華中科技大學

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：计算机 2407
学　　号：U202414728
姓　　名：谈家能
指导教师：李丹
报告日期：2025.10.1

计算机科学与技术学院

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：$[(t-t_o)/t]*100\%$,其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t_o 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)

（6）SAT 应用：将数双独游戏[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%)

口 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

目录

1 引言

1.1 课题背景与意义

1.1.1 问题背景

SAT 问题又称命题逻辑公式的可满足性问题（satisfiability problem），是判断对合取范式形式给出的命题逻辑公式是否存在一个真值指派使得该逻辑公式为真。SAT 问题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。看似简单，却可广泛应用于许多实际问题如人工智能、电子设计自动化、自动化推理、硬件设计、安全协议验证等，具有重要理论意义与应用价值。对于 SAT 问题的研究从没有停止过，在 1997 年和 2003 年，H.Kautz 与 B.Selman 两次列举出 SAT 搜索面临的挑战性问题，并于 2011 年和 2007 年，两度对当时的 SAT 问题研究现状进行了全面的综述。黄文奇提出的 Solar 算法在北京第三届 SAT 问题快速算法比赛中获得第一名。对 SAT 问题的求解主要有完备算法和不完备算法两大类。不完备算法主要是局部搜索算法，这种算法不能保证一定找到解，但是求解速度快，对于某些 SAT 问题的求解，局部搜索算法要比很多完备算法更有效。完备算法出现的时间更早，优点是可以正确判断 SAT 问题的可满足性，在算例无解的情况下可以给出完备的证明。对于求解 SAT 问题的优化算法主要有启发式算法、冲突子句学习算法、双文字监视法等。

1.1.2 研究意义

SAT 问题是第一个被证明的 NP 完全问题，而 NP 完全问题由于其极大的理论价值和困难程度，破解后将会在许多领域得到广泛应用，从而在计算复杂性理论中具有非常重要的地位。由于所有的 NP 完全问题都能够在多项式时间内进行转换，那么如果 SAT 问题能够得到高效解决，所有的 NP 完全问题都能够在多项式时间内得到解决。对 SAT 问题的求解，可用于解决计算机和人工智能领域内的 CSP 问题（约束满足问题）、语义信息的处理和逻辑编程等问题，也可用于解决计算机辅助设计领域中的任务规划与设计、三维物体识别等问题。SAT 问题的应用领域非常广泛，还能用于解决数学研究和应用领域中的旅行商问题和逻辑算数问题。许多实际问题，例如数据库检索、积木世界规划、超大规模集成电路设计、人工智能等都可以转换成 SAT 问题进而进行求解。可见对 SAT 问题求解的研究，具有重大意义。

1.2 国内外研究现状

SAT 问题作为计算机科学中的核心问题之一，国内外研究均取得了显著进展。国外在 SAT 问题研究方面起步早,理论与实践成果丰富,SAT 求解技术不断优化，如基于 DPLL 算法的改进，以及 Glucose SAT-Solver 等新求解器的出现，广泛应用于各技术创新领域。国内对 SAT 问题的研究也在深入推进，中国科学院软件研究所研发的电路 SAT 求解器 X-SAT，性能在常用数据集上大幅领先现有电路 SAT 求解器，展现了国内在该领域的强大研究实力。

1.3 课程设计的主要研究工作

DPLL算法是经典的SAT完备型求解算法，对给定的一个SAT问题实例，理论上可判定其是否满足，满足时可给出对应的一组解。本设计要求实现基于DPLL的算法与程序框架，包括程序的改进也必须在此算法的基础上进行。

2 系统需求分析与总体设计

2.1 系统需求分析

本设计要求实现一个高效 SAT 求解器，需具备输入输出功能（处理程序参数、读取 cnf 文件、输出并保存结果），能解析与验证 SAT 算例文件并建立公式内部表示，基于 DPLL 算法框架（不使用 C++现有 vector 等类库）求解中小规模算例，测量并输出以毫秒为单位的求解时间，对基本 DPLL 在存储结构或分支变元选取策略等方面进行优化并提供性能优化率结果，还需将百分号数独游戏问题转化为 SAT 问题集成到求解器中，实现简单交互性的数独求解功能。

2.2 系统总体设计

该系统包含五个模块，其中CNF文件读取模块负责解析和验证CNF格式文件，将其转换为自定义物理存储结构并提供核对功能；DPLL算法模块基于自定义数据结构实现SAT求解，集成时间测量和优化功能以提升性能；主程序交互模块作为系统入口，处理用户参数并协调各模块工作流程；生成解文件模块将求解结果按标准格式写入文件并在控制台展示关键信息；数独游戏模块实现百分号数独到SAT问题的转化，通过调用核心模块完成求解并提供简单交互界面。各模块间，主程序交互模块协调整体流程，CNF文件读取模块为DPLL算法模块和数独游戏模块提供公式解析支持，数独游戏模块通过生成临时CNF文件与核心模块交互，生成解文件模块接收DPLL结果并完成输出存储。如图2-1为系统的关系图。

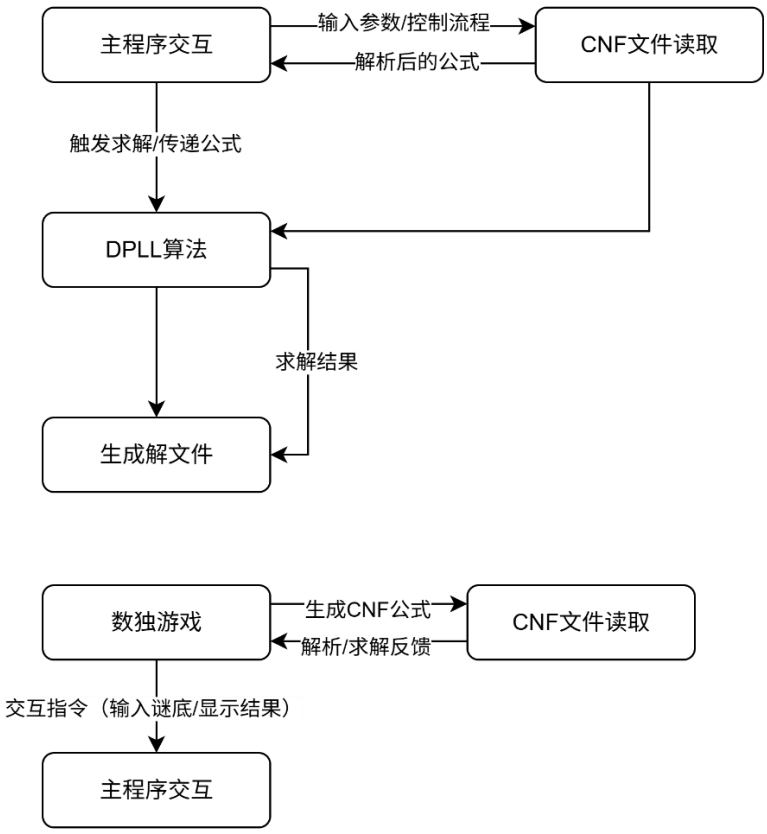

图 2-1 系统各部分关系图

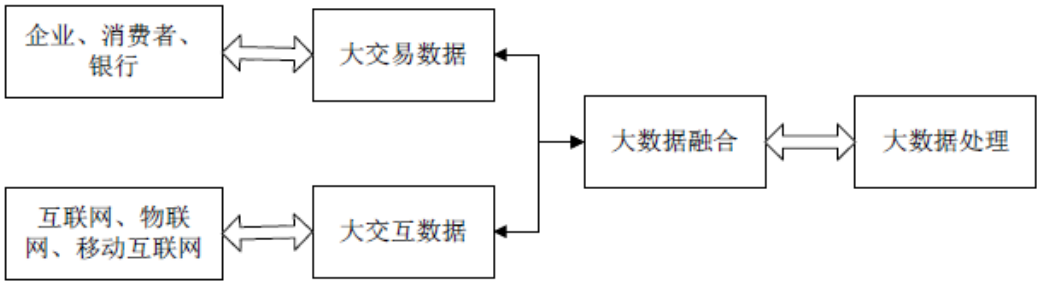

图 3-1 ××××××××××

3 系统详细设计

3.1 有关数据结构的定义

```
#define TRUE 1
#define FALSE 0
#define NOMAL -1
#define ROW 9
#define COL 9
#define NoAnwser -1
typedef int status;
static int TestNum = 0;//用于测试变量
```

//每个结点的数据类型，value表示文字的内容，next指向同一个子句中的下一个文字

```
typedef struct DataNode{
    int value = 0;
    struct DataNode* next;
}DataNode;

typedef struct HeadNode{
    int num = 0;
    struct HeadNode* down;//头结点向下
    struct DataNode* right;//同一个子句中结点向右指针
}HeadNode;
```

系统主要处理两类核心数据：`DataNode`（文字节点）和`HeadNode`（子句头节点），辅以全局常量（如`TRUE`、`ROW`）、自定义状态类型`status`及测试变量`TestNum`。`DataNode`包含表示文字内容的`value`（int型）和指向同子句下一文字的`next`指针；`HeadNode`包含子句相关计数`num`（int型）、指向下方子句头的`down`指针和指向本子句首个文字的`right`指针。

这些数据通过指针形成二维链表关联：横向层面，`HeadNode`的`right`指针串联同一子句的`DataNode`（通过`next`指针延续）；纵向层面，

`HeadNode`的`down`指针连接不同子句的头节点，整体构成存储CNF公式的结构化数据模型，全局常量和状态变量则用于流程控制与状态标识。

3.2 主要算法设计

这部分主要描述系统中模块实现的流程，可采用文字配合流程图的方式表示各模块的算法思想及流程。

系统核心流程围绕用户交互与CNF公式/数独问题的处理展开，主要包含用户交互模块、CNF解析模块、DPLL求解模块、数独游戏模块和结果验证/输出模块，各模块通过函数调用协同工作，具体流程如下：

1. 用户交互模块：程序启动后通过`display()`展示操作菜单（1-解析CNF、2-求解数独、3-验证结果、0-退出），根据用户输入的`cod`值分支执行对应功能。用户选择功能后，程序提示输入必要参数（如文件名、优化选项），并通过循环维持交互直到输入0退出。

2.DPLL算法：首先通过isHaveSingleClause()检测反复查找单文字子句，对找到的文字（SC）进行赋值（正文字赋 1，负文字赋 - 1），并调用SimplifyCnf()化简子句集；若化简后子句集为空（isemptyCnf()）则返回可满足（TRUE），若出现空子句（isHaveEmptyClause()）则返回不可满足（FALSE）。当无单文字子句时，通过SelectWord2()选择一个变元v，递归调用 DPLL 分别尝试对v赋真（合并v到子句集）和赋假（合并-v到子句集），若其中一种赋值可满足则返回 TRUE，否则返回 FALSE，期间通过CopyList()和FreeList()管理临时子句集的内存。

3. 改进改进后的`NewDPLL`算法采用非递归方式实现，通过栈（`stack`）模拟递归过程以优化性能：首先将初始子句链表入栈，循环中从栈顶获取子句集，通过`SingleSpread()`反复进行单文字传播化简；若无法化简，则复制当前子句集（`CopyList()`），用`SelectWord2()`选择变元`v`，将合并`v`后的子句集入栈继续搜索；若化简后发现可满足（`val=TRUE`）则返回真，若不可满足则弹栈并释放当前子句集（`FreeList()`），再将合并`-v`后的子句集入栈尝试另一分支。整个过程通过数组`ListL`记录中间子句集，避免递归调用的栈开销，提升大规模CNF公式的求解效率。

4.数独生成算法通过多步协作完成：首先由`randomFirstRow()`随机生成数独第一行（确保9个数字不重复），再通过`Digit()`递归填充剩余行，填充时检查行列、3×3宫格及额外约束（如对角线、特定3×3宫）避免数字重复，生成完整终盘；

接着`createStartinggrid()`从终盘随机挖空指定数量（`holes`）的格子，挖空时通过临时生成CNF文件并调用`NewDPLL()`验证剩余数字是否保证唯一解，确保初盘有效；最后`ToCnf()`将初盘转换为CNF文件（包含数独规则约束和已知数字），供后续求解使用，`print()`用于展示数独状态。整个过程兼顾随机性与解的唯一性，通过约束检查和算法验证确保数独合法性。4. 结果验证模块（对应`cod=3`）：用户输入待检验文件路径后，`isCorrect()`验证文件内容的正确性，直接在终端返回检验结果。

算法核心思想：以DPLL算法为基础求解CNF公式，通过二维链表存储子句与文字结构；数独问题通过转换为CNF公式求解，结合用户交互实现填数游戏功能，全程通过文件I/O和终端交互完成输入输出与结果验证。

4 系统实现与测试

4.1 系统实现

硬件环境：Intel Core i7-14650HX 处理器、16GB 内存、64 位 x64 系统。

软件环境：VS Code 1.103.2。

该系统的核心函数包括负责展示菜单的`display()`、解析 CNF 文件的`CnfParser()`、基础和优化版本的求解算法`DPLL()`与`NewDPLL()`、整合解析、求解及输出流程的`CoreFun()`、生成数独初盘并转换为 CNF 文件的`createSudokuToFile()`、验证结果正确性的`isTrue()`、输出结果到文件的`OutFileFun1()`以及打印数独的`print()`；这些函数以`main()`为入口，通过用户输入的操作码`cod`分支调用：`cod=1`时依次调用`CnfParser()`解析文件后由`CoreFun()`调用`DPLL()`/`NewDPLL()`求解并通过`OutFileFun1()`输出；`cod=2`时先经`createSudokuToFile()`生成数独 CNF，再通过`CnfParser()`和`CoreFun()`完成求解，过程中调用`print()`展示数独并与用户交互；`cod=3`则直接调用`isCorrect()`验证结果，形成以`CoreFun()`为核心的函数调用链，协同实现系统功能。

系统定义了两种两种核心数据结构用于存储 CNF 公式:`DataNode`表示子句中的单个文字，包含存储文字内容的`value`（int 型）和指向同子句下一文字的`next`指针；`HeadNode`作为子句头节点，包含子句相关计数`num`（int 型）、指向下方子句头节点的`down`指针和指向本子句首个文字的`right`指针，两者通过指针形成“子句链表+文字链表”的二维结构，实现 CNF 公式的结构化存储。

程序详见附录。

4.2 系统测试

首先叙述一下常用的软件测试方法，在选择几个主要的功能模块（自行掌握数量，关键要体现你的水平的一些模块）描述测试过程：

（1）先明确模块的功能、设计目标等；

分析、叙述如何选取测试数据，要求有完整的测试大纲；

（2）运行结果（这时可用截图）；

（3）分析运行结果、确认程序满足该模块的设计目标。

该系统围绕“CNF公式处理”与“数独交互游戏”两大核心目标设计，包含用户交互、CNF解析与求解、数独生成与游戏、结果验证与输出四大模块：用户交互模块通过菜单引导操作，CNF模块由`CnfParser()`解析文件并以`HeadNode`/`DataNode`二维链表存储，再用`DPLL()`/`NewDPLL()`算法求解；数独模块先通过`createSudoku()`生成终盘、`createStartinggrid()`挖空生成唯一解初盘并由`ToCnf()`转CNF文件，再结合`CoreFun()`实现用户填数交互与校验；结果模块通过`isCorrect()`验证合法性、`OutFileFun1()`输出结果到`.res`文件，整体兼顾功能完整性、算法高效性与用户友好性，可支持逻辑推理与数独游戏需求。

系统测试数据选取遵循覆盖性、典型性、可追溯性与有效性原则，测试大纲围绕四大模块展开：CNF解析与求解模块选取标准可满足/不可满足、复杂、边界及格式异常CNF文件，验证解析正确性与算法效率；数独模块测试最小（17个）、中等（40个）、最大（81个）初值数独及边界错误输入，校验生成合法性与交互逻辑；结果验证模块用正确/错误的CNF结果、数独答案及非目标格式文件，测试验证准确性；全局异常模块模拟文件不存在、超大文件、重复操作场景，检验容错能力，整体通过明确数据来源、预期结果与测试步骤，全面验证系统功能与稳定性。

算例测试运行结果

表4-1 算例测试运行结果

算例名称	优化前（ms）	优化后（ms）	优化率
1.cnf	442	387	12.4%
2.cnf	15360	11580	24.6%
3.cnf	15	10	33.3%
4（unsatisfied）.cnf	661	593	10.3%
5.cnf	64	48	25.0%
6.cnf	8007	7455	6.9%
11（unsatisfied）.cnf	97744	71274	27.1%

```
D:\vs123\summer>cs\main.exe
欢迎使用本系统，请选择你要执行的操作，请输入命令
1 -- cnf文件判定是否满足
2 -- 数独问题求解
3 -- 输出文件正确性验证
0 -- 结束程序！
1
请输入需要解析的.cnf文件。（例如：.\src\Sat\S\problem1-20.cnf）
11（unsatisfied）.cnf
请选择是否使用优化的算法
0--未优化
1--优化
0
为假
DPLL()部分运行的时间为97744ms
欢迎使用本系统，请选择你要执行的操作，请输入命令
1 -- cnf文件判定是否满足
2 -- 数独问题求解
3 -- 输出文件正确性验证
0 -- 结束程序！
1
请输入需要解析的.cnf文件。（例如：.\src\Sat\S\problem1-20.cnf）
11（unsatisfied）.cnf
请选择是否使用优化的算法
0--未优化
1--优化
1
为假
DPLL()部分运行的时间为71274ms
欢迎使用本系统，请选择你要执行的操作，请输入命令
1 -- cnf文件判定是否满足
2 -- 数独问题求解
3 -- 输出文件正确性验证
0 -- 结束程序！
0
```

图 4-1 算例 11（unsatisfied）.cnf 运行结果

```
欢迎使用本系统，请选择你要执行的操作，请输入命令
1 -- cnf文件判定是否满足
2 -- 数独问题求解
3 -- 输出文件正确性验证
0 -- 结束程序！
```

图 4-2 交互主界面

```
欢迎使用本系统，请选择你要执行的操作，请输入命令
1 -- cnf文件判定是否满足
2 -- 数独问题求解
3 -- 输出文件正确性验证
0 -- 结束程序!
2
请输入数独初盘中设定数字的个数：（不少于17，不大于81)
75
初始化后数独初盘为:
6 2 4 5 8 9 1 3 7
1 3 5 2 4 7 6 8 9
7 8 9 1 3 6 2 4 5
2 4 6 7 0 3 5 9 8
3 0 7 8 9 5 4 0 0
5 0 8 6 2 4 3 7 1
8 7 1 3 6 2 9 5 4
9 5 2 0 7 1 8 6 3
4 6 3 9 5 8 7 1 2
DPLL()部分运行的时间为120ms
数独游戏开始，您可以随时输入0以终止游戏
请输入您填入的下一个数字，111表示在第一行第一列填入数字1
```

图 4-3 百分号数独游戏的生成

```
数独游戏开始，您可以随时输入0以终止游戏
请输入您填入的下一个数字，111表示在第一行第一列填入数字1
451
6 2 4 5 8 9 1 3 7
1 3 5 2 4 7 6 8 9
7 8 9 1 3 6 2 4 5
2 4 6 7 1 3 5 9 8
3 0 7 8 9 5 4 0 0
5 0 8 6 2 4 3 7 1
8 7 1 3 6 2 9 5 4
9 5 2 0 7 1 8 6 3
4 6 3 9 5 8 7 1 2
填入成功，请填写下一个
请输入您填入的下一个数字，111表示在第一行第一列填入数字1
```

图 4-4 百分号数独游戏的填入交互

综上，系统运行结果覆盖了设计目标中的功能完整性、算法高效性、用户友好性与可扩展性，各模块输出与预期一致，程序满足设计要求。

5 总结与展望

5.1 全文总结

对自己的工作做个总结，主要工作如下：

（1）学习并理解了 SAT 问题以及 DPLL 算法是如何解决 SAT 问题

（2）实现了用链表的形式存储 CNF 文件并用递归算法求解

（3）用栈的深度优先求索优化了 DPLL 算法

（4）用挖洞法生成百分号数独，实现唯一性

（5）将数独游戏转化为 CNF 文件实现了用 DPLL 算法求解数独

5.1 工作展望

在今后的研究中，围绕着如下几个方面开展工作：

（1）进一步优化 DPLL 算法的变元选择策略，提升对超大规模 CNF 公式的求解速度；

（2）扩展数独游戏功能，如支持多种难度等级、添加错误提示的详细解释（如“该数字与行 / 列 / 宫格冲突”），并引入图形化界面提升用户体验；

6 体会

在本次项目设计与实现的过程中，我深刻体会到理论与实践之间的紧密联系，也积累了不少些关于系统开发的实际经验。

从理论学习到代码实现的转化是第一个需要跨越的门槛。最初理解 DPLL 算法时，虽然能看懂文字描述的“单文字化简”“回溯搜索”等逻辑，但真正用代码实现时，才发现需要解决诸多细节问题——比如如何用链表高效存储动态变化的子句集，如何在化简过程中避免内存泄漏，如何设计变元选择策略才能平衡求解速度与复杂度。这让我意识到，算法的理论框架只是基础，实际落地时必须结合数据结构特性和工程化思维，才能写出高效且稳定的代码。

模块化设计的重要性在项目推进中愈发凸显。项目初期，我曾尝试将 CNF 解析、算法求解、结果输出等功能混编在少数几个函数中，导致后期修改时牵一发而动全身——比如调整数独生成规则时，竟需要修改 CNF 转换的核心逻辑。后来通过拆分模块（如独立的`CnfParser()`负责解析、`CoreFun()`负责流程控制），不仅让代码结构更清晰，也大幅降低了功能扩展的难度。这种“分而治之”的思想，既是工程实践的技巧，也是应对复杂问题的有效策略。

算法优化的过程充满了试错与思考。最初实现的递归版 DPLL 算法在处理大规模 CNF 文件时，常因递归深度过深导致栈溢出。为解决这一问题，我尝试用栈模拟递归过程，设计非递归版`NewDPLL()`算法。在调试过程中，通过对比两种算法的运行日志，发现递归的本质是系统自动维护调用栈，而手动用栈实现时，需要更精细地管理子句集的复制与释放，否则会出现内存占用过高的问题。这让我明白，优化并非简单替换实现方式，而是要理解不同方案的底层原理，才能在效率与稳定性之间找到平衡。

此外，用户体验的细节打磨同样不可或缺。在数独游戏模块中，最初仅通过控制台输出“输入错误”，但实际测试时发现用户常因不清楚错误原因而反复尝试。后来添加了具体提示（如“该位置已填数字”“与行中数字冲突”），并优化了数独打印格式（用分隔线区分宫格），显著提升了交互流畅性。这让我意识到，即使是工具类程序，也需要站在用户角度思考，细节的完善往往能让系统从“可用”提升到“易用”。

整个项目的推进，既是对 SAT 问题求解、数据结构等知识的综合运用，也是对问题拆解、调试排错、迭代优化等工程能力的锻炼。每一次遇到 bug（如链表指针悬空导致的程序崩溃、CNF 转换时的约束遗漏），都是一次深入理解系统逻辑的机会。这些经历让我深刻认识到，编程不仅是代码的堆砌，更是逻辑思维与工程实践的结合——只有兼顾理论深度与实现细节，才能开发出既高效又实用的系统。

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

all.h

```
#ifndef all_h
#define all_h
#include<iostream>
#include<fstream>
#include<ctime>
#include<cmath>
#include<stack>
#include<string>
#include<cstring>
#define TRUE 1
#define FALSE 0
#define NOMAL -1
#define ROW 9
#define COL 9
#define NoAnwser -1
typedef int status;
using namespace std;

static int TestNum = 0;//用于测试变量

//每个结点的数据类型，value 表示文字的内容，next 指向同一个子句中的下一个文字
typedef struct DataNode{
    int value = 0;
    struct DataNode* next;
}DataNode;

typedef struct HeadNode{
    int num = 0;
    struct HeadNode* down;//头结点向下
    struct DataNode* right;//同一个子句中结点向右指针
}HeadNode;

HeadNode* CnfParser(string& filename);
void display();
bool DPLL(HeadNode* L, int* book);//核心算法部分
```

```
bool NewDPLL(HeadNode* L, int* book);
int SingleSpread(HeadNode* L, int* book);
int isHaveSingleClause(HeadNode* L);
int isHavePlainText(HeadNode* L);
void SimplifyCnf(HeadNode* L, int SC);
bool isemptyCnf(HeadNode* L);
bool isHaveEmptyClause(HeadNode* L);
HeadNode* merge(HeadNode* L, int v);
int SelectWord(HeadNode* L);
int SelectWord2(HeadNode* L);
void OutFileFun1(string _PATH, bool suc, int* book, int _Ctime, int FunNum, int cod);
void printList(HeadNode* L);
HeadNode* CopyList(HeadNode* L);
void FreeList(HeadNode* L);

int Digit(int a[][COL], int i, int j);
void randomFirstRow(int a0[], int n);
void createSudoku(int a[][COL]);
void createStartinggrid(const int a[][COL], int b[][COL], int numDigits);
void print(const int a[][COL]);
string ToCnf(int a[][COL],int holes);
string createSudokuToFile(int holes, int array[ROW][COL]);

void CoreFun(HeadNode* L, string& filename, int FunNum, int cod, int
array[ROW][COL]);
bool isCorrect(string& filename);

#endif
```

CnfParser.cpp

```
#include "all.h"
int FunNum;//记录 cnf 中变元个数
HeadNode* CnfParser(string& filename){
    string PATH=filename;
    ifstream fp(PATH);
    if(!fp){cout << "File can not open"; exit(0);}
    char ch;
    char exp[105];//exp:对 cnf 文件的解释 explain
    fp >> ch;
    while(ch != 'p'){
```

```
        fp.getline(exp, 100);
        //注意：此处的 exp 只能是用 c-string（数组），不能用 string
        //详细解释:n:Pointer to an array of characters where extracted characters are stored as a c-string.
        fp >> ch;
    }
    string cnf; int ClauseNum;//分别存储 CNF、变元个数、子句个数。
    fp >> cnf >> FunNum >> ClauseNum;
    //测试用例：cout << ClauseNum << " " << FunNum << endl;
    //下面循环将所有的子句读入到设计好的数据结构中
    HeadNode* L = new HeadNode;//L 为 HeadNode 结点的头结点
    L->num = ClauseNum;//字句个数（也就是有多少行）
    L->right=nullptr;
    HeadNode* pH = new HeadNode; L->down = pH; pH->right = nullptr; pH->down = nullptr;//L 中 num 表示有多少行
    for(int i = 1; i <= ClauseNum; i++){
        int tep;
        fp >> tep;
        DataNode* p = new DataNode;
        p->value = tep; pH->right = p; pH->num++; p->next = nullptr;
        fp >> tep;
        while(tep){
            p = new DataNode;
            p->value = tep;//采用头插法建立链表
            p->next = pH->right;
            pH->right = p;
            pH->num++;
            fp >> tep;
        }
        if(i != ClauseNum){
            pH = new HeadNode;
            pH->down = L->down;
            L->down = pH;
        }
        //fp.get();//get 掉换行符
    }
    fp.close();//关闭文件流

    //将解析后的 cnf 文件进行输出表示，用于进行代码块测试
```

```
    // HeadNode* _pH = L->down;

    // while(_pH){
    //     DataNode* _p = _pH->right;
    //     while(_p->next){
    //         cout << _p->value << " ";
    //         _p = _p->next;
    //     }
    //     cout << _p->value << endl;
    //     _pH = _pH->down;
    // }

    return L;
}
void FreeList(HeadNode* L){
    // 先检查 L 是否为空，避免访问空指针
    if (L == nullptr) return;

    HeadNode* _pH = L->down;
    while(_pH != nullptr){   // 用 nullptr 更规范（C++11+）
        // 释放当前 HeadNode 对应的所有 DataNode
        DataNode* _p = _pH->right;
        while(_p != nullptr){   // 遍历所有 DataNode，包括最后一个
            DataNode* _fp = _p;   // 保存当前节点
            _p = _p->next;          // 先移动到下一个节点
            delete _fp;             // 再释放当前节点（无论 next 是否为空）
            _fp=nullptr;
        }

        // 释放当前 HeadNode
        HeadNode* _fpH = _pH;
        _pH = _pH->down;   // 移动到下一个 HeadNode
        delete _fpH;
        _fpH=nullptr;
    }

    // 最后释放头节点 L
    delete L;
    L=nullptr;
```

```
}
```

display.cpp

```
#include"all.h"

void display(){
    cout << "欢迎使用本系统，请选择你要执行的操作，请输入命令" << endl;
    cout << "1 -- cnf 文件判定是否满足" << endl;
    cout << "2 --  数独问题求解" << endl;
    cout << "3 --  输出文件正确性验证" << endl;
    cout << "0 --  结束程序！ " << endl;
}
```

DPLL.cpp

```
#include"all.h"
extern int FunNum;
bool DPLL(HeadNode* L,int* book){
    int SC;//SingleClause?
    while(SC=isHaveSingleClause(L)){
        if(SC>0) book[SC]=1;
        else if(SC<0) book[0-SC]=-1;
        SimplifyCnf(L,SC);
        if(isemptyCnf(L)) return TRUE;
        else if(isHaveEmptyClause(L)) return FALSE;
    }
    int v = SelectWord2(L);
    // int v = SelectWord2(L);//采用最短子句优先选取规则进行选取变元
    //cout << "这次选取的变元为：" << v << endl;
    HeadNode* temL = CopyList(L);
    if(DPLL(merge(L, v), book)) return TRUE;
    bool result = DPLL(merge(temL, 0-v), book);
    //删除 temL 的内容空间
    FreeList(temL);
    return result;
}

bool NewDPLL(HeadNode* L, int* book){//非递归版本 DPLL 算法
    stack<HeadNode*> S;
```

```
    HeadNode* ListL[10000] = {nullptr}; int i = 0;//ListL 用来记录每次改变的链表结点内容
    HeadNode* temL = nullptr;
    int v[10000]; int val;
    S.push(L);
    while(!S.empty()){
        while((L = S.top()) && (val = SingleSpread(L, book)) == NOMAL){
            //cout << "循环开始" << endl; printList(L);
            temL = CopyList(L);
            ListL[i] = temL;
            v[i] = SelectWord2(L);
            //cout << "选取的变元为：" << v[i] << endl;
            L = merge(L, v[i]);
            S.push(L);i++;
        }
        //printList(L); cout << "循环结束" << endl;
        if(val == TRUE) return TRUE;
        S.pop();i--;
        //cout << "此处的变元为：" << v[i] << endl;
        FreeList(L);
        //cout << "栈的大小为" << S.size() << endl;
        if(!S.empty()){
            L = ListL[i]; S.pop();
            S.push(merge(L, -v[i]));
        }
    }
    return FALSE;
}

int SingleSpread(HeadNode* L, int* book){
    int SC;//存储单子句(SingleClause 的缩写)
    //cout << ++TestNum << endl;
    //printList(L);
    if(!(SC=isHaveSingleClause(L)))
    {
        SC=isHavePlainText(L);
    }
    while(SC){
        //cout << "2-" << TestNum << "  本次循环要删除的为:" << SC << endl;
```

```
            if(SC > 0) book[SC] = 1;
            else if(SC < 0) book[0-SC] = -1;//例如：book[-(-2)] = -1 表示 2 代表的变元取负值，即-2
            SimplifyCnf(L, SC);//根据单子句传播策略简化单子句
           //printList(L);
            if(isemptyCnf(L)) return TRUE;
            else if(isHaveEmptyClause(L)) return FALSE;

            if(!(SC=isHaveSingleClause(L)))
            {
                SC=isHavePlainText(L);
            }
        }
        return NOMAL;
}
int isHavePlainText(HeadNode* L){//判断 L 中是否含有纯文字
        int* mark1 = new int[FunNum + 1]();//标记数组，标记每个变元出现的正负情况
        int* mark2 = new int[FunNum + 1]();
        HeadNode* _pH = L->down;
        while(_pH){
            DataNode* _p = _pH->right;
            while(_p){
                if(_p->value > 0) mark1[_p->value] = 1;
                else if(_p->value < 0) mark2[0 - _p->value] = 1;
                _p = _p->next;
            }
            _pH = _pH->down;
        }
        for(int i = 1; i <= FunNum; i++){
            if(mark1[i]==1&&mark2[i]==0){ delete[] mark1;delete[] mark2;return i;}
            else if(mark1[i]==0&&mark2[i]==1) {delete[] mark1;delete[] mark2;return 0-i;}
        }
        delete[] mark1;delete[] mark2;
        return 0;
}
//L 中存储的合区范式是否含有单子句（只含有一个文字的子句叫做单子句）
//如果含有单子句，则返回单子句的值
```

```
//如果不含有单子句，则返回 0
int isHaveSingleClause(HeadNode* L){
    HeadNode* _pH = L->down;
    while(_pH){
        if(_pH->num == 1) return _pH->right->value;
        _pH = _pH->down;
    }
    return 0;
}

//根据单子句 SC 进行化简范式 L——含有 SC 的子句直接去掉，含有-SC 的子句中去掉-SC 这个文字
void SimplifyCnf(HeadNode* L, int SC){
    HeadNode* _pH = L->down;
    HeadNode* _fpH = L;//记录之前遍历过的结点
    int flag;
    while(_pH){
        DataNode* _p = _pH->right;
        DataNode* _fp = _p;
        flag = 0;
        while(_p){
            flag = 0;
            if(_p->value == SC){//去掉这个子句（删除一行）
                _fpH->down = _pH->down;
                _p = _pH->right;
                while(_p){
                    _fp = _p;
                    _p = _p->next;
                    delete _fp;
                }
                L->num--;
                flag = 1;
                break;//跳过这一行
            }
            else if(_p->value == 0-SC){//去掉这个文字（删除一个点，需要考虑第一个节点的特殊情况）
                if(_pH->right == _p){
                    flag = 2;//第一个结点就出现目标数据，需要删除
                    _pH->right = _p->next;
```

```
                        _pH->num--;
                        //cout << "第一个结点处地文字删除后 num 的值为：" <<
_pH->num << endl;
                    }
                    else{
                        flag = 3;
                        _fp->next = _p->next;
                        _pH->num--;
                    }
                }
                if(!flag){//没有需要删除的节点
                    if(_fp != _p) _fp = _p;
                    _p = _p->next;
                }
                else if(flag == 2){//删除的结点为第一个节点的情况处理
                    _p = _pH->right;
                    _fp = _p;
                }
                else if(flag == 3){//删除的节点不是第一个，在中间
                    delete _p;
                    _p = _fp->next;
                }
            }
            if(!flag){
                _fpH = _pH;
                _pH = _pH->down;
            }
            else if(flag == 1){
                delete _pH;
                _pH = _fpH->down;
            }
        }
}

//L 中存储的范式是否为空范式
bool isemptyCnf(HeadNode* L){
    if(L->num) return FALSE;
    else return TRUE;
}
```

```
//判断范式中是否有空子句
bool isHaveEmptyClause(HeadNode* L){
    HeadNode* _pH = L->down;
    while(_pH){
        if(_pH->num == 0) return TRUE;
        _pH = _pH->down;
    }
    return FALSE;
}

//将 v 作为一个单子句合并到范式 L 中，将合并后的范式作为返回值
HeadNode* merge(HeadNode* L, int v){
    HeadNode* pH = new HeadNode; pH->num++;
    pH->down = L->down;
    L->down = pH;
    DataNode* p = new DataNode; p->value = v;
    pH->right = p; p->next = nullptr;
    L->num++;
    //cout << "merge 完之后添加的 v 为：" << L->down->right->value << endl;
    return L;
}

//直接返回第一个文字即可
int SelectWord(HeadNode* L){
    HeadNode* _pH = L->down;
    int ans;
    while(_pH){
        DataNode* _p = _pH->right;
        while(_p){
            ans = _p->value;
            return ans;
        }
        _pH = _pH->down;
    }
    cout << "范式为空集，无法选择文字返回" << endl;
    return 0;
}

int SelectWord2(HeadNode* L){//最短子句优先
```

```
    HeadNode* _pH = L->down;
    int _min, ans = 0;
    if(!_pH){cout << "范式为空集，无法选择文字返回" << endl; return 0;}
    _min = _pH->num; ans = _pH->right->value;
    while(_pH){
        if(_pH->num != 0 && _pH->num < _min) ans = _pH->right->value;
        _pH = _pH->down;
    }
    return ans;
}

//仅用于测试使用
void printList(HeadNode* L){
    HeadNode* _pH = L->down;
    int i = 1;
    while(_pH){
        cout << "第" << i++ << "行：" << _pH->num;
        DataNode* _p = _pH->right;
        while(_p){
            cout << "->" << _p->value;
            _p = _p->next;
        }
        cout << endl;
        _pH = _pH->down;
    }
}

//将 L 中全部内容都拷贝到 temL 中，函数结构与 cnfparser 函数基本相同
HeadNode* CopyList(HeadNode* L){
    HeadNode* temL = new HeadNode;
    temL->num = L->num;
    HeadNode* pH = L->down;
    HeadNode* _pH = new HeadNode; temL->down = _pH; _pH->down = nullptr;
    for(int i = 1; i <= L->num; i++){
        _pH->num = pH->num;
        int tep = pH->right->value;
        DataNode* p = pH->right;
        DataNode* _p = new DataNode;
        _p->value = tep; _pH->right = _p; _p->next = nullptr;
        while(p->next){
```

```
                p = p->next; tep = p->value;
                _p = new DataNode;
                _p->value = tep;//采用头插法建立链表
                _p->next = _pH->right;
                _pH->right = _p;
            }
            if(i != L->num){
                _pH = new HeadNode;
                _pH->down = temL->down;
                temL->down = _pH;
            }
            pH = pH->down;
        }
        return temL;
}
```

isCorrect.cpp

```
#include"all.h"
extern int FunNum;
bool verification(HeadNode* ordL, int* book);
//验证根据 DPLL 算法生成的文件是否正确
bool isCorrect(string& filename){
    HeadNode* L = CnfParser(filename);
    HeadNode* ordL = CopyList(L);

    int _FunNum = FunNum>1000? FunNum: 1000;
    int *book = new int[_FunNum];
    memset(book, 0, sizeof(int)*_FunNum);

    int isTrue = NewDPLL(L, book);
    if(!isTrue){cout << ".cnf 文件不可满足"; delete []book;exit(0);}
    else{
```

```
        if(verification(ordL, book)){ delete []book;return TRUE;}
        else {delete []book;return FALSE;}
    }

}
/*
解析文件，生成链表 L，同时拷贝一份不会改变的 ordL;（不会改变的 L）
执行 DPLL 算法，生成.res 文件
如果 DPLL 算法返回为假，直接退出就行，为真进行下一步
直接遍历 ordL 结构，访问 book 对应的文字的值
如果有一个子句的所有的文字都不会满足，那么该解就是有问题
*/
bool verification(HeadNode* ordL, int* book){
    HeadNode* _pH = ordL->down;
    while(_pH){
        DataNode* _p = _pH->right;
        while(_p){
            if(book[abs(_p->value)]*_p->value > 0) break;
            _p = _p->next;
        }
        if(!_p) return FALSE;
        _pH = _pH->down;
    }
    return TRUE;
}
```

main.cpp

```
#include "all.h"
```

```
extern int FunNum;
int zeroNum;
int isBetter;
int main() {
    display();
    int cod = 0;
    string filename;
    cin >> cod;
    while (cod){
        if(cod == 1){//cnf 解析求解
            cout << "请输入需要解析的.cnf 文件。（例如：.\\src\\Sat\\S\\problem1-20.cnf）" <<endl;
            cin >> filename;
            cout << "请选择是否使用优化的算法" << endl;
            cout << "0--未优化" << endl; cout << "1--优化" << endl;
            cin >> isBetter;
            HeadNode* L = CnfParser(filename);
            int temp[9][9];
            CoreFun(L, filename, FunNum, cod, temp);
        }
        else if(cod == 2){//数独问题求解
            cout << "请输入数独初盘中设定数字的个数：（不少于 17，不大于 81）" << endl;
            int preNum; cin >> preNum; zeroNum = 81-preNum;
            int array[ROW][COL] = {0};
            filename = createSudokuToFile(zeroNum, array);
            HeadNode* L = CnfParser(filename);
            //printList(L);
```

```
            CoreFun(L, filename, 1000, cod, array);
            /*
            设计用户交互的部分
            请求用户输入填一个空，并指明填写的数字格式以及对应意义
                读入的数字为零，直接退出程序
                将用户输入的数字与 book 中的数进行比对，查看是否正确
                如果不正确的话，显示错误重新输入
                正确的话，就读入数字，然后把用户输入的地方的那个数字修改之后把数独表打印出来显示给用户
            如果数独中空元的个数为零，通知用户成功过关，并询问是否可以再次重新开始游戏
            */
        }
        else if(cod == 3){
            cout << "请输入待检验的文件名（例如：.\\src\\Sat\\S\\problem1-20.cnf）" << endl;
            cin >> filename;
            if(isCorrect(filename)) {cout << "输出的结果没问题" << endl;}
            else {cout << "输出的结果不对" << endl;}
        }
        display();
        TestNum = 0;
        cin >> cod;
    }
}

void CoreFun(HeadNode* L, string& filename, int FunNum, int cod, int array[ROW][COL]){
```

```
//文件输出准备部分
string _PATH = filename.replace(filename.end()-4, filename.end(), ".res");
bool suc;
int *book = new int[FunNum];
memset(book, 0, sizeof(int)*FunNum);
int begin = 0, end = 0;

//DPLL()核心处理部分
begin = clock();
bool isTrue;
if(!isBetter) {isTrue = DPLL(L, book);}
else {isTrue = NewDPLL(L, book);}
if(isTrue) {suc = TRUE;}
else {suc = FALSE;}
if(cod == 1){
    if(isTrue) {cout << "为真" << endl;}
    else {cout << "为假" << endl;}
    end = clock();

    cout << "DPLL()部分运行的时间为" << end-begin << "ms" << endl;
}
else{
    end = clock();
    cout << "DPLL()部分运行的时间为" << end-begin << "ms" << endl;
    cout << "数独游戏开始，您可以随时输入 0 以终止游戏" << endl;
    while(zeroNum){//如果还有空元
```

```
            cout << "请输入您填入的下一个数字，111 表示在第一行第一列填入数字 1" << endl;
            int opeNum; cin >> opeNum;
            int x = opeNum/100 - 1; int y = (opeNum/10)%10 - 1;
            if(!opeNum){exit(0);}
            if(book[opeNum] > 0 && array[x][y] == 0){
                array[x][y] = (opeNum%100)%10;
                zeroNum--;
                print(array);
                cout << "填入成功，请填写下一个" << endl;
            }
            else{
                cout << "输入的数字不正确！" << endl;
                continue;
            }
        }
        cout << "游戏结束，恭喜您成功过关！" << endl;
    }
    //输出.res 文件
    OutFileFun1(_PATH, suc, book, end-begin, FunNum, cod);
}
```

OutFile.cpp

```
#include"all.h"

void OutFileFun1(string _PATH, bool suc, int* book, int _Ctime, int FunNum, int cod){
    ofstream op(_PATH);
    if(!op){
```

```
            cout << "OutFile can not open!" << endl;
            exit(0);
        }
        op << "s " << suc << endl;//结果
        if(suc){
            op << "v";
            for(int i = 1, k = 1; i <= FunNum; i++){
                op << " " << book[i]*i;
                k++;
                if(k == 10) {op << endl; k = 1;}
            }
            op << endl;
        }
        if(cod == 1) op << "t " << _Ctime << "ms";
        op.close();
}
```

SudokuSover.cpp

```
#include "all.h"

#define CORRECT 0
#define WRONG -1
static int T = 0;

//根据 holes 来挖洞
//此函数输出数独初盘，同时返回解析后的.cnf 文件
string createSudokuToFile(int holes, int array[ROW][COL]) {
    int sudoku[ROW][COL]={0};
    int starting_grid[ROW][COL]={0};
    createSudoku(sudoku);//生成数独终盘
    createStartinggrid(sudoku,starting_grid,holes);//生成初盘
    memcpy(array, starting_grid, ROW*COL*sizeof(int));
    cout << "初始化后数独初盘为： " << endl;
```

```
    print(starting_grid);//输出初盘
    //转化为 cnf 文件
    string filename = ToCnf(starting_grid,holes);
    return filename;
}

int Digit(int a[][COL], int i, int j) {//递归填充数独元素
    if (i < ROW && j < COL) {
        int x,y,k;
        int check[COL+1]={CORRECT};//用于排除已经使用过的的数字
        for(x = 0 ; x < i ; x++)
            check[a[x][j]] = WRONG;//列已使用的数字置为 WRONG
        for(x = 0 ; x < j ; x++)
            check[a[i][x]] = WRONG;//行使用过的数字置为 WRONG
        for(x = i/3*3 ; x <= i; x++) {
            if(x == i)
                for(y = j/3*3 ; y < j; y++)
                    check[a[x][y]] = WRONG;
            else
                for(y = j/3*3 ; y < j/3*3 + 3; y++)
                    check[a[x][y]] = WRONG;
        }
        if(i+j==8)
        {
            for(x=0,y=8;x<9;x++,y--)
                check[a[x][y]]=WRONG;//斜对角线使用过的数字置为 WRONG
        }
        if(i>0&&j>0&&i<4&&j<4)
        {
            for(x=1;x<4;x++)
                for(y=1;y<4;y++)
                    check[a[x][y]]=WRONG;//左上角 3*3 宫已使用的数字置为
WRONG
        }
                if(i>4&&j>4&&i<8&&j<8)
        {
            for(x=5;x<8;x++)
                for(y=5;y<8;y++)
                    check[a[x][y]]=WRONG;//右下角 3*3 宫已使用的数字置为
```

```
WRONG
        }
        int flag = 0;
        for(k = 1; k <= COL && flag == 0 ; k++){//从 check 数组中查找安全的数
字
            if(check[k] == CORRECT){
                flag = 1;
                a[i][j] = k;
                if(j == COL-1 && i != ROW-1){
                    if(Digit(a,i+1,0) == CORRECT) return CORRECT;
                    else flag = 0;
                }
                else if(j != COL-1){
                    if(Digit(a,i,j+1) == CORRECT) return CORRECT;
                    else flag = 0;
                }
            }
        }
        if( flag == 0 ) {
            a[i][j] = 0;
            return WRONG;
        }
    }
    return CORRECT;
}

void randomFirstRow(int a0[], int n) {//随机生成第一行
    int i,j;
    srand((unsigned)time(nullptr));
    for( i = 0 ; i < n ; i++){
        a0[i] = rand()%9 + 1;
        j = 0 ;
        while(j < i){
            if(a0[i] == a0[j]){
                a0[i] = rand()%9 + 1;
                j = 0;
            }
            else j++;
        }
    }
```

```
}

void createSudoku(int a[][COL]){ //生成数独
    randomFirstRow(a[0],COL);//随机生成第一行
    Digit(a,1,0);//递归生成后 i 行
}

void createStartinggrid(const int a[][COL], int b[][COL], int numDigits) {//随机生成初盘
    int i,j,k;
    srand((unsigned)time(nullptr));
    for( i = 0; i < ROW ; i ++)
        for( j = 0; j < COL ; j++)
            b[i][j] = a[i][j];

    //int c[numDigits][2];//此处可以采用 c++中的 new 动态为二维数组分配内存
    int** c = new int* [numDigits];
    for(int i = 0; i < numDigits; i++){
        c[i] = new int [2];
    }
    int m,flag = 0;

    int d[81]={0},d_count=0;
    for( i = 0; i < numDigits ; i++) {
        j = rand()%9;
        k = rand()%9;

        flag = 0;
        for(m = 0; m < i ; m++)
            if( j == c[m][0] && k == c[m][1])
                flag = 1;
        for(m=0;m<d_count;m++)
            if(j*9+k==d[m])
                flag=1;
        if(flag == 0){
            int tempp = b[j][k];
            int flag2=0;
            int book[1000]={0};
            book[tempp]=1;
```

```
                for(int num=1;num<=9;num++){
                    if(tempp==num) continue;
                    b[j][k]=num;
                    string testCnf = ToCnf(b,81-i-1);
                    HeadNode* L=CnfParser(testCnf);
                    if(NewDPLL(L,book)){
                        flag2=1;
                        break;
                    }
                }
                if(flag2==0){
                b[j][k] = 0;
                c[i][0] = j;
                c[i][1] = k;
                }
                else d[d_count++]=j*9+k;
            }
            else
                i--;
    }
    for(int i = 0; i < numDigits; i++){
    delete[] c[i];   // 先释放每行的内存
    }
    delete[] c;   // 再释放指针数组的内存
}

void print(const int a[][COL]){//打印数独数组
    int i,j;
    for( i = 0 ; i < ROW ; i++){
        for( j = 0 ; j < COL ; j++)
            printf("%d ", a[i][j]);
        cout<<endl;
    }
}

string ToCnf(int a[][COL],int holes) {
    ofstream in (".\\sudoku.cnf");//定义输入文件
    if(!in.is_open())
        cout<<"can't open!\n";
    in<<"p"<<" "<<"cnf"<<" "<<729<<" "<<8829+81-holes<<" "<<endl;
```

```
//single clause
for (int x = 0; x < ROW; ++x) {
    for (int y = 0; y < COL; ++y)
        if(a[x][y] != 0)
            in<<(x+1)*100 + (y+1)*10 + a[x][y]<<" "<<0<<endl;
}
//entry
for (int x = 1; x <= 9; ++x) {
    for (int y = 1; y <= 9; ++y) {
        for (int z = 1; z <= 9; ++z)
            in << x * 100 + y * 10 + z << " ";
        in<<0;
        in<<endl;
    }
}
//row
for (int y = 1; y <= 9; ++y) {
    for (int z = 1; z <= 9; ++z)
        for (int x = 1; x <= 8; ++x)
            for (int i = x+1; i <= 9; ++i)
                in<<0 - (x*100 + y*10 + z)<<" "
                  <<0 - (i*100 + y*10 + z)<<" "<<0<<endl;
}
//column
for (int x = 1; x <= 9; ++x) {
    for (int z = 1; z <=9 ; ++z)
        for (int y = 1; y <= 8; ++y)
            for (int i = y+1; i <= 9; ++i)
                in<<0-(x*100 + y*10 + z)<<" "
                  <<0-(x*100 + i*10 + z)<<" "<<0<<endl;
}
//3*3 sub-grids
for (int z = 1; z <= 9 ; ++z) {
    for (int i = 0; i <=2 ; ++i)
        for (int j = 0; j <=2 ; ++j)
            for (int x = 1; x <= 3 ; ++x)
                for (int y = 1; y <= 3; ++y)
                    for (int k = y+1; k <= 3; ++k)
                        in<<0 - ((3*i+x)*100 + (3*j+y)*10 + z)<<" "
                            <<0-((3*i+x)*100 + (3*j+k)*10 + z)<<"
```

```
"<<0<<endl;
    }
    for (int z = 1; z <= 9; z++) {
        for (int i = 0; i <= 2; i++)
            for (int j = 0; j <= 2; j++)
                for (int x = 1; x <= 3; x++)
                    for (int y = 1; y <= 3; y++)
                        for (int k = x + 1; k <= 3; k++)
                            for (int l = 1; l <= 3; l++)
                                in << 0 - ((3*i+x)*100 + (3*j+y)*10 + z) << ' '
                                    << 0 - ((3*i+k)*100 + (3*j+l)*10 + z) << ' '
<< 0 <<endl;
    }
    for(int z=1;z<=9;z++){
        for(int y=9;y>=1;y--)
            for(int i=y-1;i>=1;i--)
                in<<0-((10-y)*100+y*10+z)<<"  "<<0-((10-i)*100+i*10+z)<<"
"<<0<<endl;
    }//斜对角线

    for(int z=1;z<=9;z++){
        for(int x=2;x<=4;x++)
            for(int y=2;y<=4;y++)
                for(int i=2;i<=4;i++)
                    for(int j=2;j<=4;j++)
                        if(!(x==i&&y==j))
                                                in<<0-((x)*100+y*10+z)<<"   "<<0-
((i)*100+j*10+z)<<" "<<0<<endl;
    }//左上角 3*3 宫
        for(int z=1;z<=9;z++){
            for(int x=6;x<=8;x++)
                for(int y=6;y<=8;y++)
                    for(int i=6;i<=8;i++)
                        for(int j=6;j<=8;j++)
                            if(!(x==i&&y==j))
                                                in<<0-((x)*100+y*10+z)<<"   "<<0-
((i)*100+j*10+z)<<" "<<0<<endl;
    }//右下角 3*3 宫
    in.close();
    return ".\\sudoku.cnf";//返回一个 string 类型的对象
```

```
}
```