 -171 -172 173 -174 -175 -176 -177 -178 -179 180 181 -182 -183 -184 -185
186 -187 -188 -189 -190 -191 -192 -193 194 -195 196 -197 -198 199 200
t 108.047000ms
```

图4.2.2-4 将结果保存至.res 文件中

```
|-----|
|-----请给出你的选择-----|
|-----|

|=====Main Menu=====|
|-----|
|          1. SAT求解          |
|          2. 百分号数独游戏    |
|          0.  EXIT            |
|=====|

请选择 [1/2/0]
1
已经存在 cnf 文件,是否覆盖?[1/0]
```

图4.2.2-5 重复 cnf 文件询问是否覆盖

4.2.3 数独交互界面及功能模块测试

```

*****Menu for Percent_Sudoku*****
-----
1. 生成数独
2. 查看答案
3. 游玩数独
0. EXIT
*****

请选择[1/2/3/0]
1
你想要多少个提示数[18~81]
20
读取成功!
生成中
生成成功!
    
```

图4.2.4-1 生成一个有 22 个提示数的数独

x	x	x		x	6	8		x	x	x
x	x	x		x	1	x		x	x	x
x	x	8		x	x	5		x	x	x
-----			+	-----			+	-----		
x	x	x		x	8	x		9	x	4
4	x	x		6	9	x		x	x	x
9	x	x		1	x	4		x	x	x
-----			+	-----			+	-----		
x	x	4		x	x	x		x	x	1
x	x	9		x	7	x		x	x	x
x	5	x		x	x	9		x	x	x

请输入要插入的行与列,如'2 2'
 查看答案就输入'0 0'
 退出程序就输入'-1 -1'
 提交总答案就输入'-2 -2'

图4.2.4-2 数独游玩界面（x 表示空位）

```
1 1
请输入要插入的值
2
插入成功！
 2 x x | x 6 8 | x x x
x x x | x 1 x | x x x
x x 8 | x x 5 | x x x
-----+-----+-----
x x x | x 8 x | 9 x 4
4 x x | 6 9 x | x x x
9 x x | 1 x 4 | x x x
-----+-----+-----
x x 4 | x x x | x x 1
x x 9 | x 7 x | x x x
x 5 x | x x 9 | x x x
-----+-----+-----
 2 x x | x 6 8 | x x x
x x x | x 1 x | x x x
x x 8 | x x 5 | x x x
-----+-----+-----
x x x | x 8 x | 9 x 4
4 x x | 6 9 x | x x x
9 x x | 1 x 4 | x x x
-----+-----+-----
x x 4 | x x x | x x 1
x x 9 | x 7 x | x x x
x 5 x | x x 9 | x x x
```

图4.2.4-3 插入有效数字并打印新数独

```
 2 x x | x 6 8 | x x x
x x x | x 1 x | x x x
x x 8 | x x 5 | x x x
-----+-----+-----
x x x | x 8 x | 9 x 4
4 x x | 6 9 x | x x x
9 x x | 1 x 4 | x x x
-----+-----+-----
x x 4 | x x x | x x 1
x x 9 | x 7 x | x x x
x 5 x | x x 9 | x x x
请输入要插入的行与列,如 '2 2'
查看答案就输入 '0 0'
退出程序就输入 '-1 -1'
提交总答案就输入 '-2 -2'
1 2
请输入要插入的值
2
一行不能有相同数字
错误答案,请重新输入
```

图4.2.4-4 无效输入并给出详细原因

```

查看答案就输入 '0 0'
退出程序就输入 '-1 -1'
提交总答案就输入 '-2 -2'
0 0
初始数独是
  x  x  x | x  6  8 | x  x  x
  x  x  x | x  1  x | x  x  x
  x  x  8 | x  x  5 | x  x  x
-----+-----+-----
  x  x  x | x  8  x | 9  x  4
  4  x  x | 6  9  x | x  x  x
  9  x  x | 1  x  4 | x  x  x
-----+-----+-----
  x  x  4 | x  x  x | x  x  1
  x  x  9 | x  7  x | x  x  x
  x  5  x | x  x  9 | x  x  x
答案是：
  1  9  3 | 2  6  8 | 7  4  5
  5  4  2 | 7  1  3 | 6  8  9
  7  6  8 | 9  4  5 | 3  1  2
-----+-----+-----
  2  1  5 | 3  8  7 | 9  6  4
  4  3  7 | 6  9  2 | 1  5  8
  9  8  6 | 1  5  4 | 2  7  3
-----+-----+-----
  3  7  4 | 5  2  6 | 8  9  1
  8  2  9 | 4  7  1 | 5  3  6
  6  5  1 | 8  3  9 | 4  2  7

```

图4.2.4-5 查看参考答案

4.2.4 算例测试总结

文件	变量数 V	子句数 C	C/V	最优解
1.cnf	200	1200	6.00	直接拿第一个子句的 第一个单词
2.cnf	1075	3152	2.93	在最小子句找出现 次数最多的字
3.cnf	301	2780	9.24	直接拿第一个子句的 第一个单词
4 (unsatisfied) .cnf	512	9685	18.92	在最小子句找出现 次数最多的字
5.cnf	20	1532	76.60	找出现最多的字
6.cnf	265	5666	21.39	直接拿第一个子句的 第一个单词
7 (unsatisfied) .cnf	3000	8881	2.96	找出现最多的字
8 (unsatisfied) .cnf	238	634	2.66	直接拿第一个子句的 第一个单词
9 (unsatisfied) .cnf	99	264	2.67	找出现最多的字
10.cnf	3176	10297	3.24	找出现最多的字
11 (unsatisfied) .cnf	60	936	15.60	在最小子句找出现 次数最多的字
12.cnf	354	2596	7.34	超时

5 总结与展望

5.1 全文总结

在为期两周的程序设计中，我围绕 SAT 这一 NP 难题进行了深入探索，设计并实现了一个基于 SAT 求解的百分号数独程序。通过查阅并参考资料，我成功将数独问题转化为 SAT 问题，并运用求解器进行求解。

核心工作主要包括以下五个方面：

- (1) 学习并实现了基于 DPLL 算法的 SAT 求解器。
- (2) 对该求解器的变元选择策略进行了优化改进，提升了其性能。
- (3) 完成了百分号数独的初始化工作，包括生成 CNF 文件、利用挖洞法随机生成谜题、绘制数独盘面，并实现了基础交互功能。
- (4) 设计了测试方案，并通过多个算例成功验证了系统的有效性。
- (5) 将各个模块衔接整合，构建了一个完整的百分比数独求解简易系统。

本研究展示了 SAT 理论在实际问题中的应用，虽然目前的求解器仍是初级版本，但为未来的深入研究积累了宝贵的实践经验。

5.2 工作展望

在今后的研究中，我将围绕以下六个方面继续开展工作：

- (1) 我将进一步优化 SAT 求解器的性能。尽管当前的求解器已在变元选择策略上做了初步改进，未来我还会引入更先进的启发式算法，如 VSIDS (Variable State Independent Decaying Sum) 或 LRB (Learning Rate Branching) 等现代策略，通过融合多种方法提升求解效率，并在不同规模与复杂度的算例中验证其性能。
- (2) 我计划实现更丰富的数独生成算法。当前系统采用挖洞法生成题目，后续我将引入基于复杂度的可调节生成机制，以产生更具挑战性与多样性的数独谜题。我还打算集成难度分析模型，根据用户解题历史动态调整题目难度，增强游戏的趣味性和互动性。

- (3) 我将致力于增强用户交互与游戏体验。目前的数独游戏已具备基础交互功能，未来我会优化用户界面设计，增加提示系统、解题回放与错误反馈等功能，并引入计时器、排行榜等游戏化元素，提升用户的沉浸感和参与度。
- (4) 我希望能拓展 SAT 求解器的应用场景。SAT 技术不仅适用于数独求解，还可广泛应用于硬件验证、逻辑电路设计、AI 规划与其他约束满足问题。未来我将探索其在这些领域的具体应用，并针对不同场景进行专项优化。
- (5) 我打算构建更健壮的测试与验证体系。当前已完成多算例测试，后续我将建立大规模自动化测试框架，覆盖不同类型、规模和复杂度的 SAT 问题，确保系统在多种条件下的稳定性与效率，并对性能瓶颈和异常进行深入分析。
- (6) 我希望推动开源与社区合作。我计划将 SAT 求解器及数独生成工具在 GitHub 等平台开源，吸引全球开发者共同参与，通过社区反馈和协作进一步优化项目，推动其持续发展。

6 体会

在本次项目中，我对算法设计与优化有了更加深刻的理解。DPLL 算法的核心思想虽然清晰，但在实际实现过程中，如何设计高效的搜索策略并进行针对性优化，对最终性能的影响极为显著。值得一提的是，在老师和助教的耐心指导下，我少走了许多弯路，这使我不仅完成了算法的基础实现，更开始深入思考其运行机制与性能瓶颈所在。

具体而言，在实现 DPLL 算法时，选择合适的文字和处理单子句成为关键。最初我采用了一种较为简单的变元选择策略，但随着测试用例复杂度的增加，发现其处理大规模问题时效率有限。通过反复实验和对比不同策略，并在与助教的讨论中获得了宝贵建议，我认识到优化不仅仅是追求“更快”，更需要在性能与实现复杂度之间找到平衡。例如，某些复杂策略虽然能在特定场景提升效率，但在简单算例中反而可能降低整体速度。

此外，DPLL 算法本质上是基于递归的求解过程。在递归调用中，我遇到了明显的性能瓶颈。由于未对递归深度进行有效控制，算法在复杂情况下容易陷入过深的递归栈中。在此特别感谢助教提供的参考资料，让我对递归优化有了新的认识。在后续改进中，我计划引入递归深度限制，并设计更合理的剪枝策略，以减少无效搜索空间。毫无疑问，这一过程加深了我对递归算法设计及其优化方法的理解。

与此同时，调试和测试在本项目中展现出极高的重要性。面对复杂的递归调用和多变的逻辑分支，一些细微的错误往往隐藏极深，导致程序出现难以预料的行为。通过本次开发，我深刻体会到系统化测试对保障代码质量的关键作用，这也得益于老师在课堂上强调的测试方法论。

实际上，在开发初期，我就意识到 SAT 求解器的各个模块虽然相互依赖，但逻辑相对独立。因此，我为每个核心模块编写了单元测试。这样一来，这种方法不

仅帮助我快速定位和修复各个模块中的问题，也避免了在复杂系统中常见的困境，让我切身感受到分而治之思想在大型项目中的价值，这也正是老师一直强调的模块化设计理念。

不仅这样，在设计测试用例时，我注重从简单到复杂的渐进过程，特别是将百分比数独问题作为复杂的测试案例，这不仅验证了程序的正确性，也成为评估算法性能的重要手段。在此过程中，助教提供的测试案例库给了我很大启发。

另一方面，在调试过程中，我意识到除了修复功能错误，代码结构的合理性和执行效率同样关键。于是，通过持续的代码审查，我发现并消除了不必要的重复计算和内存资源浪费，这些优化不仅提升了程序运行速度，也增强了代码的可读性和可维护性。最终，这一过程带给我极大的成就感，也要感谢老师在代码评审中给予的宝贵建议，使我的代码质量得到了明显提升。

作为一名计算机专业的大二学生，我此前的编程经验多局限于基本编程模型和技术的验证性训练，而这次课程设计无疑是一次全面的挑战。第一次实验课时，当我第一次看到 SAT 问题和抽象语法树这两个课题选项时，感到十分迷茫。但在老师的启发式教学和学长的建议后，我最终选择了 SAT 问题。然而，第一节课我花了整整三个小时才基本理解 SAT 问题与 DPLL 算法的核心概念，那时项目的基本架构还没开始搭建。

之后，通过持续学习相关技术博客和论文，并在助教的定期指导下，我在第六次实验课时终于完成了项目并通过检查。然而，在撰写课程报告的过程中，我仍然发现了数个隐藏的 bug，这反映出之前对程序的测试覆盖仍有不足。总体而言，本次课程设计让我遇到了许多以前验证性实验中从未遇到过的问题，同时也积累了宝贵的程序设计和代码组织经验。更重要的是，我认识到，一个真正有价值的程序不仅要正确解决问题，还应具备良好的用户体验、可维护性和可验证性，而这些方面都是老师在教学过程中不断强调的工程化思维，让我受益匪浅。

参考文献

- [1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社, 2000
- [2] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University, Canada, 2009
- [3] 陈稳. 基于 DPLL 的 SAT 算法的研究与应用. 硕士学位论文, 电子科技大学, 2011
- [4] Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243
- [5] 360 百科: 数独游戏 <https://baike.so.com/doc/3390505-3569059.html>
Twodoku: <https://en.grandgames.net/multisudoku/twodoku>
- [6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.
- [7] Ins Lynce and Jol Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale. Springer, 2006.
- [8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57
- [9] Sudoku Puzzles Generating: from Easy to Evil.
http://zhangroup.aporc.org/images/files/Paper_3485.pdf
- [10] 薛源海, 蒋彪彬, 李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识, 2009, 39(21): 1-7
- [11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015, 32(2): 187-191

附录

SAT. hpp

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#include <windows.h>
#include <winnt.h>
#include <chrono>
typedef struct literal_node // 子句中的文字的链表
{
    int literal;
    struct literal_node *next;
} literal_node, *literal_list;
typedef struct clause_node // 每个子句的链表
{
    literal_list head;
    struct clause_node *next;
} clause_node, *clause_list;
typedef struct cnfNode // 总的cnf 文件链表
{
    clause_list root;
    int bool_count;
    int clause_count;
} cnf_node, *CNF;
#define SIZE 9
#define MIN(x, y) ((x) < (y) ? (x) : (y))
#define MAX(x, y) ((x) > (y) ? (x) : (y))
// 显示主界面
void main_display();
// 显示菜单
void display_menu();
// 数独主流程
void main_sudoku();
```

```
// 显示数独界面
void display_sudoku();
// 生成数独题目
void generate_sudoku(int (&fixed_board)[SIZE + 1][SIZE + 1], int (&answer_board)[SIZE + 1][SIZE + 1], int (&play_board)[SIZE + 1][SIZE + 1], int hint, bool value[SIZE * SIZE * SIZE + 1], bool (&is_num)[SIZE + 1][SIZE + 1]);
// 打印棋盘
void print_board(int board[SIZE + 1][SIZE + 1]);
// 数独游戏交互
void play_sudoku(int answer_board[SIZE + 1][SIZE + 1], int (&play_board)[SIZE + 1][SIZE + 1], bool is_num[SIZE + 1][SIZE + 1], int fixed_board[SIZE + 1][SIZE + 1]);
// 检查数字是否合法
bool is_valid(int hang, int lie, int value, int play_board[SIZE + 1][SIZE + 1]);
// 写CNF文件
bool write_file(int (&board)[SIZE + 1][SIZE + 1], int num, char name[])
;
// 打乱数组
void shuffle(int (&arr)[], int size);
// 用SAT解数独
bool solve_sudoku(int (&board)[SIZE + 1][SIZE + 1], bool (&value)[SIZE * SIZE * SIZE + 1]);
// 读取CNF文件
bool read_file(char file_name[], CNF &cnf);
// 释放CNF链表
void destroy_cnf(CNF &cnf);
// 打印CNF链表
void print_cnf(CNF cnf);
// 查找单子句
int find_unit_clause(clause_list cl);
// 移除子句
void remove_clause(clause_list &cl);
// 化简子句
void simplify(clause_list &cl, int lit);
// 判断是否满足
bool is_satisfy(clause_list cl);
// 判断是否有空子句
bool is_empty_clause(clause_list cl);
// 选择分支变量1
```

```

int choose_literal_1(CNF cnf);
// 选择分支变量2
int choose_literal_2(CNF cnf);
// 选择分支变量3
int choose_literal_3(CNF cnf);
// 拷贝子句链表
clause_list copy_cnf(clause_list cl);
// 保存SAT解
bool save_file(int result, char file_name[], double time, bool value[],
    int bool_count, double time_);
// 基本DPLL算法
bool DPLL(CNF cnf, bool value[], int flag);
// 支持超时的DPLL
int DPLL_2(CNF cnf, bool value[], int flag, const std::chrono::steady_c
    lock::time_point &start, double timeout_seconds);
// 按flag和超时求解, 返回耗时
int solve_with_timeout_flag(CNF cnf, bool value[], int flag, double tim
    eout_seconds, double &elapsed_seconds);

```

main.cpp

```

#include "SAT.hpp"
#include "%_sudoku.hpp"
#include "cnfparser.hpp"
#include "display.hpp"
#include "solver.hpp"
int main()
{
    SetConsoleOutputCP(CP_UTF8);
    main_display();
    return 0;
}

```

display.hpp

```

#include "SAT.hpp"
void main_display()
{
    CNF cnf;
    bool file_exist = 0;
    int choice = 1;
}

```



```

char file_name[100];
while (choice)
{
    display_menu();
    scanf("%d", &choice);
    switch (choice)
    {
        case 1: // SAT 求解
        {
            int ch = 0;
            if (!file_exist)
            {
                do
                {
                    printf("请输入文件地址\n");
                    scanf("%s", file_name);
                    file_exist = 1;
                } while (!read_file(file_name, cnf)); // 检查cnf 文件是否存在，顺便初始化cnf 链表
            }
            else
            {
                printf("已经存在 cnf 文件,是否覆盖?[1/0]\n");
                scanf("%d", &ch);
                if (ch == 1)
                {
                    destroy_cnf(cnf);
                    do
                    {
                        printf("请输入文件地址\n");
                        scanf("%s", file_name);
                    } while (!read_file(file_name, cnf));
                }
            }
            printf("需要我遍历一遍 cnf 文件吗[1/0]\n");
            scanf("%d", &ch);
            if (ch)
            {
                print_cnf(cnf);
                getchar();
                getchar();
            }
        }
    }
}

```

```

    }
    // printf("求解中...\n");
    // CNF cnf_new = new cnf_node;
    // cnf_new->root = copy_cnf(cnf->root);
    // cnf_new->bool_count = cnf->bool_count;
    // cnf_new->clause_count = cnf->clause_count;
    // bool *value = new bool[cnf->bool_count + 1];
    // for (int i = 1; i <= cnf->bool_count; i++)
    //     value[i] = 1;
    printf("尝试在最小子句找出现次数最多的字并求解...\n");
    bool *value = new bool[cnf->bool_count + 1];
    for (int i = 1; i <= cnf->bool_count; ++i)
        value[i] = 1;
    double t1 = 0.0, t2 = 0.0, t3 = 0.0;
    int res1 = solve_with_timeout_flag(cnf, value, 1, 30.0, t1)
;
    printf("尝试使用 MOMS 算法并求解...\n");
    int res2 = solve_with_timeout_flag(cnf, value, 2, 30.0, t2)
;
    printf("尝试取第一个单词并求解...\n");
    int res3 = solve_with_timeout_flag(cnf, value, 2, 30.0, t3)
;
    printf("flag1 (method1) time: %.6f s, result=%d\n", t1, res
1);
    printf("flag2 (method2) time: %.6f s, result=%d\n", t2, res
2);
    printf("flag2 (method2) time: %.6f s, result=%d\n", t3, res
3);

    double optimization_rate = 0.0;
    int faster_flag = 0;
    double faster_time = 0.0, slower_time = 0.0;
    // 判断哪个方法有有效结果 (res==1 或 res==0 表示有结论)
    if (!(res1 == res2 && res1 == -1 && res3 != -1)) // 只要有算
出结果
    {
        faster_time = MIN(MIN(t1, t2), t3);
        if (faster_time == t1)
            faster_flag = 1;
        if (faster_time == t2)
            faster_flag = 2;
        if (faster_time == t3)

```

```

        faster_flag = 3;
        slower_time = MAX(MAX(t1, t2), t3);
        optimization_rate = ((slower_time - faster_time) / slower_time) * 100;
        if (faster_flag == 1)
            printf("更快的方法: 在最小子句找出现次数最多的字, time=%.6f s\n", faster_time);
        else if (faster_flag == 2)
            printf("更快的方法: MOMS 算法, time=%.6f s\n", faster_time);
        else if (faster_flag == 3)
            printf("更快的方法: 取第一个单词, time=%.6f s\n", faster_time);
        if (res1 != -1 && res2 != -1 && res3 != -1)
            printf("优化率: %.2f%%\n", optimization_rate);
        else
            printf("优化率:100%%\n");
        printf("是否要保存到文件中?[1/0]\n");
        scanf("%d", &ch);
        if (ch)
        {
            int result = t1 < t2 ? res1 : res2;
            if (save_file(result, file_name, slower_time, value, cnf->bool_count, faster_time))
                printf("保存成功!\n");
            else
                printf("保存失败\n");
        }
    }
    else
    {
        printf("两种方法均无有效结论。 \n");
    }
    printf("\n");
    delete[] value;
    break;
}
case 2: // 百分号数独游戏
    main_sudoku();
    break;
case 0: // Exit

```

```

        return;
    default:
        printf("指令无效\n");
        break;
    }
}
return;
}

void display_menu()
{
    printf("\n|-----|\n");
    printf("|-----请给出你的选择-----|\n");
    printf("|-----|\n\n");
    printf("|=====Main Menu=====|\n");
    printf("|-----|\n");
    printf("|          1. SAT 求解          |\n");
    printf("|          2. 百分号数独游戏      |\n");
    printf("|          0.  EXIT              |\n");
    printf("|=====|\n\n");
    printf("请选择[1/2/0]\n");
    return;
}

void display_sudoku()
{
    printf("|*****Menu for Percent_Sudoku*****|\n");
    printf("|-----|\n");
    printf("|          1. 生成数独          |\n");
    printf("|          2. 查看答案          |\n");
    printf("|          3. 游玩数独          |\n");
    printf("|          0. EXIT              |\n");
    printf("|*****|\n\n");
    printf("请选择[1/2/3/0]\n");
    return;
}

```

cnfparser. hpp

```

#include "SAT.hpp"
bool read_file(char file_name[], CNF &cnf)
{
    FILE *fp = fopen(file_name, "r");

```

```

if (!fp)
{
    printf("文件打开失败: %s\n", file_name);
    return 0;
}
cnf = new cnf_node;
char ch;
while ((ch = getc(fp)) == 'c')
{
    while ((ch = getc(fp)) != '\n')
        continue;
}
getc(fp);
getc(fp);
getc(fp);
getc(fp);
fscanf(fp, "%d%d", &cnf->bool_count, &cnf->clause_count);
if (cnf->bool_count <= 0 || cnf->clause_count <= 0)
{
    printf("CNF header 未找到或无效\n");
    fclose(fp);
    free(cnf);
    return 0;
}
cnf->root = NULL;
clause_list last_clause = NULL;
for (int ci = 0; ci < cnf->clause_count; ++ci)
{
    clause_list cl = new clause_node;
    cl->head = NULL;
    cl->next = NULL;
    literal_list last_lit = NULL;
    int lit;
    while (fscanf(fp, "%d", &lit) == 1)
    {
        if (lit == 0)
            break;
        literal_list ln = new literal_node;
        ln->literal = lit;
        ln->next = NULL;
        if (last_lit == NULL)

```

```

        cl->head = ln;
    else
        last_lit->next = ln;
        last_lit = ln;
    }
    if (cnf->root == NULL)
    {
        cnf->root = cl;
        last_clause = cl;
    }
    else
    {
        last_clause->next = cl;
        last_clause = cl;
    }
}
fclose(fp);
printf("读取成功! \n");
return 1;
}
void destroy_cnf(CNF &cnf)
{
    while (cnf->root)
    {
        clause_list temp = cnf->root;
        literal_list delete_node = temp->head;
        while (delete_node)
        {
            literal_list p = delete_node;
            delete_node = delete_node->next;
            delete p;
        }
        cnf->root = cnf->root->next;
        delete temp;
    }
    cnf->root = NULL;
    return;
}
void print_cnf(CNF cnf)
{
    clause_list p = cnf->root;

```

```

if (p == NULL) // 如果没有子句
{
    printf(" 没有句子\n");
    return;
}
printf("  The CNF is:\n");
printf("  boolCount:%d\n", cnf->bool_count);
printf("  clauseCount:%d\n", cnf->clause_count);
while (p)
{
    literal_list q = p->head;
    printf("  ");
    while (q)
    {
        printf("%-5d", q->literal);
        q = q->next;
    }
    printf("\n");
    p = p->next;
}
return;
}

```

solver.hpp

```

#include "SAT.hpp"
int find_unit_clause(clause_list cl) // 找到单子句
{
    while (cl)
    {
        if (cl->head != NULL && cl->head->next == NULL)
            return cl->head->literal;
        cl = cl->next;
    }
    return 0;
}
void remove_clause(clause_list &cl) // 删除子句
{
    literal_list temp = cl->head;
    while (temp)
    {

```

```

        literal_list p = temp;
        temp = temp->next;
        delete (p);
    }
    delete (cl);
    cl = NULL;
    return;
}
void simplify(clause_list &cl, int lit) // 化简子句
{
    clause_list pre = NULL, p = cl;
    while (p)
    {
        bool is_deleted = 0;
        literal_list lpre = NULL, q = p->head;
        while (q)
        {
            if (q->literal == lit) // 文字等于 literal
            {
                if (!pre)
                {
                    cl = cl->next;
                    remove_clause(p);
                    p = cl;
                }
                else
                {
                    pre->next = p->next;
                    remove_clause(p);
                    p = pre->next;
                }
                is_deleted = 1;
                break;
            }
            else if (q->literal == -lit) // 文字等于 -literal
            {
                if (lpre)
                {
                    lpre->next = q->next;
                    delete (q);
                    q = lpre->next;
                }
            }
        }
    }
}

```



```

        }
        else
        {
            p->head = q->next;
            delete (q);
            q = p->head;
        }
    }
    else // 啥都没有
    {
        lpre = q;
        q = q->next;
    }
}
if (!is_deleted)
{
    pre = p;
    p = p->next;
}
}
}
clause_list copy_cnf(clause_list cl)
{
    clause_list new_cnf = new clause_node;
    clause_list cl_new = new_cnf, cl_old = cl;
    while (cl_old)
    {
        literal_list lit_new;
        literal_list lit_old = cl_old->head;
        if (lit_old)
        {
            lit_new = new literal_node;
            lit_new->literal = lit_old->literal;
            lit_new->next = NULL;
            cl_new->head = lit_new;
            lit_old = lit_old->next;
        }
        while (lit_old)
        {
            lit_new->next = new literal_node;
            lit_new = lit_new->next;
        }
    }
}

```

```

        lit_new->literal = lit_old->literal;
        lit_new->next = NULL;
        lit_old = lit_old->next;
    }
    if (cl_old->next)
    {
        cl_new->next = new clause_node;
        cl_new = cl_new->next;
        cl_new->next = NULL;
        cl_new->head = NULL;
    }
    cl_old = cl_old->next;
}
return new_cnf;
}
bool is_satisfy(clause_list cl) // 判断cnf是否可以满足
{
    if (!cl)
        return 1;
    return 0;
}
bool is_empty_clause(clause_list cl) // 判断有没有空子句
{
    while (cl)
    {
        if (!cl->head)
            return 1;
        cl = cl->next;
    }
    return 0;
}
int choose_literal_1(CNF cnf) // 没有优化的选择
{
    return cnf->root->head->literal;
}
int choose_literal_2(CNF cnf) // 找出现最多的字
{
    int *value = new int[cnf->bool_count * 2 + 1];
    for (int i = 0; i < cnf->bool_count * 2 + 1; i++)
        value[i] = 0;
    for (clause_list cl = cnf->root; cl != NULL; cl = cl->next)

```

```

{
    for (literal_list lit = cl->head; lit != NULL; lit = lit->next)
    {
        if (lit->literal > 0)
            value[lit->literal]++;
        else
            value[cnf->bool_count - lit->literal]++;
    }
}
int max = 0, max_lit;
for (int i = 0; i < cnf->bool_count; i++)
{
    if (value[i] > max)
    {
        max = value[i];
        max_lit = i;
    }
}
if (max == 0)
{
    for (int i = cnf->bool_count; i < cnf->bool_count * 2; i++)
    {
        if (value[i] > max)
        {
            max = value[i];
            max_lit = cnf->bool_count - i;
        }
    }
}
delete[] value;
return max_lit;
}
int choose_literal_3(CNF cnf) // 在最小子句找出现次数最多的字
{
    clause_list max_cl;
    int *value = new int[cnf->bool_count * 2 + 1];
    for (int i = 0; i <= cnf->bool_count * 2; i++)
        value[i] = 0;
    int max = INT_MAX;
    for (clause_list cl = cnf->root; cl; cl = cl->next)
    {

```

```

        int count = 0;
        literal_list lit = cl->head;
        while (lit)
        {
            count++;
            lit = lit->next;
        }
        if (count < max)
        {
            max = count;
            max_cl = cl;
        }
    }
    for (literal_list lit = max_cl->head; lit != NULL; lit = lit->next)
    {
        value[lit->literal + cnf->bool_count]++;
    }
    max = 0;
    int max_lit;
    for (int i = 0; i < cnf->bool_count * 2 + 1; i++)
    {
        if (value[i] > max)
        {
            max = value[i];
            max_lit = i - cnf->bool_count;
        }
    }
    delete[] value;
    return max_lit;
}

bool save_file(int result, char file_name[], double time, bool value[],
    int bool_count, double time_)
{
    FILE *fp;
    char name[100];
    for (int i = 0; file_name[i] != '\0'; i++)
    {
        if (file_name[i] == '.' && file_name[i + 4] == '\0')
        {
            name[i] = '.';
            name[i + 1] = 'r';

```

```

        name[i + 2] = 'e';
        name[i + 3] = 's';
        name[i + 4] = '\\0';
        break;
    }
    name[i] = file_name[i];
}
if (fopen_s(&fp, name, "w"))
{
    printf("保存失败\\n");
    return 0;
}
fprintf(fp, "s %d", result);
if (result == 1)
{
    fprintf(fp, "\\nv\\n");
    for (int i = 1; i <= bool_count; i++)
    {
        if (value[i] == 1)
            fprintf(fp, "%d ", i);
        else
            fprintf(fp, "%d ", -i);
    }
}
fprintf(fp, "\\nt %lfms", time * 1000);
if (time_ != 0)
{
    fprintf(fp, "\\nt %lfms(optimized)", time_ * 1000);          // 运行时间/毫秒
    double optimization_rate = ((time - time_) / time) * 100;    // 优化率
    fprintf(fp, "\\nOptimization Rate: %.2lf%%", optimization_rate);
}
fclose(fp);
return 1;
}
int choose_literal_4(CNF cnf) // MOMS
{
    if (!cnf || !cnf->root)
        return 0;
    int V = cnf->bool_count;

```

```

int m = INT_MAX;
// 找最短子句长度
for (clause_list c = cnf->root; c != NULL; c = c->next)
{
    int len = 0;
    for (literal_list p = c->head; p != NULL; p = p->next)
        ++len;
    if (len > 0 && len < m)
        m = len;
}
if (m == INT_MAX)
    return 0;
int *cnt_pos = new int[V + 1]();
int *cnt_neg = new int[V + 1]();
for (clause_list c = cnf->root; c != NULL; c = c->next)
{
    int len = 0;
    for (literal_list p = c->head; p != NULL; p = p->next)
        ++len;
    if (len != m)
        continue;
    for (literal_list p = c->head; p != NULL; p = p->next)
    {
        int L = p->literal;
        if (L > 0)
            cnt_pos[L]++;
        else
            cnt_neg[-L]++;
    }
}
int best_v = 0, best_total = -1;
for (int v = 1; v <= V; ++v)
{
    int tot = cnt_pos[v] + cnt_neg[v];
    if (tot > best_total)
    {
        best_total = tot;
        best_v = v;
    }
}
int res = 0;

```

```

    if (best_total > 0)
        res = (cnt_pos[best_v] >= cnt_neg[best_v]) ? best_v : -best_v;
    delete[] cnt_pos;
    delete[] cnt_neg;
    return res;
}
// ==== 全局活动度数组 ====
double *g_activity = NULL;
int g_activity_size = 0;
// 初始化 VSIDS 分数: 统计出现次数
void vsids_init_from_cnf(CNF cnf)
{
    int V = cnf->bool_count;
    if (g_activity)
        delete[] g_activity;
    g_activity = new double[V + 1];
    g_activity_size = V;
    for (int i = 0; i <= V; i++)
        g_activity[i] = 0.0;
    for (clause_list c = cnf->root; c != NULL; c = c->next)
    {
        for (literal_list p = c->head; p != NULL; p = p->next)
        {
            g_activity[abs(p->literal)] += 1.0;
        }
    }
}
// 用静态 VSIDS 分数选择 literal
int choose_literal_vsids_static(CNF cnf) // https://www.researchgate.net/publication/279633448\_Understanding\_VSIDS\_Branching\_Heuristics\_in\_Conflict-Driven-Clause-Learning\_SAT\_Solvers
{
    if (!cnf || !cnf->root)
        return 0;
    if (!g_activity)
        return 0;
    int best_v = 0;
    double best = -1.0;
    int V = cnf->bool_count;
    // 1. 找出出现频率最高且仍在 CNF 中的变量
    for (int v = 1; v <= V; v++)

```

```

    {
        if (g_activity[v] <= 0.0)
            continue;
        bool appears = false;
        for (clause_list c = cnf->root; c != NULL && !appears; c = c->n
ext)
        {
            for (literal_list p = c->head; p != NULL; p = p->next)
            {
                if (abs(p->literal) == v)
                {
                    appears = true;
                    break;
                }
            }
            if (!appears)
                continue;
            if (g_activity[v] > best)
            {
                best = g_activity[v];
                best_v = v;
            }
        }
        if (best_v == 0)
            return 0; // 没有找到
        // 2. polarity 选择: 统计正负出现次数
        int pos = 0, neg = 0;
        for (clause_list c = cnf->root; c != NULL; c = c->next)
        {
            for (literal_list p = c->head; p != NULL; p = p->next)
            {
                if (abs(p->literal) == best_v)
                {
                    if (p->literal > 0)
                        pos++;
                    else
                        neg++;
                }
            }
        }
    }
}

```



```

    return (pos >= neg) ? best_v : -best_v;
}
bool DPLL(CNF cnf, bool value[], int flag) // 主要的东西
{
    int unit_lit = find_unit_clause(cnf->root);
    while (unit_lit != 0)
    {
        value[abs(unit_lit)] = (unit_lit > 0) ? 1 : 0;
        simplify(cnf->root, unit_lit);
        if (is_satisfy(cnf->root))
            return 1;
        if (is_empty_clause(cnf->root))
            return 0;
        unit_lit = find_unit_clause(cnf->root);
    }
    if (is_empty_clause(cnf->root))
        return 0;
    int lit = choose_literal_3(cnf);
    // if (flag == 1)
    //     lit = choose_literal_1(cnf); // 直接拿第一个子句的第一个单词
    // else if (flag == 2)
    //     lit = choose_literal_2(cnf); // 找出现最多的字
    // else if (flag == 3)
    //     lit = choose_literal_3(cnf); // 在最小子句找出现次数最多的字
    // else if (flag == 4)
    //     lit = choose_literal_moms(cnf);
    // 假设 TRUE
    CNF cnf_new = new cnf_node;
    cnf_new->root = copy_cnf(cnf->root);
    cnf_new->bool_count = cnf->bool_count;
    cnf_new->clause_count = cnf->clause_count;
    clause_list cl_new = new clause_node;
    cl_new->head = new literal_node;
    cl_new->head->next = NULL;
    cl_new->head->literal = lit;
    cl_new->next = cnf_new->root;
    cnf_new->root = cl_new;
    if (DPLL(cnf_new, value, flag) == 1)
        return 1;
    destroy_cnf(cnf_new);
    // 假设 FALSE

```

```

    clause_list cl_new2 = new clause_node;
    cl_new2->head = new literal_node;
    cl_new2->head->next = NULL;
    cl_new2->head->literal = -lit;
    cl_new2->next = cnf->root;
    cnf->root = cl_new2;
    return DPLL(cnf, value, flag);
}

int DPLL_2(CNF cnf, bool value[], int flag, const std::chrono::steady_clock::time_point &start, double timeout_seconds)
{
    auto timed_out = [&](void) -> bool
    {
        auto now = std::chrono::steady_clock::now();
        double elapsed = std::chrono::duration<double>(now - start).count();
        return elapsed >= timeout_seconds;
    };
    if (timed_out())
        return -1;
    int unit_lit = find_unit_clause(cnf->root);
    while (unit_lit != 0)
    {
        if (timed_out())
            return -1;
        value[abs(unit_lit)] = (unit_lit > 0) ? 1 : 0;
        simplify(cnf->root, unit_lit);
        if (is_satisfy(cnf->root))
            return 1;
        if (is_empty_clause(cnf->root))
            return 0;
        unit_lit = find_unit_clause(cnf->root);
    }
    if (is_empty_clause(cnf->root))
        return 0;
    int lit;
    // if (flag == 1)
    //     lit = choose_literal_1(cnf);
    // else if (flag == 2)
    //     lit = choose_literal_2(cnf);
    if (flag == 1)

```

```

        lit = choose_literal_3(cnf);
    else if (flag == 2)
        lit = choose_literal_4(cnf);
    else if (flag == 3)
        lit = choose_literal_1(cnf);
    if (timed_out())
        return -1;
    CNF cnf_new = new cnf_node;
    cnf_new->root = copy_cnf(cnf->root);
    cnf_new->bool_count = cnf->bool_count;
    cnf_new->clause_count = cnf->clause_count;
    clause_list cl_new = new clause_node;
    cl_new->head = new literal_node;
    cl_new->head->next = NULL;
    cl_new->head->literal = lit;
    cl_new->next = cnf_new->root;
    cnf_new->root = cl_new;
    int res = DPLL_2(cnf_new, value, flag, start, timeout_seconds);
    if (res == 1)
    {
        destroy_cnf(cnf_new);
        return 1;
    }
    if (res == -1)
    {
        destroy_cnf(cnf_new);
        return -1;
    }
    destroy_cnf(cnf_new);
    // 假设 FALSE
    clause_list cl_new2 = new clause_node;
    cl_new2->head = new literal_node;
    cl_new2->head->next = NULL;
    cl_new2->head->literal = -lit;
    cl_new2->next = cnf->root;
    cnf->root = cl_new2;
    return DPLL_2(cnf, value, flag, start, timeout_seconds);
}

int solve_with_timeout_flag(CNF cnf, bool value[], int flag, double timeout_seconds, double &elapsed_seconds)
{

```

```

    auto _start = std::chrono::steady_clock::now();
    // 复制 CNF 给 DPLL 使用
    CNF cnf_copy = new cnf_node;
    cnf_copy->root = copy_cnf(cnf->root);
    cnf_copy->bool_count = cnf->bool_count;
    cnf_copy->clause_count = cnf->clause_count;
    // 重置 assignment 为未赋值 (0)
    for (int i = 1; i <= cnf->bool_count; ++i)
        value[i] = 1;
    int res = DPLL_2(cnf_copy, value, flag, _start, timeout_seconds);
    auto _end = std::chrono::steady_clock::now();
    elapsed_seconds = std::chrono::duration<double>(_end - _start).count();
    t();
    destroy_cnf(cnf_copy);
    return res; // 1=SAT, 0=UNSAT, -1=TIMEOUT
}

```

%_sudoku. hpp

```

#include "SAT.hpp"
// 两个窗口
static const int percentA[9][2] = {
    {2, 2}, {2, 3}, {2, 4}, {3, 2}, {3, 3}, {3, 4}, {4, 2}, {4, 3}, {4, 4}};
static const int percentB[9][2] = {
    {6, 6}, {6, 7}, {6, 8}, {7, 6}, {7, 7}, {7, 8}, {8, 6}, {8, 7}, {8, 8}};
void main_sudoku() // 数独主函数
{
    int choice = 1;
    bool flag = 0;
    int fixed_board[SIZE + 1][SIZE + 1] = {0}; // 生成的数独
    int answer_board[SIZE + 1][SIZE + 1] = {0}; // 答案数独
    int play_board[SIZE + 1][SIZE + 1] = {0}; // 用来玩的数独
    bool is_num[SIZE + 1][SIZE + 1]; // 判断数独某个位置存不存在提示数字
    bool value[SIZE * SIZE * SIZE + 1] = {0};
    while (choice)
    {
        display_sudoku();
        scanf("%d", &choice);
    }
}

```

```

switch (choice)
{
case 1: // 生成数独
    printf("你想要多少个提示数[18~81]\n");
    int hint;
    do
    {
        scanf("%d", &hint);
        if (hint < 18 || hint > 81)
            printf("数字无效, 请重新输入");
    } while (hint < 18 || hint > 81);
    generate_sudoku(fixed_board, answer_board, play_board, hint
, value, is_num);
    flag = 1;
    printf("生成成功!\n");
    break;
case 2: // 查看答案
    if (!flag)
    {
        printf("请先生成数独\n");
        continue;
    }
    flag = 0;
    printf("          初始数独\n");
    print_board(fixed_board);
    printf("          答案\n");
    solve_sudoku(answer_board, value);
    print_board(answer_board);
    break;
case 3: // 游玩数独
    if (!flag)
    {
        printf("请先生成数独\n");
        continue;
    }
    solve_sudoku(answer_board, value);
    play_sudoku(answer_board, play_board, is_num, fixed_board);
    flag = 0;
    //
    break;
case 0: // Exit

```

```

        return;
    default:
        printf("指令无效\n");
        break;
    }
}
}
}
void generate_sudoku(int (&fixed_board)[SIZE + 1][SIZE + 1], int (&answer_board)[SIZE + 1][SIZE + 1], int (&play_board)[SIZE + 1][SIZE + 1], int hint, bool value[SIZE * SIZE * SIZE + 1], bool (&is_num)[SIZE + 1][SIZE + 1])
{
    char file_name[100] = "percent_sudoku.cnf";
    srand(time(NULL));
    while (1)
    {
        // 清空棋盘与标记
        for (int i = 1; i <= SIZE; i++)
            for (int j = 1; j <= SIZE; j++)
            {
                fixed_board[i][j] = 0;
                play_board[i][j] = 0;
                answer_board[i][j] = 0;
                is_num[i][j] = 0;
            }
        int nA[9], nB[9];
        for (int i = 0; i < SIZE; i++)
        {
            nA[i] = i + 1;
            nB[i] = i + 1;
        }
        shuffle(nA, SIZE);
        shuffle(nB, SIZE);
        for (int i = 0; i < SIZE; i++)
        {
            int hang = percentA[i][0];
            int lie = percentA[i][1];
            play_board[hang][lie] = nA[i];
            fixed_board[hang][lie] = nA[i];
            answer_board[hang][lie] = nA[i];
            is_num[hang][lie] = 1;
        }
    }
}

```

```

    }
    for (int i = 0; i < SIZE; i++)
    {
        int hang = percentB[i][0];
        int lie = percentB[i][1];
        play_board[hang][lie] = nB[i];
        fixed_board[hang][lie] = nB[i];
        answer_board[hang][lie] = nB[i];
        is_num[hang][lie] = 1;
    }
    for (int i = 1; i <= SIZE * SIZE * SIZE; i++)
        value[i] = 0;
    write_file(fixed_board, 27, file_name);
    CNF cnf = new cnf_node;
    cnf->root = NULL;
    read_file(file_name, cnf);
    printf("生成中\n");
    if (DPLL(cnf, value, 3) == 0)
    {
        destroy_cnf(cnf);
        continue;
    }
    for (int i = 1; i <= SIZE * SIZE * SIZE; i++)
    {
        if (value[i])
        {
            int hang = (i - 1) / (SIZE * SIZE) + 1;
            int lie = ((i - 1) / SIZE) % SIZE + 1;
            int val = (i - 1) % SIZE + 1;
            fixed_board[hang][lie] = play_board[hang][lie] = answer
_board[hang][lie] = val;
            is_num[hang][lie] = 1;
        }
    }
    int remove = 81 - hint;
    int per_hang = remove / SIZE;
    int rem = remove % SIZE;
    for (int hang = 1; hang <= SIZE; hang++)
    {
        int to_remove = per_hang;
        while (to_remove > 0)

```

```

        {
            int lie = rand() % SIZE + 1;
            if (fixed_board[hang][lie] != 0)
            {
                fixed_board[hang][lie] = play_board[hang][lie] = answer_board[hang][lie] = 0;
                is_num[hang][lie] = 0;
                to_remove--;
            }
        }
    }
    while (rem > 0)
    {
        int r = rand() % SIZE + 1;
        int c = rand() % SIZE + 1;
        if (fixed_board[r][c] != 0)
        {
            fixed_board[r][c] = play_board[r][c] = answer_board[r][c] = 0;
            is_num[r][c] = 0;
            rem--;
        }
    }
    destroy_cnf(cnf);
    return;
}
}

void play_sudoku(int answer_board[SIZE + 1][SIZE + 1], int (&play_board)[SIZE + 1][SIZE + 1], bool is_num[SIZE + 1][SIZE + 1], int fixed_board[SIZE + 1][SIZE + 1])
{
    while (1)
    {
        print_board(play_board);
        int hang, lie;
        printf("请输入要插入的行与列,如'2 2'\n 查看答案就输入'0 0'\n 退出程序就输入'-1 -1'\n 提交总答案就输入'-2 -2'\n");
        scanf("%d %d", &hang, &lie);
        if (hang == 0 && lie == 0) // 查看答案
        {
            printf("初始数独是\n");
        }
    }
}

```



```

        print_board(fixed_board);
        printf("答案是: \n");
        print_board(answer_board);
        return;
    }
    else if (hang == -1 && lie == -1) // 退出程序
        return;
    else if (hang == -2 && lie == -2) // 提交答案
    {
        bool is_complete = 1;
        for (int i = 1; i < SIZE + 1; i++)
        {
            if (is_complete == 0)
                break;
            for (int j = 1; j < SIZE + 1; j++)
            {
                if (play_board[hang][lie] == 0)
                {
                    is_complete = 0;
                    break;
                }
            }
        }
        if (is_complete == 0)
        {
            printf("数独还没有完成\n");
            continue;
        }
        else if (is_complete == 1)
        {
            printf("恭喜挑战成功!\n");
            return;
        }
    }
    else if (hang > 9 || hang < 1 || lie > 9 || lie < 1)
    {
        printf("数值非法,请重新输入\n");
        continue;
    }
    else if (is_num[hang][lie])
    {

```

```

        printf("这个位置已经存在提示数值,请重新输入\n");
        continue;
    }
    printf("请输入要插入的值\n");
    int value;
    scanf("%d", &value);
    if (value < 1 || value > 9)
    {
        printf("数值非法,请重新输入\n");
        continue;
    }
    if (is_valid(hang, lie, value, play_board))
    {
        printf("插入成功!\n");
        play_board[hang][lie] = value;
        print_board(play_board);
        printf("\n");
    }
    else
        printf("错误答案,请重新输入\n");
}
}

void print_board(int board[SIZE + 1][SIZE + 1])
{
    for (int i = 1; i <= SIZE; i++)
    {
        for (int j = 1; j <= SIZE; j++)
        {
            if (board[i][j] == 0)
                printf(" x ");
            else
                printf("%2d ", board[i][j]);
            if (j % 3 == 0 && j != SIZE)
                printf("|");
        }
        printf("\n");
        if (i % 3 == 0 && i != SIZE)
        {
            for (int k = 1; k <= SIZE; k++)
            {
                printf("---");
            }
        }
    }
}

```

```

        if (k % 3 == 0 && k != SIZE)
            printf("+");
    }
    printf("\n");
}
}
}
bool is_valid(int hang, int lie, int value, int play_board[SIZE + 1][SIZE + 1]) // 判断插入的数字在那个位置是不是有效的
{
    for (int i = 1; i < SIZE + 1; i++) // 一行不能有相同数字
    {
        if (i != lie && value == play_board[hang][i])
        {
            printf("一行不能有相同数字\n");
            return 0;
        }
    }
    for (int i = 1; i < SIZE + 1; i++) // 一列不能有相同数字
    {
        if (i != hang && value == play_board[i][lie])
        {
            printf("一列不能有相同数字\n");
            return 0;
        }
    }
    // 每个 3x3 单元格里不能有相同数字
    int start_hang = (hang - 1) / 3 * 3 + 1;
    int start_lie = (lie - 1) / 3 * 3 + 1;
    for (int i = start_hang; i < start_hang + 3; i++)
    {
        for (int j = start_lie; j < start_lie + 3; j++)
        {
            if ((i != hang || j != lie) && value == play_board[i][j])
            {
                printf("每个 3x3 单元格里不能有相同数字\n");
                return 0;
            }
        }
    }
}
// 副对角线不能有相同数字

```

```

    if (hang + lie == 10)
    {
        for (int i = 1, j = SIZE; i < SIZE + 1, j >= 1; i++, j--)
        {
            if ((i != hang || j != lie) && value == play_board[i][j])
            {
                printf("副对角线不能有相同数字\n");
                return 0;
            }
        }
    }
    // 百分号窗口不能有相同数字
    bool in_a = 0, in_b = 0;
    if ((hang >= 2 && hang <= 4) && (lie >= 2 && lie <= 4))
        in_a = 1;
    else if ((hang >= 6 && hang <= 8) && (lie >= 6 && lie <= 8))
        in_b = 1;
    if (in_a) // 在左上的窗口
    {
        for (int i = 2; i <= 4; i++)
        {
            for (int j = 2; j <= 4; j++)
            {
                if ((i != hang || j != lie) && value == play_board[i][j
            ])
                {
                    printf("百分号窗口中不能有相同数字\n");
                    return 0;
                }
            }
        }
    }
    else if (in_b) // 在右下的窗口
    {
        for (int i = 6; i <= 8; i++)
        {
            for (int j = 6; j <= 8; j++)
            {
                if ((i != hang || j != lie) && value == play_board[i][j
            ])
                {

```

```

        printf("百分号窗口中不能有相同数字\n");
        return 0;
    }
}
}
return 1;
}
}
void shuffle(int (&arr)[], int size)
{
    srand(time(NULL));
    for (int i = size - 1; i >= 0; i--)
    {
        int j = rand() % (i + 1);
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    return;
}
bool write_file(int (&board)[SIZE + 1][SIZE + 1], int num, char name[])
{
    FILE *fp;
    if (fopen_s(&fp, name, "w"))
    {
        printf(" Fail!\n");
        return ERROR;
    }
    fprintf(fp, "c %s\n", name);
    fprintf(fp, "p cnf 729 %d\n", num + 12654);
    for (int i = 1; i <= SIZE; i++)
    {
        for (int j = 1; j <= SIZE; j++)
        {
            if (board[i][j] != 0)
                fprintf(fp, "%d 0\n", (i - 1) * SIZE * SIZE + (j - 1) *
SIZE + board[i][j]);
        }
    }
    for (int i = 1; i <= SIZE; i++) // 每个格子必须填入一个数字
    {

```

```

        for (int j = 1; j <= SIZE; j++)
        {
            for (int k = 1; k <= SIZE; k++)
            {
                fprintf(fp, "%d ", (i - 1) * SIZE * SIZE + (j - 1) * SI
ZE + k);
            }
            fprintf(fp, "0\n");
        }
    }
    for (int i = 1; i <= SIZE; i++) // 每个格子不能填入两个数字
    {
        for (int j = 1; j <= SIZE; j++)
        {
            for (int k = 1; k <= SIZE; k++)
            {
                for (int l = k + 1; l <= SIZE; l++)
                {
                    fprintf(fp, "%d %d 0\n", 0 - ((i - 1) * SIZE * SIZE
+ (j - 1) * SIZE + k), 0 - ((i - 1) * SIZE * SIZE + (j - 1) * SIZE + l
));
                }
            }
        }
    }
    for (int i = 1; i <= SIZE; i++) // 每一行必须填入 1~9
    {
        for (int j = 1; j <= SIZE; j++)
        {
            for (int k = 1; k <= SIZE; k++)
            {
                fprintf(fp, "%d ", (i - 1) * SIZE * SIZE + (k - 1) * SI
ZE + j);
            }
            fprintf(fp, "0\n");
        }
    }
    for (int i = 1; i <= SIZE; i++) // 每一行不能填入两个相同的数字
    {
        for (int j = 1; j <= SIZE; j++)
        {

```

```

        for (int k = 1; k <= SIZE; k++)
        {
            for (int l = k + 1; l <= SIZE; l++)
            {
                fprintf(fp, "%d %d 0\n", 0 - ((i - 1) * SIZE * SIZE
+ (k - 1) * SIZE + j), 0 - ((i - 1) * SIZE * SIZE + (l - 1) * SIZE + j
));
            }
        }
    }
}
for (int i = 1; i <= SIZE; i++) // 每一列必须填入 1-9
{
    for (int j = 1; j <= SIZE; j++)
    {
        for (int k = 1; k <= SIZE; k++)
        {
            fprintf(fp, "%d ", (k - 1) * SIZE * SIZE + (i - 1) * SI
ZE + j);
        }
        fprintf(fp, "0\n");
    }
}
for (int i = 1; i <= SIZE; i++) // 每一列不能填入两个相同的数字
{
    for (int j = 1; j <= SIZE; j++)
    {
        for (int k = 1; k <= SIZE; k++)
        {
            for (int l = k + 1; l <= SIZE; l++)
            {
                fprintf(fp, "%d %d 0\n", 0 - ((k - 1) * SIZE * SIZE
+ (i - 1) * SIZE + j), 0 - ((l - 1) * SIZE * SIZE + (i - 1) * SIZE + j
));
            }
        }
    }
}
}
for (int i = 1; i <= SIZE; i += 3) // 每个 3x3 宫格必须填入 1-9
{
    for (int j = 1; j <= SIZE; j += 3)

```

```

    {
        for (int k = 1; k <= SIZE; k++)
        {
            for (int l = 0; l < 3; l++)
            {
                for (int m = 0; m < 3; m++)
                {
                    fprintf(fp, "%d ", ((i + l - 1) * SIZE * SIZE +
(j + m - 1) * SIZE + k));
                }
            }
            fprintf(fp, "0\n");
        }
    }
}
for (int i = 1; i <= SIZE; i += 3) // 每个3x3 宫格不能填入两个一样的数
字
{
    for (int j = 1; j <= SIZE; j += 3)
    {
        for (int k = 1; k <= SIZE; k++)
        {
            for (int l = 0; l < 3; l++)
            {
                for (int m = 0; m < 3; m++)
                {
                    for (int n = k + 1; n <= SIZE; n++)
                    {
                        fprintf(fp, "%d %d 0\n", 0 - ((i + l - 1) *
SIZE * SIZE + (j + m - 1) * SIZE + k), 0 - ((i + l - 1) * SIZE * SIZE
+ (j + m - 1) * SIZE + n));
                    }
                }
            }
        }
    }
}
}
for (int k = 1; k <= SIZE; k++) // 副对角线上要有1~9
{
    for (int i = 1; i <= SIZE; i++)
    {

```



```

        int hang = i;
        int lie = SIZE - i + 1;
        fprintf(fp, "%d ", (hang - 1) * SIZE * SIZE + (lie - 1) * S
SIZE + k);
    }
    fprintf(fp, "0\n");
}
for (int k = 1; k <= SIZE; k++) // 不能重复副对角线上的数字
{
    for (int p = 1; p <= SIZE; p++)
    {
        for (int q = p + 1; q <= SIZE; q++)
        {
            int row1 = p, col1 = SIZE - p + 1;
            int row2 = q, col2 = SIZE - q + 1;
            fprintf(fp, "%d %d 0\n", 0 - ((row1 - 1) * SIZE * SIZE
+ (col1 - 1) * SIZE + k),
                    0 - ((row2 - 1) * SIZE * SIZE + (col2 - 1) * SI
ZE + k));
        }
    }
}
for (int k = 1; k <= SIZE; k++) // 百分号窗口中要有1~9
{
    for (int idx = 0; idx < 9; idx++)
    {
        int i = percentA[idx][0];
        int j = percentA[idx][1];
        fprintf(fp, "%d ", (i - 1) * SIZE * SIZE + (j - 1) * SIZE +
k);
    }
    fprintf(fp, "0\n");
    for (int idx = 0; idx < 9; idx++)
    {
        int i = percentB[idx][0];
        int j = percentB[idx][1];
        fprintf(fp, "%d ", (i - 1) * SIZE * SIZE + (j - 1) * SIZE +
k);
    }
    fprintf(fp, "0\n");
}
}

```

```

for (int k = 1; k <= SIZE; k++) // 窗口内不能有两个一样的数字
{
    for (int a = 0; a < 9; a++)
    {
        for (int b = a + 1; b < 9; b++)
        {
            int i1 = percentA[a][0], j1 = percentA[a][1];
            int i2 = percentA[b][0], j2 = percentA[b][1];
            fprintf(fp, "%d %d 0\n", 0 - ((i1 - 1) * SIZE * SIZE +
(j1 - 1) * SIZE + k), 0 - ((i2 - 1) * SIZE * SIZE + (j2 - 1) * SIZE + k
));
        }
    }
    for (int a = 0; a < 9; a++)
    {
        for (int b = a + 1; b < 9; b++)
        {
            int i1 = percentB[a][0], j1 = percentB[a][1];
            int i2 = percentB[b][0], j2 = percentB[b][1];
            fprintf(fp, "%d %d 0\n", 0 - ((i1 - 1) * SIZE * SIZE +
(j1 - 1) * SIZE + k), 0 - ((i2 - 1) * SIZE * SIZE + (j2 - 1) * SIZE + k
));
        }
    }
}
fclose(fp);
return 1;
}

bool solve_sudoku(int (&board)[SIZE + 1][SIZE + 1], bool (&value)[SIZE
* SIZE * SIZE + 1])
{
    for (int i = 0; i < SIZE * SIZE * SIZE + 1; i++)
    {
        if (value[i])
        {
            int hang = (i - 1) / (SIZE * SIZE) + 1;
            int lie = (i - 1) / SIZE % SIZE + 1;
            int val = (i - 1) % SIZE + 1;
            board[hang][lie] = val;
        }
    }
}

```

```
    return 1;  
}
```