

**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： CS2402**

**学 号： U202414574 \_**

**姓 名： 邵书毅**

**指导教师： 李丹\_\_**

**报告日期： 2025.9.8**

**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-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to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%)

* **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**

**[任务书](#_Toc29011)** [1](#_Toc29011)

[设计内容 1](#_Toc17129)

[设计要求 1](#_Toc12499)

[ 参考文献 2](#_Toc27775)

**[引言](#_Toc9791)** [4](#_Toc9791)

[1.1课题背景与意义 4](#_Toc25511)

[1.2国内外研究现状 4](#_Toc25542)

[1.3课程设计的主要研究工作 5](#_Toc22803)

**[系统需求分析与总体设计](#_Toc31779)** [6](#_Toc31779)

[2.1系统需求分析 6](#_Toc7739)

[2.2系统总体设计 6](#_Toc6533)

**[系统详细设计](#_Toc14499)** [7](#_Toc14499)

[3.1有关数据结构的定义 7](#_Toc18248)

[3.2 主要算法设计 8](#_Toc14988)

**[系统实现与测试](#_Toc6552)** [11](#_Toc6552)

[4.1系统实现 11](#_Toc20124)

[4.2系统测试 13](#_Toc772)

**[总结与展望](#_Toc19877)** [19](#_Toc19877)

[5.1全文总结 19](#_Toc20394)

[5.1工作展望 20](#_Toc6030)

**[体会](#_Toc15094)** [20](#_Toc15094)

**[参考文献](#_Toc28855)** [22](#_Toc28855)

**[附录](#_Toc31737)** [23](#_Toc31737)

**1引言**

**1.1课题背景与意义**

对于计算机科学与技术、信息安全与物联网专业大二学生，在前两个学期已经学习了C语言程序设计、数据结构两门面向编程知识与技术的基础理论课，以及C语言程序设计实验、数据结构实验两门编程实践课程，不仅具有较为系统性的C语言、常用数据结构基本知识，而且具有初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。但两门实验课仍属于对基本编程模型与技术的验证性训练，而“程序设计”综合课程设计正是使大家从简单验证到综合应用，甚至在编程中实现智慧与风格升华的重要实践环节，为后续学习与进行计算机系统编程打下坚实的基础，让综合编程技能成为大家的固有能力与通向未来专业之门的钥匙。

SAT问题是计算机科学中的经典问题，其可满足性判定在形式验证、人工智能、自动推理等领域具有广泛应用。DPLL算法是解决SAT问题的经典算法之一，通过回溯和剪枝策略高效地搜索解空间。本设计通过实现DPLL算法，并结合数独游戏的应用，展示SAT问题在实际问题中的求解能力。

**1.2国内外研究现状**

SAT求解器的发展经历了从基础DPLL到现代CDCL（Conflict-Driven Clause Learning）算法的演进。国内外研究者提出了多种优化策略，如变元选择启发式、子句学习、非时序回溯等，显著提高了求解效率。  
 国内的SAT研究，特别是中科院软件所等机构，展现出强烈的应用驱动和技术突破色彩：

1.注重实际应用：研究常围绕电子设计自动化（EDA）、操作系统、云计算等国家战略需求和产业发展中的核心难题展开。

2.追求性能极致：不仅在传统算法上精益求精，也积极探索如FPGA硬件加速等新路径，追求算力的极致提升。

3.强调自主创新：致力于研发完全自主知识产权的求解器，并积极参与国际竞赛，提升影响力。

国际上的SAT研究则更显多样化和交叉性：

1. 探索新计算范式：研究量子计算等新型计算范式如何应用于SAT问题。
2. 与人工智能深度融合：探索如何用机器学习（ML） 技术来指导求解过程，例如用强化学习动态调整求解器参数、预测好的分支变量，或者用图神经网络学习公式的结构特征。
3. 应对更大规模问题：随着问题规模的不断增长，并行与分布式求解技术持续受到关注。
4. 理论研究持续深入：对SAT问题难易边界、结构复杂性（如社区结构、复杂网络特征）的理论研究仍在不断深入，以期从根本上指导算法设计。

2022年，华中科技大学计算机学院何琨教授团队提出将随机游走策略与决策树模型相结合的创新方法，使求解器在面对具有不同特征的算例时采用不同的随机游走策略辅助搜索，很好地提升了SAT求解器的鲁棒性。

数独作为NP完全问题，可通过归约为SAT问题进行求解，已有多个研究展示了其可行性与高效性。

**1.3课程设计的主要研究工作**

按工程化的基本流程分别完成如下任务：

⑴阅读“程序设计”综合课程设计任务书，熟悉问题，查阅文献，了解问题背景及相关知识。

⑵对设计问题进行需求分析，分析问题中所涉及的数据对象，划分功能，人机交互需求与数据文件读写等，并对问题进行形式化表示。

⑶基于上述需求分析，进行系统设计，明确程序的模块结构；设计数据结构（逻辑结构及其物理结构），参考并设计主要子问题的求解算法。

⑷程序实现，基于系统设计，制定相应的实现方案，编写各程序模块，完成程序编写与调试任务。

⑸程序测试，设计测试用例对程序进行功能测试，性能测量及理论分析。

⑹程序优化，对设计方案中的结构，算法进行一定优化，测试与分析性能改善结果。在设计报告中明确说明你的优化策略与方案。

⑺设计总结，按规范化要求撰写“程序设计”综合课程设计报告。

⑻成果提交：将程序源代码/工程文件、可独立运行的可执行程序、简要操作手册及“程序设计”综合课程设计报告电子版打包。

**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，基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。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。
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。
3. DPLL过程：基于DPLL算法框架，实现SAT算例的求解（数据结构不要使用C++现有的vector等类库）。
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. 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，提供明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。
6. SAT应用：将百分号数独游戏(%-Sudoku)[12-13]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/简单交互性。

**2.2系统总体设计**

设计程序模块化，程序源代码进行模块化组织。主要模块包括如下：

1）主控、交互与显示模块（display）；

2）CNF解析模块（cnfparser）；

3）核心DPLL模块( solver)；

4）百分号数独模块，包括游戏格局生成、归约、求解(X-Sudoku)。

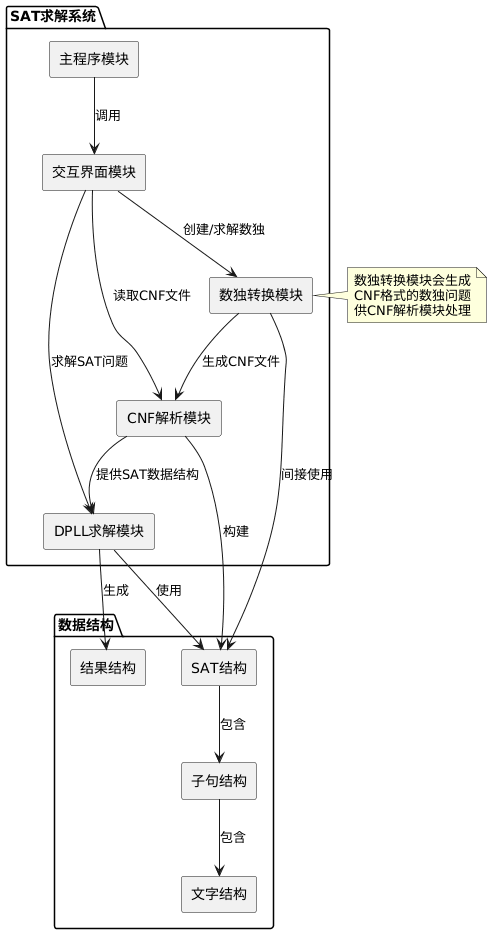


图2.2系统模块图

**3系统详细设计**

**3.1有关数据结构的定义**

系统中使用的主要数据结构如下：

| 数据类型 | 描述 | 成员说明 |
| --- | --- | --- |
| L | 文字节点 | int x：变元值（正负表示文字）；L\* next：下一个文字指针。 |
| Clause | 子句结构 | L Lhead：文字头节点；Clause\* next：下一个子句指针。 |
| SAT | SAT实例结构 | char name[20]：文件名；int m, n：变元数与子句数；Clause Chead：子句头节点。 |
| Result | 求解结果 | int time：用时；int ans：是否可满足；int\* bl：变元赋值数组。 |

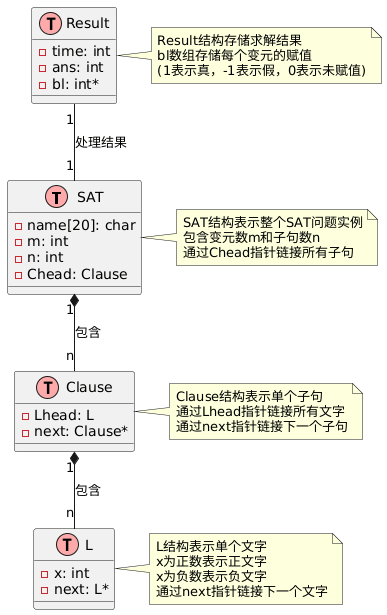


图3.1.2 数据在系统间的关联

**3.2 主要算法设计**

核心函数 DPLL(Clause, bl, M, choose):{

若子句链表为空，返回满足

若存在空子句，返回不满足

单子句传播：处理所有单子句，更新赋值

选择变元 l（根据choose策略）

递归尝试 l 为真：

若满足，返回真

递归尝试 l 为假：

若满足，返回真

返回不满足

}

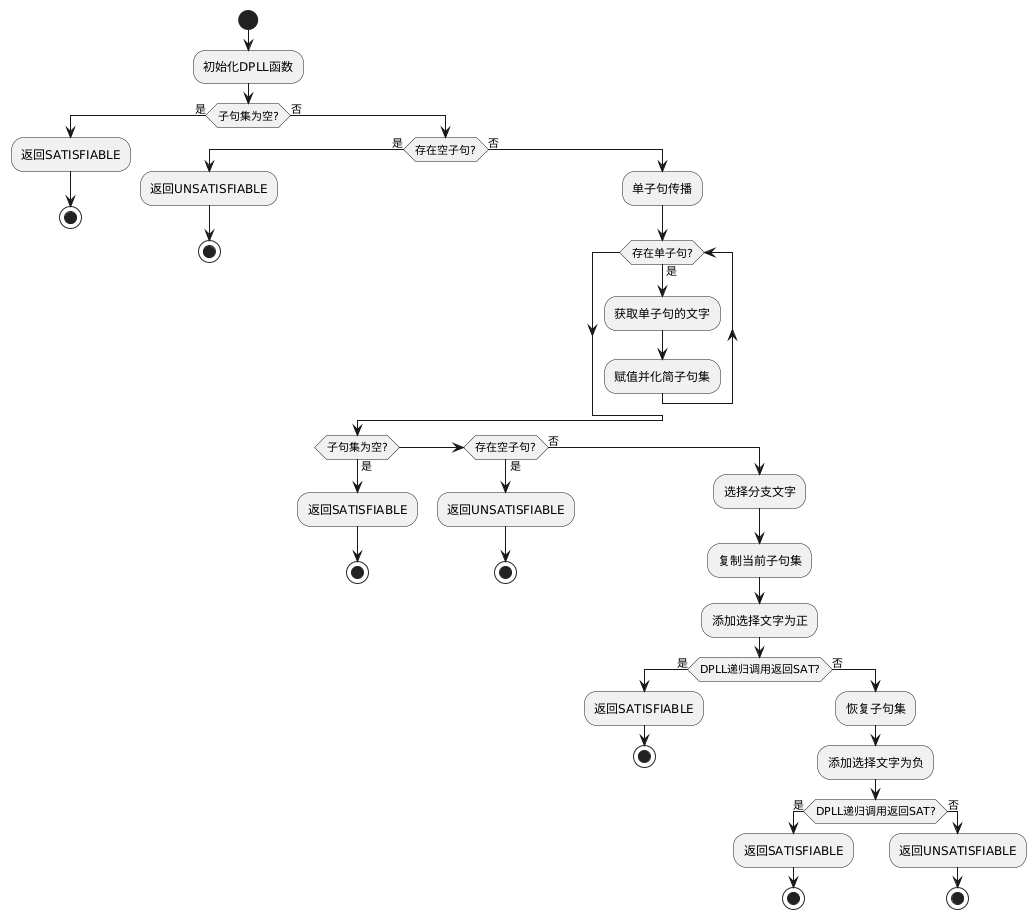


图3.2.1 dpll递归求解函数

1. CNF解析模块(`cnfparser.cpp`)：

功能：负责读取和解析标准`CNF`格式文件。

跳过文件注释行（以`c`开头的行）。

解析文件头，获取变元总数和子句总数。

构建公式的内部表示：使用链表结构（`Clause`链表包含`L`文字链表）存储所有子句。

提供将内部表示重新输出为CNF文件的功能（用于验证解析正确性）。

2. DPLL求解模块(`solver.cpp`)：

功能：实现SAT问题的核心求解算法。

基础DPLL框架：递归地进行布尔约束传播（BCP）、决策、回溯。

单子句传播：识别并处理单子句，提前赋值以简化问题。

分支策略：提供两种变元选择策略：

策略0：选择第一个子句的第一个文字。

策略1：选择出现频率最高的文字（优化策略）。

化简与回溯：根据赋值简化子句，处理冲突并回溯。

结果保存：将求解结果（满足性、赋值、时间）输出到`.res`文件。

1. 数独转换模块(`X-Sudoku.cpp`)：

功能：将数独问题（包括百分号等附加规则）转化为CNF公式。

生成数独模板：创建或读取数独谜题文件。

规则编码：将数独规则编码为CNF子句，包括：

每个格子至少一个数字。

每个格子至多一个数字。

行、列、九宫格内数字不重复。

附加规则：对角线、百分号区域内的数字不重复。

已知条件编码：将预填数字作为单子句加入CNF。

输出CNF文件：生成可供DPLL求解器读取的CNF文件。

4. 交互界面模块(`display.cpp`)：

功能：提供用户友好的命令行交互界面。

主菜单驱动：显示功能选项（读取CNF、求解SAT、求解数独、退出）。

流程控制：根据用户选择调用相应模块（如`Readcnf`, `Solve`, `SolveSudoku`）。

参数输入：接收用户指定的文件路径、求解策略等参数。

结果展示：显示求解状态（成功/失败）、数独解答、耗时等信息。

资源管理：协调各模块调用，并在退出时自动清理内存。

这四大模块协同工作，实现了从：问题输入（CNF/数独）→ 转换/解析 → 求解 → 结果输出/展示 的完整流程。

**4系统实现与测试**

**4.1系统实现**

开发环境：

操作系统：Windows 11

编译器：GCC 9.3.0

开发语言：C语言

宏定义（函数运行结果状态量）：

#define OK 1;

#define NO 0;

数据类型定义：

//变元结构

typedef struct L{

    int x;//变元

    L\* next;

}L;

//子句结构

typedef struct Clause{

    L Lhead;

    Clause\* next;

}Clause;

//定义读入的SAT结构

typedef struct SAT{

    char name[20];

    int m;//变元数

    int n;//子句数

    Clause Chead;//数据

}SAT;

//解答返回值

typedef struct Result{

    int time;

    int ans;

    int\* bl;

}Result;

以下是代码中所有函数，按模块分类分别简述：

CNF解析模块 (cnfparser.cpp)

SAT\* Readcnf(char\* file)

读取CNF文件并构建SAT数据结构，返回该SAT

int Reoutput(SAT \*sat)

将SAT结构重新输出为CNF格式文件，返回函数运行结果状态量

DPLL求解模块 (solver.cpp)

Clause Copy(Clause CL)

深拷贝子句链表，返回新链表表头

int ClearL(Clause\* head)

清理子句中所有的文字链表，返回函数运行结果状态量

int Clear(Clause &head)

清理整个子句链表，返回函数结果状态量

int Change(Clause &CL, int l)

根据赋值l化简子句集，返回函数运行结果状态量

int EmptyClause(Clause CL)

检查是否存在空子句，返回值有空1/无空0

int FindOne(Clause CL)

寻找单子句，返回该变元编号

int AddClause(Clause &CP, int l)

添加只包含文字l的子句，返回函数运行结果状态量

int Choose1(Clause CL, int M)

变元选择优化策略，选择出现频率最多的字，返回字

int DPLL(Clause &CL, int\* bl, int M, int choose)

DPLL算法核心实现，返回函数运行结果状态量

int SaveResult(SAT\* sat, Result\* result, int choose)

保存求解结果到文件，返回函数运行结果状态量

Result\* Solve(SAT\* sat, int choose)

求解SAT问题的主入口函数，返回求解的结果

数独转换模块 (X-Sudoku.cpp)

char\* CreateSudoku(int choose)

创建数独谜题文件，返回创建的数独文件名

char\* Transformcnf(char\* file)

将数独问题转换为CNF格式，返回转换成的cnf文件名

int SolveSudoku(int CoR)

解决数独问题的主函数，返回函数运行结果状态量

交互界面模块 (display.cpp)

int Display()

显示主交互界面和处理用户输入，返回函数运行结果状态量

CNF解析文件（\*.res）格式：

**s**求解结果//1表示满足，0表示不满足，-1表示在限定时间内未完成求解

**v** -1 2 -3 … //满足时，每个变元的赋值序列，-1表示第一个变元1取假，2表示第二个变元取真，用空格分开，此处为示例。

**t** //以毫秒为单位的DPLL执行时间，可增加分支规则执行次数信息

数独文件格式（Sudoku.txt）：

.......9....23.6...3.7....5.....4........69........8...82..9....9.....2..1.......

一行代表一个数独棋盘的数据，共81个字符，按序对应数独棋盘上81个格子，“.”代表空格子，数字代表该格子已经填入的数字。

**4.2系统测试**

**4.2.1 常用的软件测试方法：**

单元测试：针对各个独立模块进行测试，验证每个函数的功能正确性

集成测试：测试模块之间的接口和协作关系

功能测试：验证系统是否满足需求规格说明中规定的功能要求

边界值测试：测试输入参数的边界条件

性能测试：测量系统在不同负载下的性能表现

回归测试：确保修改后的代码不会破坏现有功能

**4.2.2 主要功能模块测试**

模块一：CNF解析模块（Readcnf）

模块功能与设计目标：

功能：读取CNF格式文件，解析并构建SAT数据结构  
设计目标：

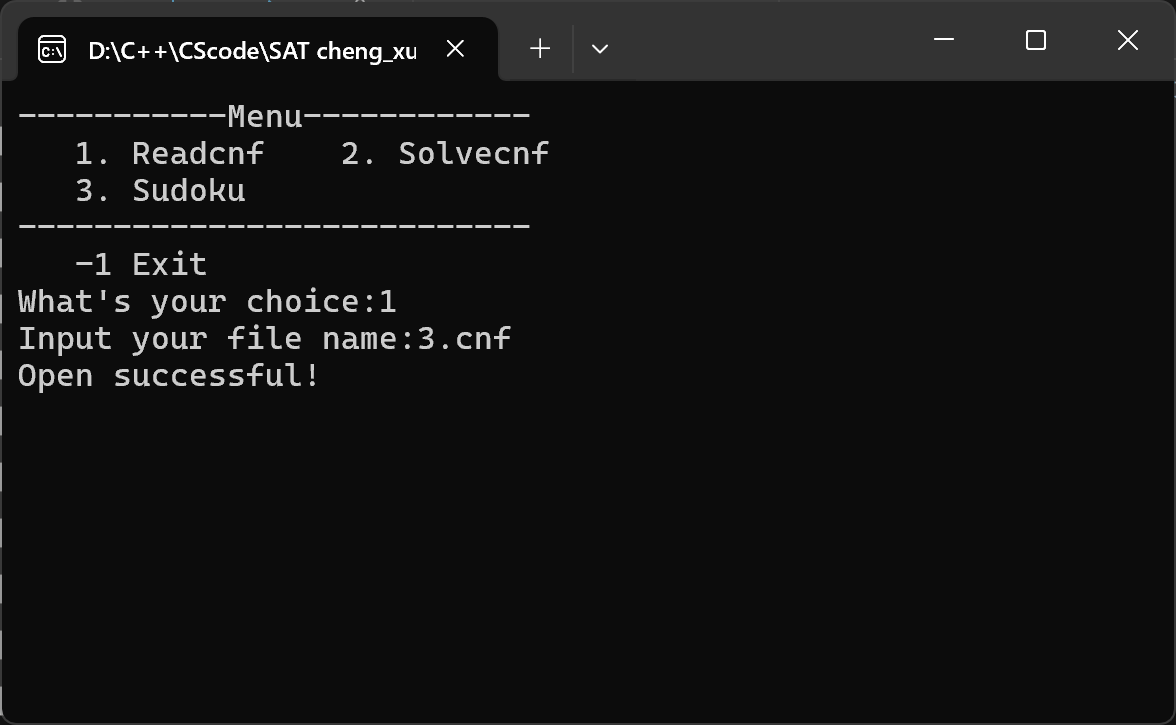
正确解析CNF文件头信息（变元数、子句数）

准确构建子句和文字的链表结构

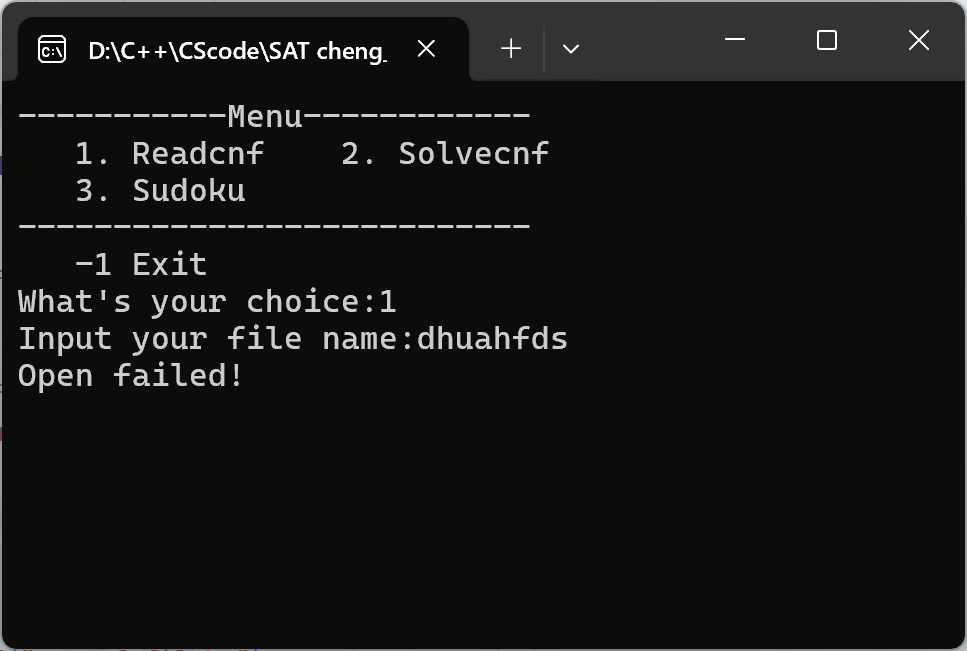
处理注释行和文件格式异常

支持标准CNF格式

运行与算例测试：



读取失败：



模块二：Solve求解模块

功能：实现DPLL算法求解SAT问题  
设计目标：

正确实现DPLL算法框架

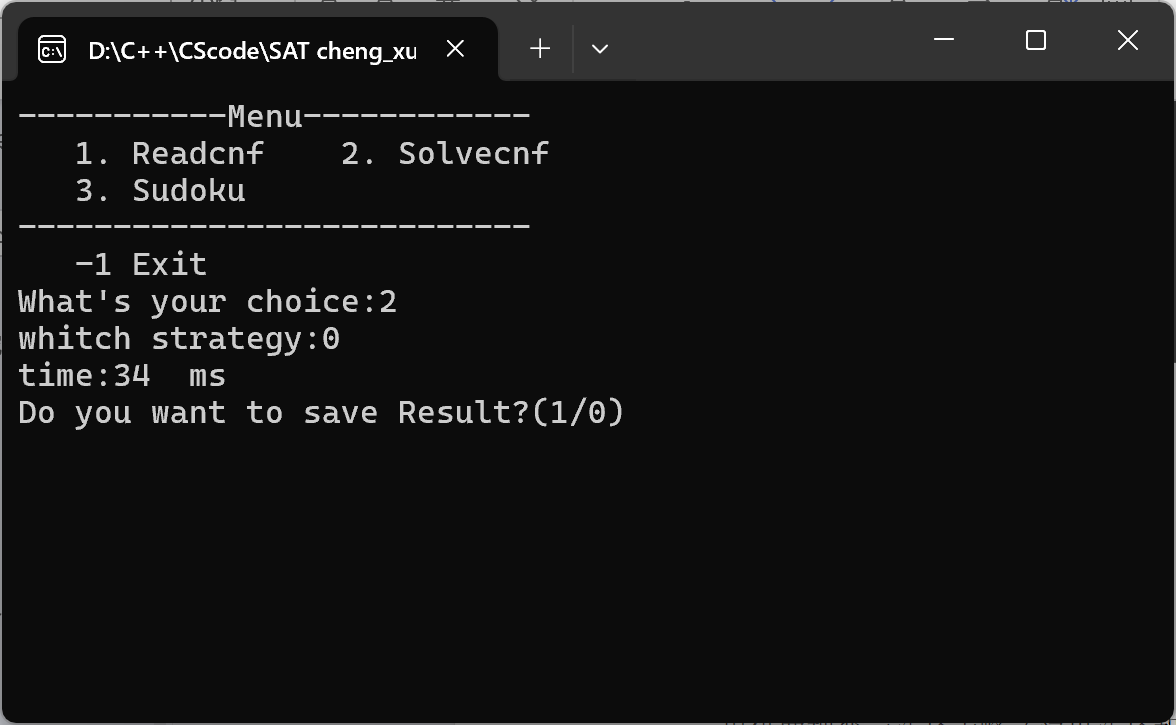
支持单子句传播和回溯机制

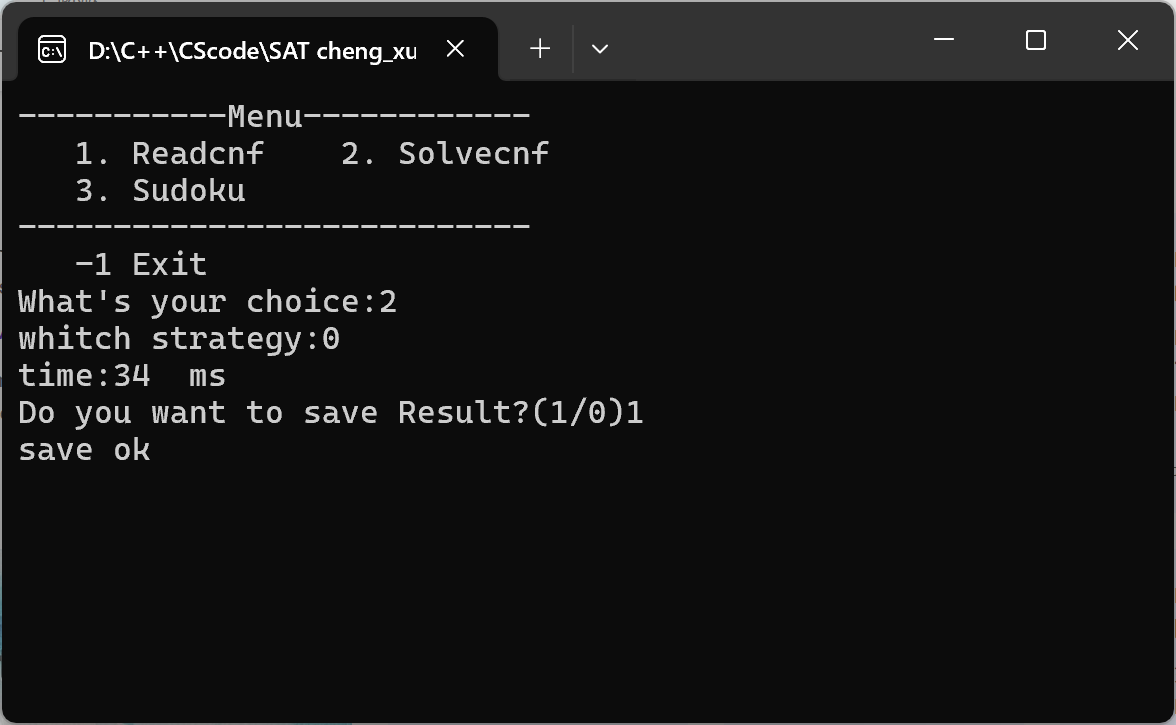
提供两种变元选择策略（简单选择和启发式选择）

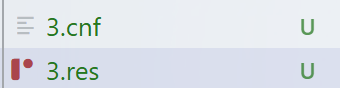
正确处理可满足和不可满足的SAT实例

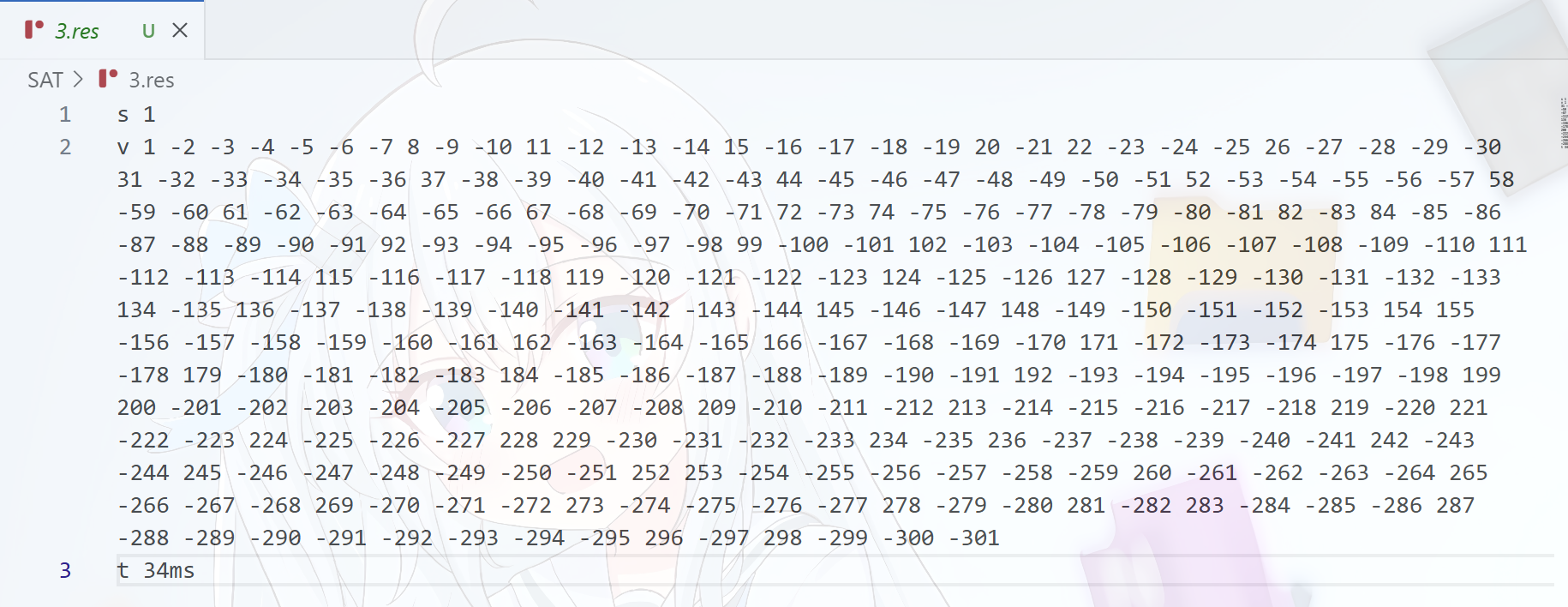
记录求解时间性能

运行与算例测试：

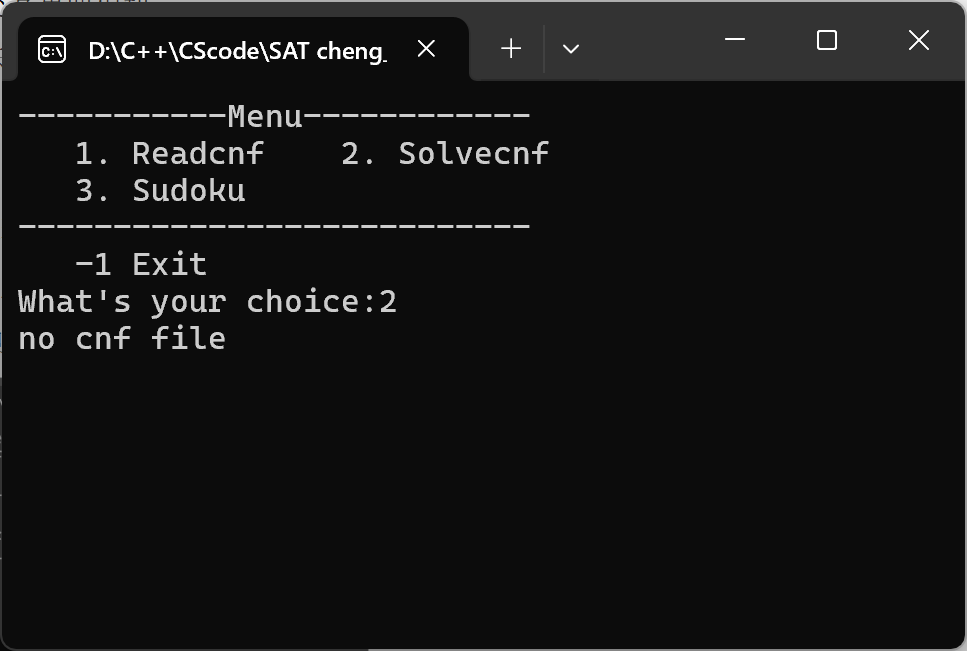




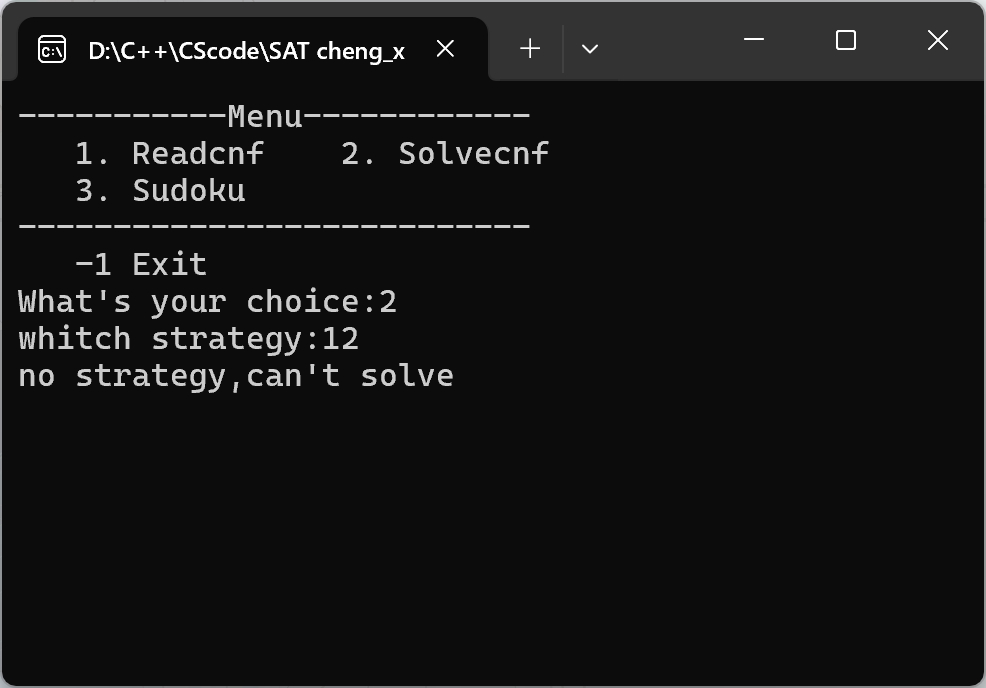




无读取文件时：



错误策略输入时：



模块三：数独转换模块

功能：将数独问题转换为CNF格式  
设计目标：

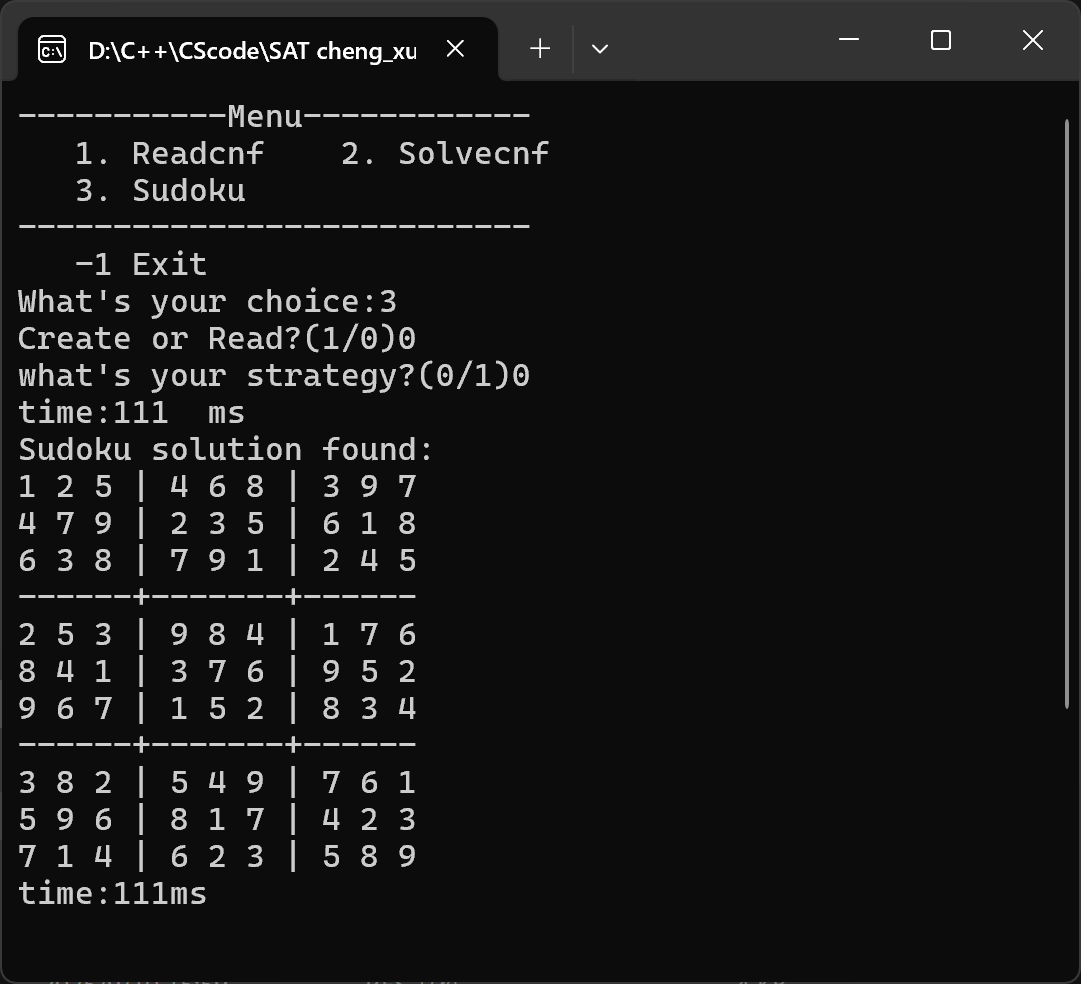
正确编码数独基本规则（每个格子、每行、每列、每宫的数字约束）

支持百分号数独的特殊约束（对角线、百分号区域）

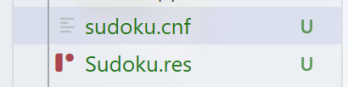
正确处理已知数字作为单子句加入

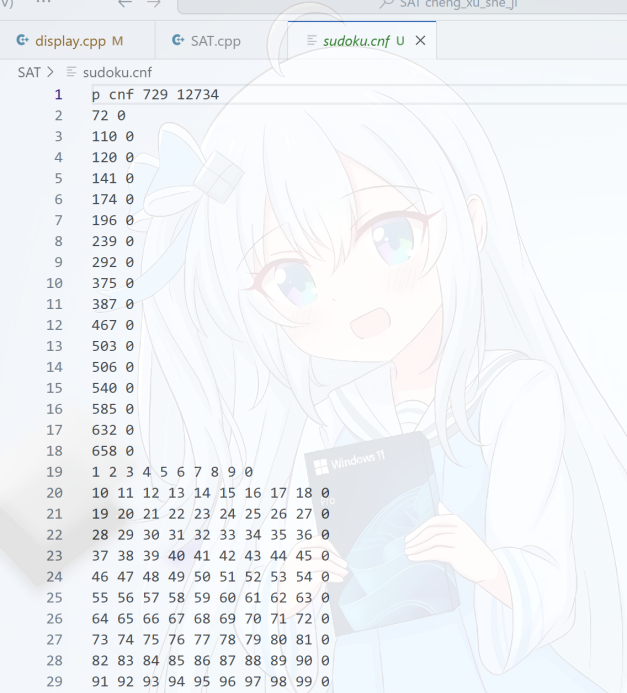
生成符合DIMACS标准的CNF文件

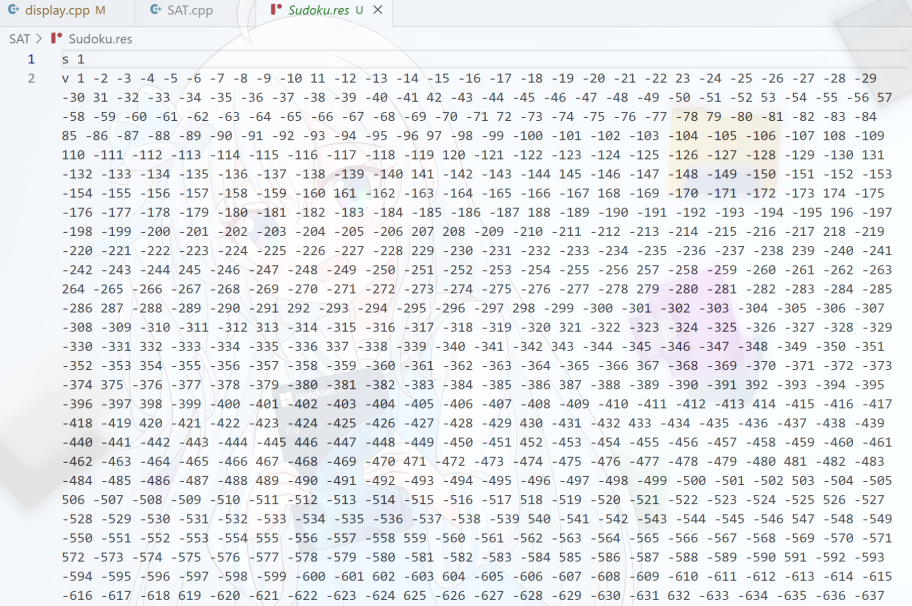
运行与算例测试：



以下是数独程序生成文件：







模块四：交互界面模块

功能：显示并提供用户交互与选择

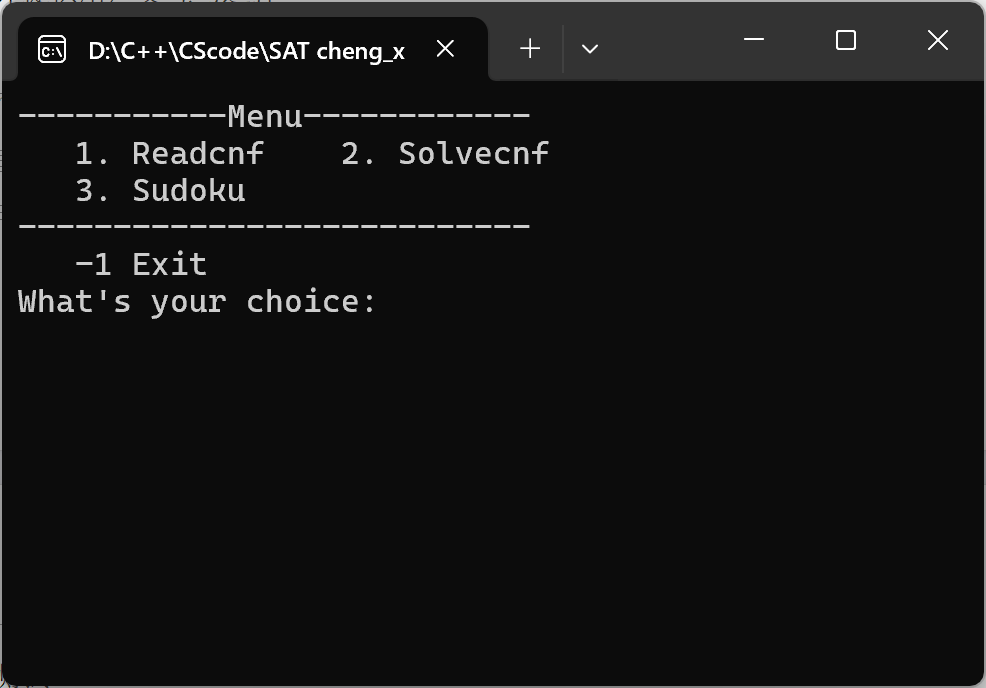
设计目标：

简洁明了易于理解

正确处理用户输入

正确调用相关功能模块

运行与算例测试：



**4.4.3 算例测试结果**



**5总结与展望**

**5.1全文总结**

本课程设计基于DPLL算法实现了一个完整的SAT求解器，并将其应用于百分号数独游戏的求解中。系统具备CNF文件解析、SAT求解、数独问题归约、用户交互等核心功能，并在算法优化方面进行了初步探索。

在系统实现过程中，我设计了合理的数据结构（如子句链表、文字链表）来存储SAT实例，实现了基础的DPLL算法框架，并在此基础上引入了变元选择启发式策略（如出现频率最高的文字优先选择），显著提升了求解效率。系统还支持将数独问题（百分号数独的特殊约束）转化为CNF格式，并通过SAT求解器进行求解，最终将结果可视化输出。

通过功能测试与性能分析，系统在中小规模SAT算例和数独谜题上表现良好，验证了算法正确性与实用性。

本设计不仅加深了对SAT问题、DPLL算法及其应用的理解，也锻炼了系统设计、代码实现与工程调试的能力。

**5.1工作展望**

尽管本系统基本实现了设计要求，但仍存在进一步优化和扩展的空间。

未来的工作可以从以下几个方面展开：

算法优化：引入冲突驱动子句学习（CDCL）、非时序回溯等现代SAT求解技术，进一步提升求解效率；尝试多种变元选择启发式策略，并进行动态策略切换。

并行化处理：探索多线程或分布式计算框架下的DPLL算法实现，以应对更大规模的SAT实例。

功能扩展：支持更多类型的逻辑谜题（如N皇后、图着色等）归约为SAT问题，增强系统的通用性与实用性。

用户交互与可视化：开发图形用户界面（GUI），支持数独谜题的可视化输入与解答展示，提升用户体验。

性能分析与调优：使用更丰富的测试集进行系统性性能评估，识别瓶颈并进行针对性优化。

# 6 体会

这次“程序设计综合课程设计”无疑是我大学学习生涯中遭遇的最具挑战性的编程项目之一。从最初仔细研读任务书、广泛查阅相关文献，到最终成功完成一个功能完备的 SAT 求解器，并将其巧妙地应用于数独游戏的求解，整个过程不仅极大地锻炼了我的编程技能和系统设计能力，更让我深刻体会到了理论知识在实际应用中的巨大价值。

在技术层面，我深刻认识到了数据结构与算法设计对于程序性能的重要性。在项目初期，我投入了大量时间精心设计链表结构，逐一攻克了深拷贝、内存管理以及子句化简与回溯等难题。这一过程中，我不仅熟练掌握了 C 语言中的指针和内存操作技巧，更深刻理解了选择合适的数据结构对于程序效率和稳定性的关键作用。DPLL 算法递归实现的技巧性，更是让我将书本上的递归和回溯概念转化为了实际可运行的代码。

调试环节无疑是整个项目中最耗时却也收获最大的部分。在实现 DPLL 算法的过程中，一个小小的错误就可能导致整个程序的崩溃。我学会了通过分模块测试等方法迅速定位问题所在，还通过输出 CNF 文件来验证解析的正确性。这种系统化的方法论收获，对我来说比单纯的代码编写本身更有价值。

这次设计经历也让我对“优化”有了全新的认识。最初的基础 DPLL 算法在处理大规模算例时运行缓慢，通过参考大量文献，我成功实现了启发式选择策略，优化后的程序性能得到了大幅提升。这让我深刻明白，一个优秀的程序不仅应该功能完备，更应该高效且健壮。

将 SAT 问题与数独游戏相结合，更是让我深入理解了“归约”这一重要的问题求解方法。我将数独规则巧妙地转化为 CNF 子句，利用求解器得出了正确的答案，这充分展示了理论知识如何指导实践，解决现实中的复杂问题。

总的来说，这次课程设计是一次全方位的能力锻炼，不仅提升了我的编程和系统设计能力，更让我看到了自己的不足，激发了我对编程世界的无限探索兴趣。

**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**

“cnfparser.pp”

SAT\* Readcnf(char\* file){

    FILE \*fp=fopen(file,"r");

    if(fp==NULL){

        return NULL;

    }

    SAT \*sat=(SAT\*)malloc(sizeof(SAT));

    strcpy(sat->name,file);

    //跳过注释

    char x=fgetc(fp);

    while(x=='c'){

        while(fgetc(fp)!='\n');

        x=fgetc(fp);

    }

    //跳过第一行,并读出数据

    char line[100];fscanf(fp,"%s",line);

    fscanf(fp,"%d %d",&sat->m,&sat->n);

    //读入数据

    sat->Chead.next=NULL;

    Clause \*chead=&sat->Chead;

    for(int i=0;i<sat->n;i++){

        chead->next=(Clause\*)malloc(sizeof(Clause));

        chead=chead->next;

        chead->next=NULL;

        L\* tail=&chead->Lhead;

        int num=0;

        while(1){

            fscanf(fp,"%d",&num);

            if(num==0) break;

            tail->next=(L\*)malloc(sizeof(L));

            tail=tail->next;

            tail->next=NULL;

            tail->x=num;

        }

    }

    fclose(fp);

    return sat;

}

“solver.cpp”

//复制，用深拷贝

Clause Copy(Clause CL){

    Clause cp\_clause=CL;

    //复制句子

    Clause\* ctail=&cp\_clause, \*chead=CL.next;

    while(chead!=NULL){

        ctail->next=(Clause\*)malloc(sizeof(Clause));

        ctail=ctail->next;

        //复制句头

        ctail->next=NULL;

        //复制字

        L \*ltail=&ctail->Lhead,\*lhead=chead->Lhead.next;

        while(lhead!=NULL){

            ltail->next=(L\*)malloc(sizeof(L));

            ltail=ltail->next;

            ltail->x=lhead->x;

            ltail->next=NULL;

            lhead=lhead->next;

        }

        chead=chead->next;

    }

    return cp\_clause;

}

//清理函数

//清理字

int ClearL(Clause\* head){

    L\* ltemp;

    while(head->Lhead.next!=NULL){

        ltemp=head->Lhead.next;

        head->Lhead.next=head->Lhead.next->next;

        free(ltemp);

    }

    return OK;

}

//清理句子链表

int Clear(Clause &head){

    if(head.next==NULL) return OK;

    Clause \*ctemp;

    while(head.next!=NULL){

        ClearL(head.next);

        ctemp=head.next;

        head.next=head.next->next;

        free(ctemp);

    }

    return OK;

}

//化简,若出现真则删句子;若出现假字,则删字

int Change(Clause &CL,int l){

    if(CL.next==NULL) return OK;

    Clause\* Cpre=&CL,\*Cp=CL.next;

    while(Cp!=NULL){

        L\* Lpre=&Cp->Lhead,\*Lp=Cp->Lhead.next;

        int ifdelete=0;

        while(Lp!=NULL){

            if(Lp->x==l){

                ifdelete=1;

                break;

            }else if(Lp->x==-l){

                Lpre->next=Lp->next;

                free(Lp);

            }else{

                Lpre=Lpre->next;

            }

            Lp=Lpre->next;

        }

        if(ifdelete){

            ClearL(Cp);

            Cpre->next=Cp->next;

            free(Cp);

        }else{

            Cpre=Cpre->next;

        }

        Cp=Cpre->next;

    }

    return OK;

}

//但是注意以下的特别的情况:

//若字都删完了,句子依然在,就成了空句子,永远不能满足,所以该方案必然失败

int EmptyClause(Clause CL){

    Clause\* Cp=CL.next;

    while(Cp!=NULL){

        if(Cp->Lhead.next==NULL) return OK;

        Cp=Cp->next;

    }

    return NO;

}

//寻找单子句优先处理

int FindOne(Clause CL){

    Clause\* Cp=CL.next;

    while(Cp!=NULL){

        if(Cp->Lhead.next->next==NULL) return Cp->Lhead.next->x;

        Cp=Cp->next;

    }

    return NO;

}

//加入结点

int AddClause(Clause &CP,int l){

    Clause\* temp=(Clause\*)malloc(sizeof(Clause));

    temp->Lhead.next=(L\*)malloc(sizeof(L));

    temp->Lhead.next->x=l;

    temp->Lhead.next->next=NULL;

    temp->next=CP.next;

    CP.next=temp;

    return OK;

}

//优化选择

// int Choose0(Clause CL){

//     return CL.next->Lhead.next->x;

// }

int Choose1(Clause CL,int M){

    int\* cnt = (int\*)calloc(2\*M + 10, sizeof(int));

    Clause\* chead=CL.next;

    L\* lhead=NULL;

    while(chead!=NULL){

        lhead=chead->Lhead.next;

        while(lhead!=NULL){

            cnt[lhead->x+M]++;

            lhead=lhead->next;

        }

        chead=chead->next;

    }

    int max=0,max\_i=M;

    for(int i=0;i<=2\*M;i++){

        if(max<cnt[i]){

            max=cnt[i];

            max\_i=i;

        }

    }

    free(cnt);

    return max\_i-M;

}

//递归函数

//隐式回溯,因为赋值只由选择的那一个字决定后续的所有赋值,所以用全局变量各条分支也不会互相影响

int DPLL(Clause &CL,int\* bl,int M,int choose){

    //实时显示运行时间

    // clear\_line();

    // printf("%d",clock()-T);

    if(CL.next==NULL) return OK;

    Clause cp\_CL;

    int chs=0;

    while((chs=FindOne(CL))!=0){

        bl[abs(chs)] = (chs > 0 )? 1 : -1;

        Change(CL,chs);

        if(CL.next==NULL){

            Clear(cp\_CL);

            return OK;

        }else if(EmptyClause(CL)==1){

            Clear(cp\_CL);

            return NO;

        }

    }

    if(choose==0) chs=CL.next->Lhead.next->x;

    else if(choose==1) chs=Choose1(CL,M);

    //假设字chs为真

    cp\_CL=Copy(CL);

    AddClause(cp\_CL,chs);

    if(DPLL(cp\_CL,bl,M,choose)==1){

        Clear(cp\_CL);

        return OK;

    }

    Clear(cp\_CL);

    cp\_CL=Copy(CL);

    //回溯,尝试假设chs为假

    AddClause(cp\_CL,-chs);

    if(DPLL(cp\_CL,bl,M,choose)==1){

        Clear(cp\_CL);

        return OK;

    }

    Clear(cp\_CL);

    return NO;

}

int SaveResult(SAT\* sat,Result\* result,int choose){

    //处理输出文件名称

    char res\_file[20];

    strcpy(res\_file,sat->name);

    int t=0;

    while(res\_file[t++]!='.');

    //标记优化的结果

    if(choose!=0){

        t--;

        res\_file[t++]='\_',res\_file[t++]='0'+choose,res\_file[t++]='.';

    }

    res\_file[t++]='r',res\_file[t++]='e',res\_file[t++]='s',res\_file[t]='\0';

    //写入答案

    FILE\* fp=fopen(res\_file,"w");

    if(fp==NULL) return 0;

    if(result->ans==1){

        fprintf(fp,"s 1\nv ");

        for(int i=1;i<=sat->m;i++){

            fprintf(fp,"%d ",i\*result->bl[i]);

        }

    }else{

        fprintf(fp,"s -1");

    }

    fprintf(fp,"\nt %dms",result->time);

    fclose(fp);

    return 1;

}

//调用主函数

Result\* Solve(SAT\* sat,int choose){

    if(choose<0||choose>1) return NULL;

    Result\* result=(Result\*)malloc(sizeof(Result));

    result->bl=(int\*)malloc(sizeof(int)\*(sat->m+10));

    for(int i=0;i<sat->m+10;i++){

        result->bl[i]=0;

    }

    result->time=0;//记录用时

    int start=T=clock();

    printf("        ms");

    result->ans=DPLL(sat->Chead,result->bl,sat->m,choose);

    int end=clock();

    result->time=end-start;

    printf("\rtime:%d  ms\n",result->time);//为了与clearline函数一致

    return result;

}

“X-Sudoku.cpp”

char\* Transformcnf(char\* file){

    static char cnf\_file[20] = "Sudoku.cnf";

    FILE\* sudoku\_fp = fopen(file, "r");

    if(sudoku\_fp == NULL) return NULL;

    FILE\* cnf\_fp = fopen(cnf\_file, "w");

    if(cnf\_fp == NULL) {

        fclose(sudoku\_fp);

        return NULL;

    }

/\*

变元按序编码:(i-1)\*81+(j-1)\*9+k

数独有9x9=81个格子，每个格子有9种可能，共729个变量

每个格子都会产生9个变元,所以对于某一个,前面有(i-1)\*81+(j-1)\*9个,再加上其在当前格子9个变元里的序号,就是其编号

\*/

    //读取数独文件,记录已知位置

    int known[81]={0};

    int known\_cnt = 0;

    for(int i=0; i<9; i++){

        for(int j=0; j<9; j++){

            char sudoku\_char=fgetc(sudoku\_fp);

            if(sudoku\_char!='.'){

                known[known\_cnt++]=i\*81+j\*9+sudoku\_char-'0';

            }

        }

    }

    fclose(sudoku\_fp);

    int clauses = 0;

//通过基本的约束条件(由于数独的特殊性,约束条件大部分都是二者关系,即两两约束),排列组合计算子句数量

    //1.每个格子数字只能填1~9,即9个变元必有一真:81\*1=81条

    clauses += 81;

    //2.每个格子最多有一个数字,即9个变元间两两互斥:81\*C(9,2)=81\*36=2916条

    clauses += 2916;

//以下情况本质类似:格子两两组合,每个组合里都有9个相同变元不同时出现的限制,所以是9\*C(9,2)

    //3.每行每个数字只能出现一次:9\*9\*C(9,2)=81\*36=2916条

    clauses += 2916;

    //4.每列每个数字只能出现一次:同 2916条

    clauses += 2916;

    //5.每个3x3宫每个数字只能出现一次:9\*9\*C(9,2)=81\*36=2916条

    clauses += 2916;

    //6.一条对角线每个数字只能出现一次:1\*9\*C(9,2)=324条

    clauses += 324;

    //7.百分号两个圆构成的九宫格:2\*9\*C(9,2)=648条

    clauses+=648;

    //特殊情况约束条件:数独中已填数字的数量

    clauses += known\_cnt;

    //写入CNF文件头

    fprintf(cnf\_fp, "p cnf 729 %d\n", clauses);

    //先写已知条件,便于dpll求解

    for(int t=0;t<known\_cnt;t++){

        fprintf(cnf\_fp,"%d 0\n",known[t]);

    }

    //1.每个格子至少有一个数字

    for(int i=1; i<=9; i++){

        for(int j=1; j<=9; j++){

            for(int k=1; k<=9; k++){

                fprintf(cnf\_fp, "%d ", (i-1)\*81 + (j-1)\*9 + k);

            }

            fprintf(cnf\_fp, "0\n");

        }

    }

/\*

将二者互斥关系转换成真假满足条件

\*/

    //2.每个格子最多有一个数字

    for(int i=1; i<=9; i++){

        for(int j=1; j<=9; j++){

            for(int k=1; k<=8; k++){

                for(int l=k+1; l<=9; l++){

                    fprintf(cnf\_fp, "-%d -%d 0\n", (i-1)\*81 + (j-1)\*9 + k,(i-1)\*81 + (j-1)\*9 + l);

                }

            }

        }

    }

    //3.每行每个数字只能出现一次

    for(int i=1; i<=9; i++){

        for(int k=1; k<=9; k++){

            for(int j=1; j<=8; j++){

                for(int l=j+1; l<=9; l++){

                    fprintf(cnf\_fp, "-%d -%d 0\n",

                           (i-1)\*81 + (j-1)\*9 + k,

                           (i-1)\*81 + (l-1)\*9 + k);

                }

            }

        }

    }

    //4.每列每个数字只能出现一次

    for(int j=1; j<=9; j++){

        for(int k=1; k<=9; k++){

            for(int i=1; i<=8; i++){

                for(int l=i+1; l<=9; l++){

                    fprintf(cnf\_fp, "-%d -%d 0\n",

                           (i-1)\*81 + (j-1)\*9 + k,

                           (l-1)\*81 + (j-1)\*9 + k);

                }

            }

        }

    }

    //5.每个3x3九宫格每个数字只能出现一次

    for(int boxRow=0; boxRow<3; boxRow++){

        for(int boxCol=0; boxCol<3; boxCol++){

            for(int k=1; k<=9; k++){

                for(int i=1; i<=3; i++){

                    for(int j=1; j<=3; j++){

                        int cell1\_i = boxRow\*3 + i;

                        int cell1\_j = boxCol\*3 + j;

                        for(int m=1; m<=3; m++){

                            for(int n=1; n<=3; n++){

                                int cell2\_i = boxRow\*3 + m;

                                int cell2\_j = boxCol\*3 + n;

                                if(cell1\_i != cell2\_i || cell1\_j != cell2\_j){

                                    fprintf(cnf\_fp, "-%d -%d 0\n",

                                           (cell1\_i-1)\*81 + (cell1\_j-1)\*9 + k,

                                           (cell2\_i-1)\*81 + (cell2\_j-1)\*9 + k);

                                }

                            }

                        }

                    }

                }

            }

        }

    }

    //6.对角线 (i + j = 10)

    for(int k=1; k<=9; k++){

        for(int i=1; i<=8; i++){

            int j1 = 10 - i;

            for(int m=i+1; m<=9; m++){

                int j2 = 10 - m;

                fprintf(cnf\_fp, "-%d -%d 0\n",

                    (i-1)\*81 + (j1-1)\*9 + k,

                    (m-1)\*81 + (j2-1)\*9 + k);

            }

        }

    }

    //7.百分号的两个圆

    for (int k = 1; k <= 9; k++){

        for (int i = 1; i <= 3; i++){

            for (int j = 1; j <= 3; j++){

                int cell1\_i = 1 + i;

                int cell1\_j = 1 + j;

                for (int m = 1; m <= 3; m++){

                    for (int n = 1; n <= 3; n++){

                        int cell2\_i = 1 + m;

                        int cell2\_j = 1 + n;

                        if (cell1\_i != cell2\_i || cell1\_j != cell2\_j){

                            fprintf(cnf\_fp, "-%d -%d 0\n",

                                (cell1\_i-1)\*81+(cell1\_j-1)\*9+k,

                                (cell2\_i-1)\*81+(cell2\_j-1)\*9+k);

                        }

                    }

                }

            }

        }

    }

    for (int k = 1; k <= 9; k++){

        for (int i = 1; i <= 3; i++){

            for (int j = 1; j <= 3; j++){

                int cell1\_i = 5 + i;

                int cell1\_j = 5 + j;

                for (int m = 1; m <= 3; m++){

                    for (int n = 1; n <= 3; n++){

                        int cell2\_i = 5 + m;

                        int cell2\_j = 5 + n;

                        if (cell1\_i != cell2\_i || cell1\_j != cell2\_j){

                            fprintf(cnf\_fp, "-%d -%d 0\n",

                                (cell1\_i-1)\*81+(cell1\_j-1)\*9+k,

                                (cell2\_i-1)\*81+(cell2\_j-1)\*9+k);

                        }

                    }

                }

            }

        }

    }

    fclose(cnf\_fp);

    return cnf\_file;

}

//解决数独问题

int SolveSudoku(int CoR){

    //创建数独文件

    char\* sudoku\_file=CreateSudoku(CoR);

    if(sudoku\_file==NULL){

        printf("create failed!\n");

        return 0;

    }

    //数独文件转化为cnf文件

    char\* cnf\_file=Transformcnf(sudoku\_file);

    if(cnf\_file==NULL){

        printf("transform failed!\n");

        return 0;

    }

    //读取CNF文件

    SAT\* sat = Readcnf(cnf\_file);

    if(sat == NULL) {

        printf("Failed to read CNF file\n");

        return 0;

    }

    //解决SAT问题

    printf("what's your strategy?(0/1)");

    int strategy=0;

    scanf("%d",&strategy);

    Result\* result = Solve(sat, strategy);

    if(result == NULL) {

        printf("Failed to solve SAT problem\n");

        Clear(sat->Chead);

        free(sat);

        return 0;

    }

    //输出结果

    if(result->ans == 1){

        printf("Sudoku solution found:\n");

        SaveResult(sat,result,strategy);

        int grid[9][9] = {0};

        //解析SAT解为数独网格

        for(int i=1; i<=729; i++){

            if(result->bl[i] > 0){

                int var = i;

                int row = (var-1) / 81 + 1;

                int col = ((var-1) % 81) / 9 + 1;

                int num = (var-1) % 9 + 1;

                grid[row-1][col-1] = num;

            }

        }

        //打印数独解

        for(int i=0; i<9; i++){

            for(int j=0; j<9; j++){

                printf("%d ", grid[i][j]);

                if((j+1) % 3 == 0 && j < 8) printf("| ");

            }

            printf("\n");

            if((i+1) % 3 == 0 && i < 8) printf("------+-------+------\n");

        }

    }else{

        printf("no solution  for sudoku .\n");

    }

    printf("time:%dms\n", result->time);

    //清理内存

    free(result->bl);

    free(result);

    Clear(sat->Chead);

    free(sat);

    return 1;

}