

**课程设计报告**

**题目：基于SAT的双蜂窝数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： 计算机2201班**

**学 号： U202215345**

**姓 名： 贾竟一**

**指导教师： 李剑军**

**报告日期： 2023.8.31**

**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（数据结构不要使用C++现有的vector等类库）。(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**蜂窝数独游戏**[12]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** I

**参考文献** II

**1.引言** 1

1.1课题背景与意义 1

1.2国内外研究现状 2

1.3课程设计的主要研究工作 2

**2.系统需求分析与总体设计** 3

2.1系统需求分析 3

2.2系统总体设计 3

**3.系统详细设计** 4

3.1有关数据结构的定义 4

3.2主要算法设计 5

**4.系统实现与测试** 7

4.1系统实现 7

4.2系统测试 9

**5.总结与展望** 13

5.1全文总结 13

5.2工作展望 13

**6.体会** 14

**附录** 15

**1引言**

**1.1课题背景与意义**

**1.1.1课题背景**

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。

DPLL算法是基于树/二叉树的回溯搜索算法，主要使用两种基本处理策略：

单子句规则。如果子句集S中有一个单子句L,那么L一定取真值，于是可以从S中删除所有包含L的子句（包括单子句本身），得到子句集S1，如果它是空集，则S可满足。否则对S1中的每个子句，如果它包含文字¬L,则从该子句中去掉这个文字，这样可得到子句集合S2。S可满足当且仅当S2可满足。

单子句传播策略就是反复利用单子句规则化简S的过程。

分裂策略。按某种策略选取一个文字L.如果L取真值，则根据单子句传播策略，可将S化成S2；若L取假值（即¬L成立）时，S可化成S1.

交错使用上述两种策略可不断地对公式化简，并最终达到终止状态，其执行过程可表示为一棵二叉搜索树,如下图所示。

**1.1.2课题意义**

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**

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。

**1.3课程设计的主要研究工作**

1. 读入cnf文件，通过学习DPLL算法，能够对中小规模的算例进行求解，将结果保存在ref文件中并计算求解所用时间。
2. 将蜂窝数独问题抽象成相应的SAT问题，转化为cnf文件，将DPLL算法应用于蜂窝数独游戏中，满足用户需求。
3. 深入学习与理解SAT问题的特点和DPLL算法的设计与优化，撰写实验报告。**2系统需求分析与总体设计**

**2.1系统需求分析**

系统应能够读入 cnf 文件，对相应的SAT问题进行求解，同时对求解策略进行一定的优化。如果可满足应将结果保存在res文件中，计算并对比两次求解所需时间。

系统还应能够设计一种算法自动生成蜂窝数独棋盘，采用从完整合法填充开始，基于挖洞法生成，并以合适的形式输出棋盘，用户可选择是否对当前棋盘进行可行性验证，若要验证，可调用SAT求解器判断用户的填法是否正确。

**2.2系统总体设计**

系统总体由五个模块组成，分别是程序调控与交互界面的display模块，变量、数据结构以及函数定义的define模块，能够读入cnf文件并建立相应数据结构的cnfparser模块，使用DPLL算法的SAT问题求解器solver模块以及可以基于挖洞法随机生成蜂窝数独棋盘，可判断填充是否正确并具有一定交互性的hanidoku模块。（如图2-1所示）

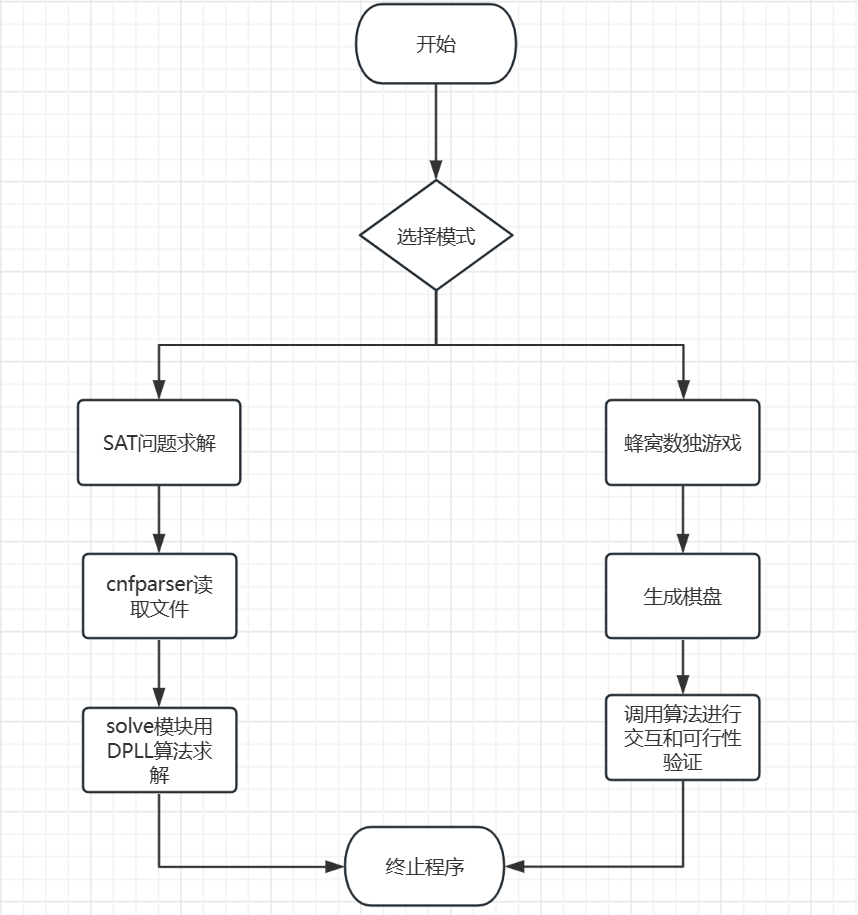


图2-1系统总体设计图

**3系统详细设计**

**3.1有关数据结构的定义**

本程序研究的对象是一个合取范式，包含若干子句，子句中又包含若干文字，本程序所使用的数据结构为十字链表。

其中最基本的节点为node,表示SAT命题中的文字，value保存文字的真值，next指向下一个文字。

clause表示子句，其中num表示子句中文字即node的个数，用head指向第一个文字，next指向下一子句。

numNode则是一个结构体，用来保存在每个位置上元素的x, y坐标和值。其中x表示横坐标，y表示纵坐标，v表示值。（如图3-1所示）

typedef struct node           //保存文字

{

    int value;                //文字名，小于0表示其在句子中的带有非

    node\* next;               //下一变量

} node;

typedef struct clause           //保存子句

{

    int num;                    //标记子句中文字的个数

    node\* head;                 //指向下一个文字

    clause\* next;              //指向下一个句子

} clause;

typedef struct numNode

{

    int x;               //棋盘上元素横坐标

    int y;               //棋盘上元素纵坐标

    int v;               //棋盘上元素的值

}numNode;

图3-1 数据结构定义

数据结构的逻辑关系如图3-2所示



图3-2 数据项之间的联系

**3.2主要算法设计**

solver模块：首先根据单子句规则进行删除，直至找不到单子句，调用启发函数找到一个变元，先复制当前的数据，然后加入对上述变元赋值的单子句进行下一次DPLL过程，返回值为true则求解完毕，返回至为false则在已复制的数据的基础上加入相反的单子句进行DPLL过程。（如图3-3所示）



图2.2 DPLL算法搜索树

基于单子句传播与分裂策略的DPLL算法可以描述为一个如后所示的递归过程**DPLL( *S* )**, DPLL算法也可用非递归实现。

**DPLL( *S*) :**

/\* *S*为公式对应的子句集。若其满足，返回TURE；否则返回FALSE. \*/

**{**

**while(*S*中存在单子句) {**//单子句传播

**在*S*中选一个单子句*L*；**

**依据单子句规则，利用*L*化简*S*；**

**if *S* = Φ return(TRUE);**

**else if (*S*中有空子句 ) return（FALSE）；**

**}**//while

**基于某种策略选取变元*v*； //策略对DPLL性能影响很大**

**if DPLL（*S* ∪*v* ）return(TURE); //在第一分支中搜索**

**return DPLL(*S* ∪¬*v*);//回溯到对*v*执行分支策略的初态进入另一分支**

**}**

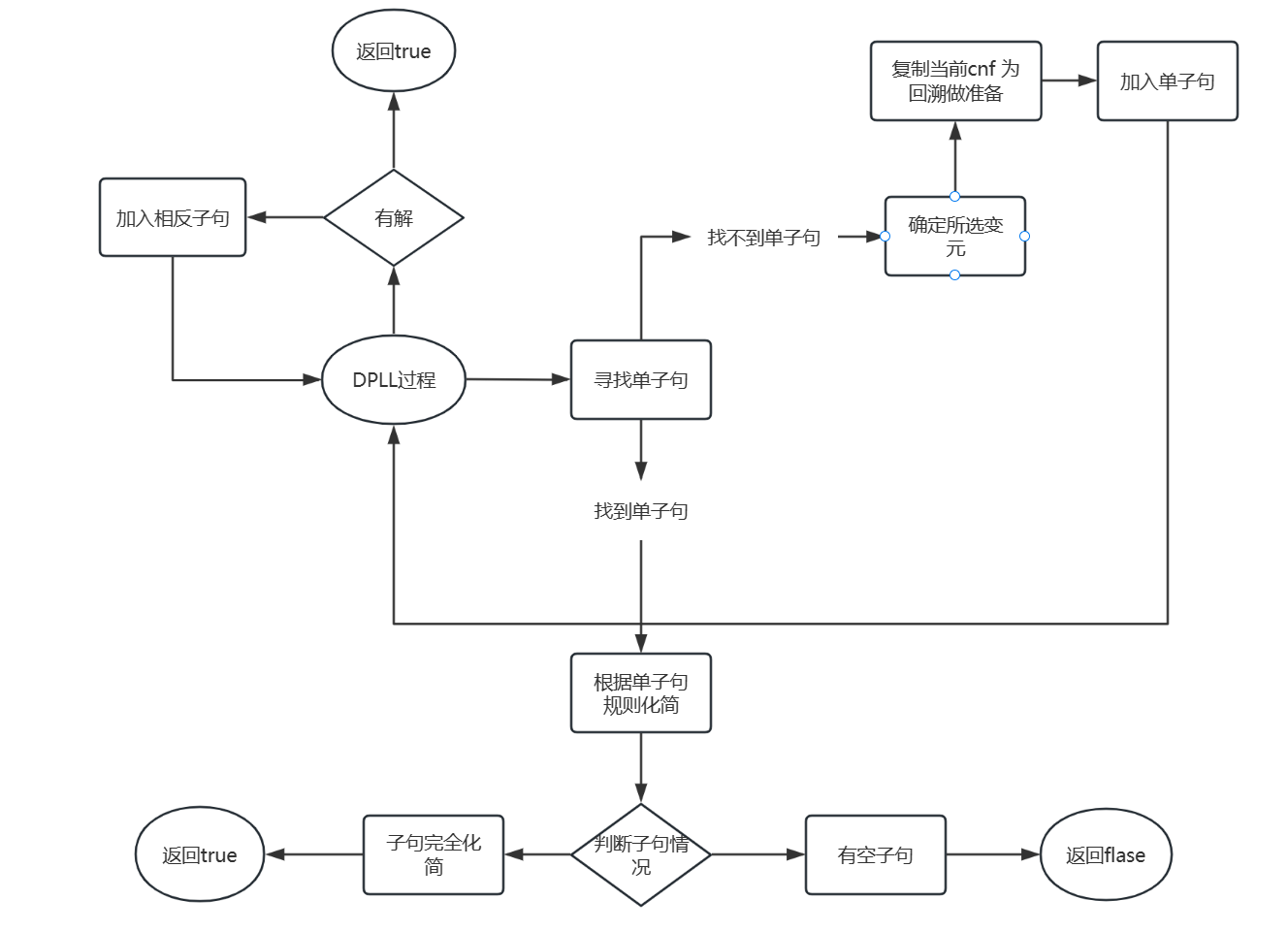


图3-3 solver模块主要算法流程图

cnfparser模块：对cnf文件进行解析，读取cnf文件并生成十字链表。同时也可将求解得到的cnf结果保存到文件中，显示优化前和优化后的问题求解时间。

hanidoku模块：首先根据难度自自定义挖洞的数量，然后随机选取棋盘库中的标准棋盘，基于挖洞法生成棋盘并打印，所挖去的洞的位置标记为0。然后用户可以根据蜂窝数独的行和列进行填充，填充后可以自行选择是否进行可行性验证，验证时生成棋盘约束条件下的cnf文件，读取后调用嵌入的DPLL进行可行性验证。（如图3-4所示）

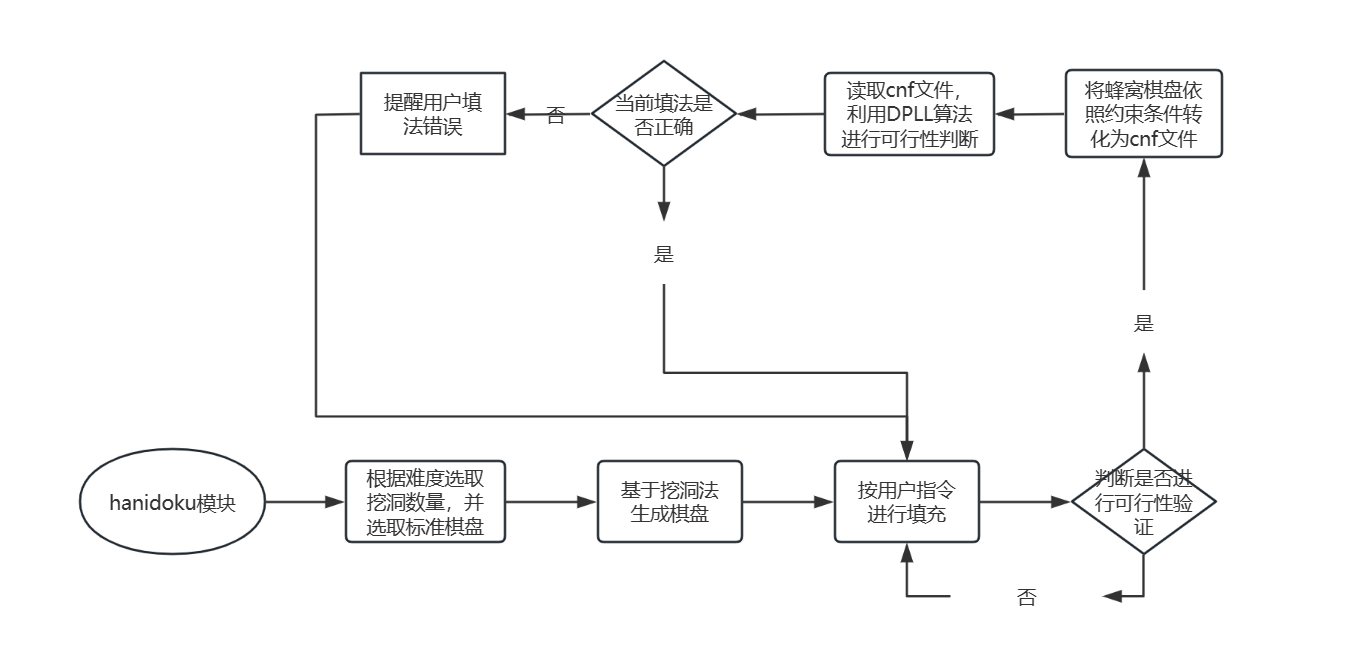


图3-4 hanidoku模块流程图

display模块：打印出菜单界面，内部使用switch语句根据用户输入调用相应模块，同时给出提示并输出结果。

cnfparser模块：读入cnf文件并依次建立相应数据结构。

define模块：提供必要的调用库和函数原型声明以及宏定义。

**4系统实现与测试**

**4.1系统实现**

系统的软件环境：运行环境Windows11，开发环境 Dev C++

系统的硬件环境：CPU为12th Gen Intel(R) Core(TM) i7-12700H

GPU为NVIDIA GeForce RTX 3050Ti。

系统所包含的函数及相应说明参见图4-1。

int ABS(int a);                                                   //辅助计算函数

int read(FILE \*fp);                                               //快读

status ReadCNF(clause\*& cnf, int satORsudoku);                 //读取cnf文件

void destroyClause(clause\*& cnf);                               //删除SAT问题

void removeClause(clause\*& cnf, clause\*& cl);                  //删除子句

void removeNode(clause\* cl, node\*& nd);                         //删除文字

status addClause(clause\* cl, clause\*& cnf);                     //加入单子句

bool isUnitClause(clause\* cl);                                  //子句判断是否为单子句

void delelteSingleClause(clause\* s, clause\*& cnf);              //单子句规则

bool emptyClause(clause\* cnf);                                    //子句判空

void getForgery(clause\*& forgery, clause\* cnf);                  //获取副本

int getMaxBool(clause\* cnf);                                      //获取出现次数最多的变元

int getFirstBool(clause\* cnf);                                   //获取第一个变元

int getNextBool\_3(clause\* cnf);                                  //获取短子句中绝对值出现最多的变元

int getNextBool\_4(clause\* cnf);                                   //获取短子句中出现最多的变元

bool DPLL(clause\*& cnf, int\* v, int satORsudoku);                 //DPLL算法

status printCNF(int s, int\* v, double t1, double t2);             //创建res文件

void ReadSatissudo(char a[][COL], FILE \*fp);                      //从标准棋盘中读取蜂窝数独

void CreatMod(char a[][COL], char b[][COL], int numDigits);       //挖洞法生成随机棋盘

void show(char a[][COL]);                                         //打印蜂窝数独棋盘

void TurnToInt(char a[][COL], numNode b[][10]);                   //得到数独每一个元素的位置坐标

void ToCnf(int a[][COL], const char fileName, int holes);         //得到当前棋盘的cnf文件

void hanidoku();                                                  //进入蜂窝数独游戏

void Turn(numNode a[][10], numNode b[][10]);                   //将蜂窝数独的元素沿着对角线进行翻转

void LinProcecc(numNode a[][10], FILE \*fp, int line);       //对每行进行约束条件限制，得到cnf文件

void Non\_Repeat(FILE \*fp, numNode a[][10]);                       //每两格数字不重复

void AddNode(char a[][COL], int x, int y, int z);                 //向蜂窝数独棋盘中进行加点

图4-1 各函数及其说明

hanidoku模块中各函数的作用及调用关系如图4-2所示

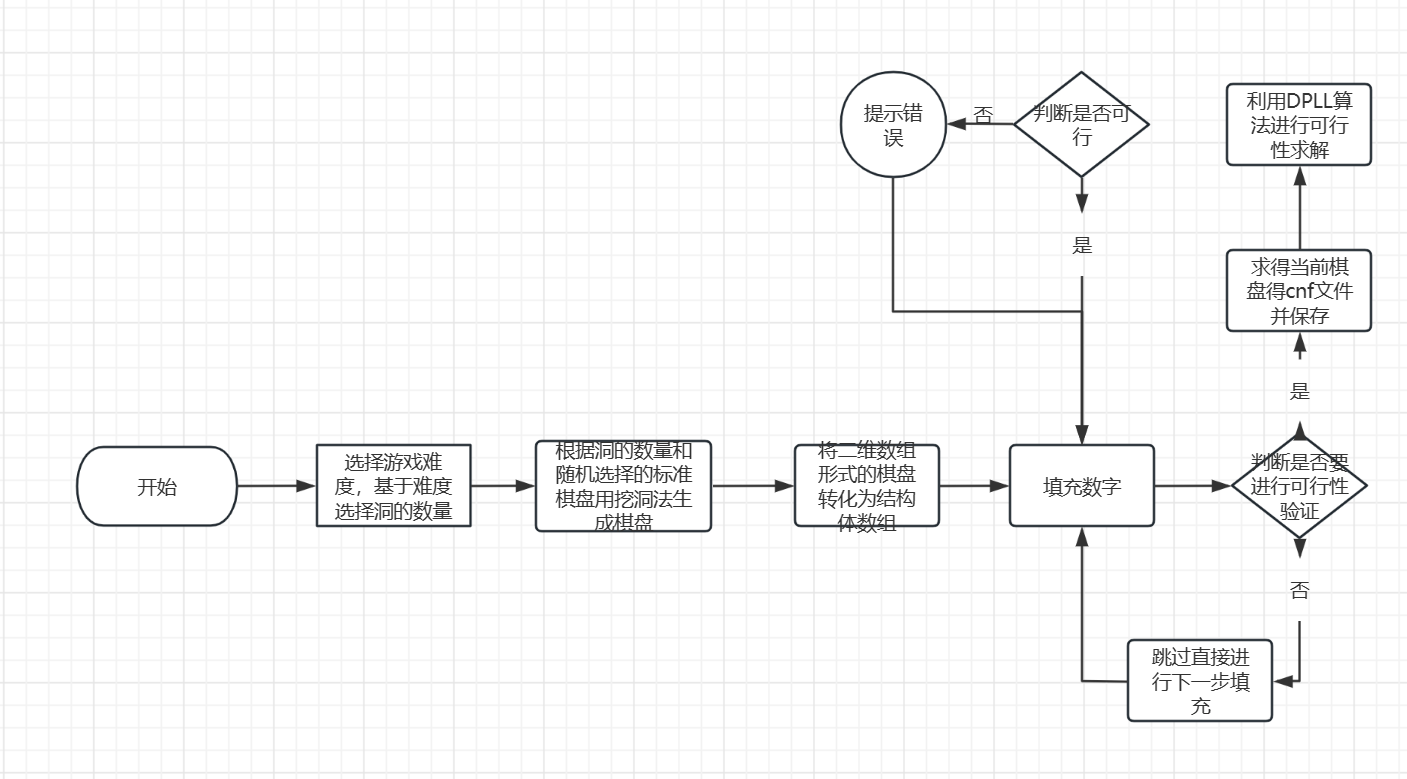


图4-2 hanidoku模块各函数功能及调用关系图

solver模块中各函数的作用及调用关系如图4-3所示

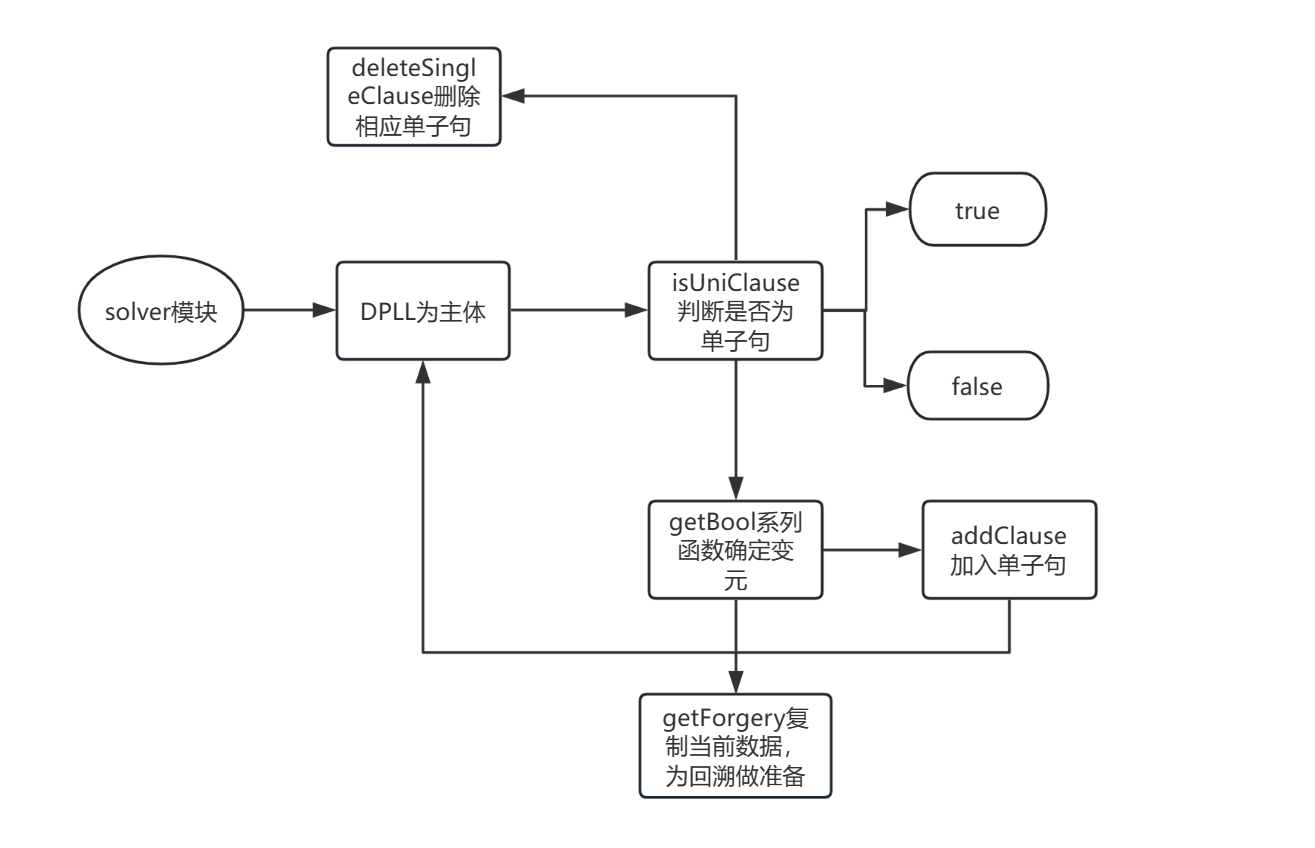


图4-3 solver模块各函数功能及调用关系图

**4.2系统测试**

4.2.1常用软件测试方法

分模块进行测试，先验证各模块功能是否正常，再对模块进行性能测试。

4.2.2各模块功能及设计目标

solver模块：通过DPLL算法设计SAT问题求解器，实现对中小规模的cnf算例的求解并记录求解所需时间，若满足则将解保存到相应的res文件中，同时设计一个优化算法，对已有算法进行优化，提升求解效率。

hanidoku模块：依据难度在已有棋盘的基础上基于挖洞法生成蜂窝数独棋盘，通过输出提示与用户交互，跟据用户输入进行数独的填写，提示，输出答案等操作，能够及时更新数独棋盘，并可根据用户需要判断当前棋盘是否存在合法填充方案。

4.2.3测试大纲

先选择功能测试算例，检查系统是否具有正确求解可满足和不可满足算例的功能，功能性测试算例通过之后再选择性能测试算例，根据变元与子句的多少综合判断算例规模之后，从易到难依次进行测试，观察算例的求解所需时间，判断DPLL算法的优化情况、试探程序能够较快求解的最大算例规模。

4.2.4运行结果

先进行读取、打印cnf文件以及保存结果的功能测试，运行结果如图4-4、4-5、4-6所示

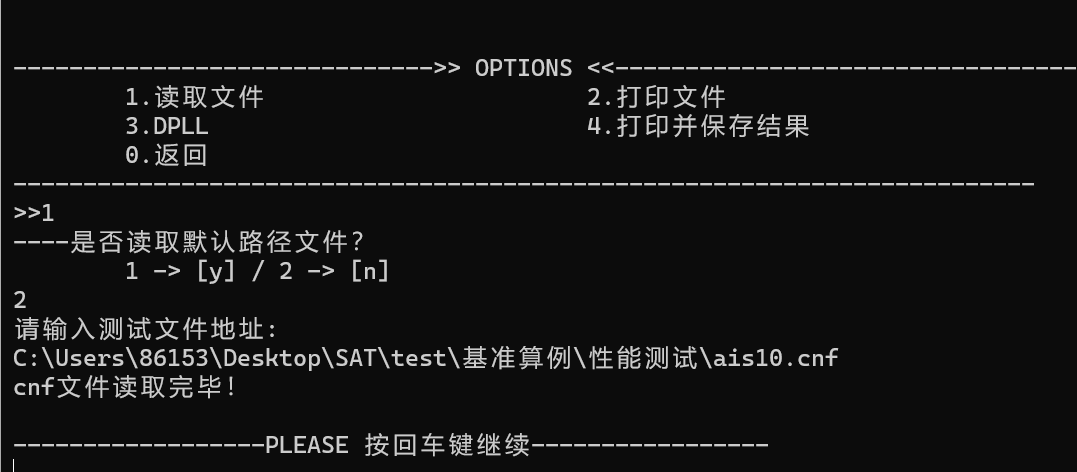


图4-4 读取cnf文件功能测试

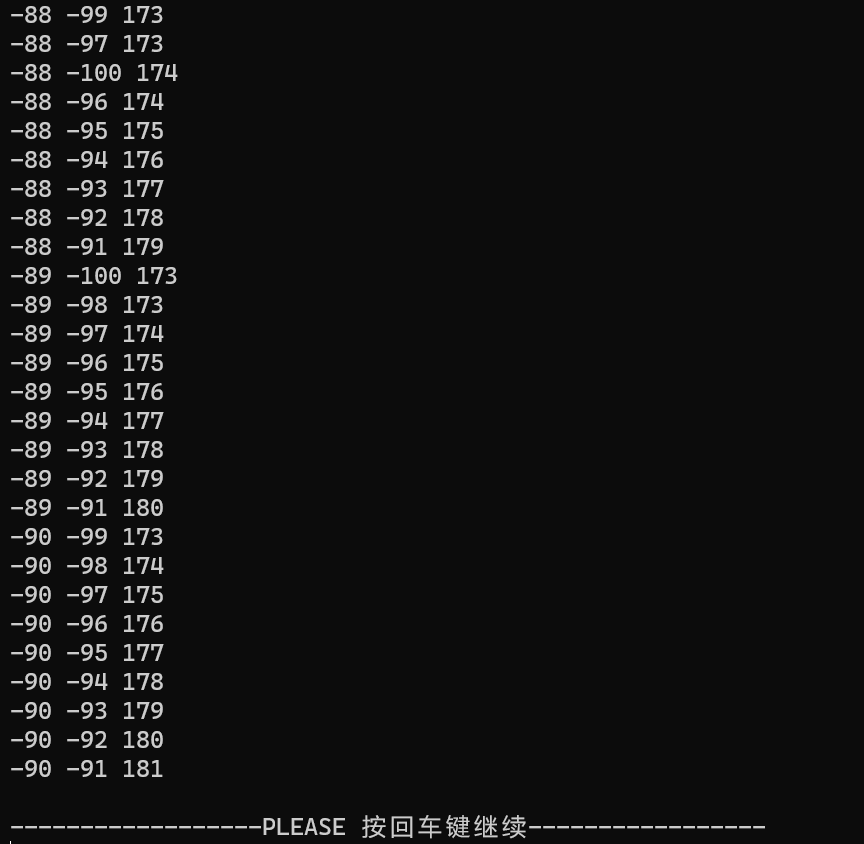


图4-5 打印cnf文件功能测试

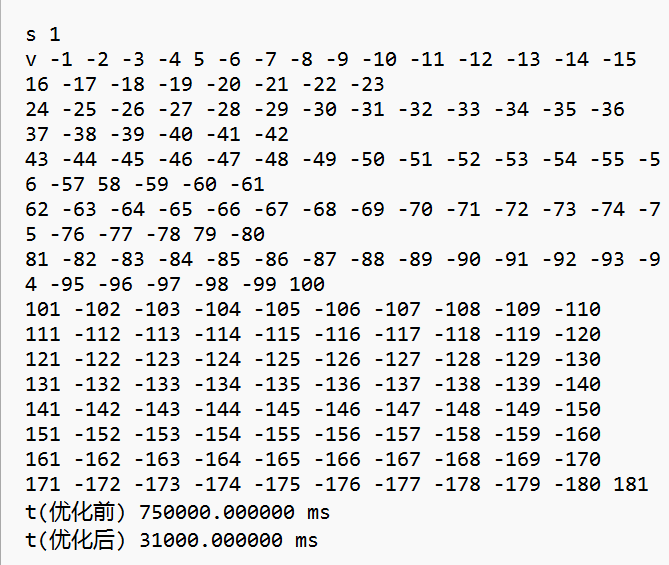
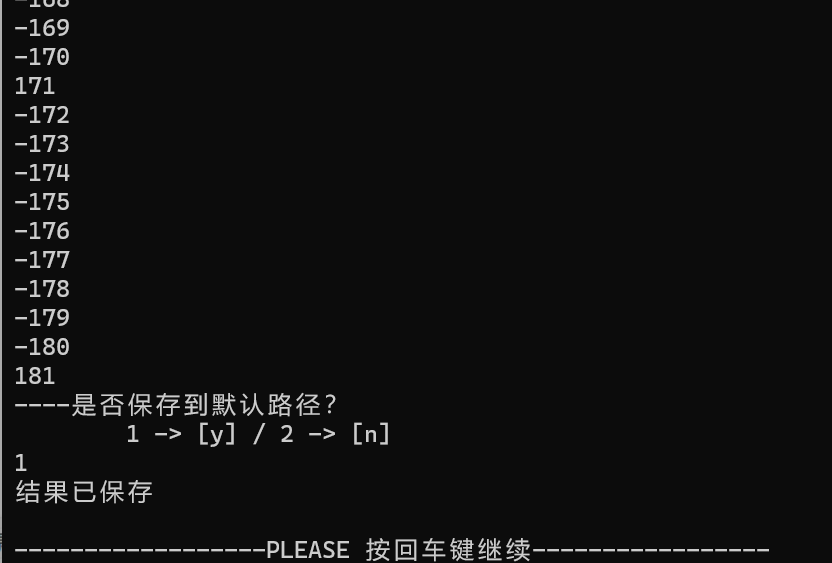


图4-6 输出并保存结果的功能测试

再进行DPLL功能测试，分别选择满足算例和不满足算例进行测试，运行结果如图4-7、4-8所示

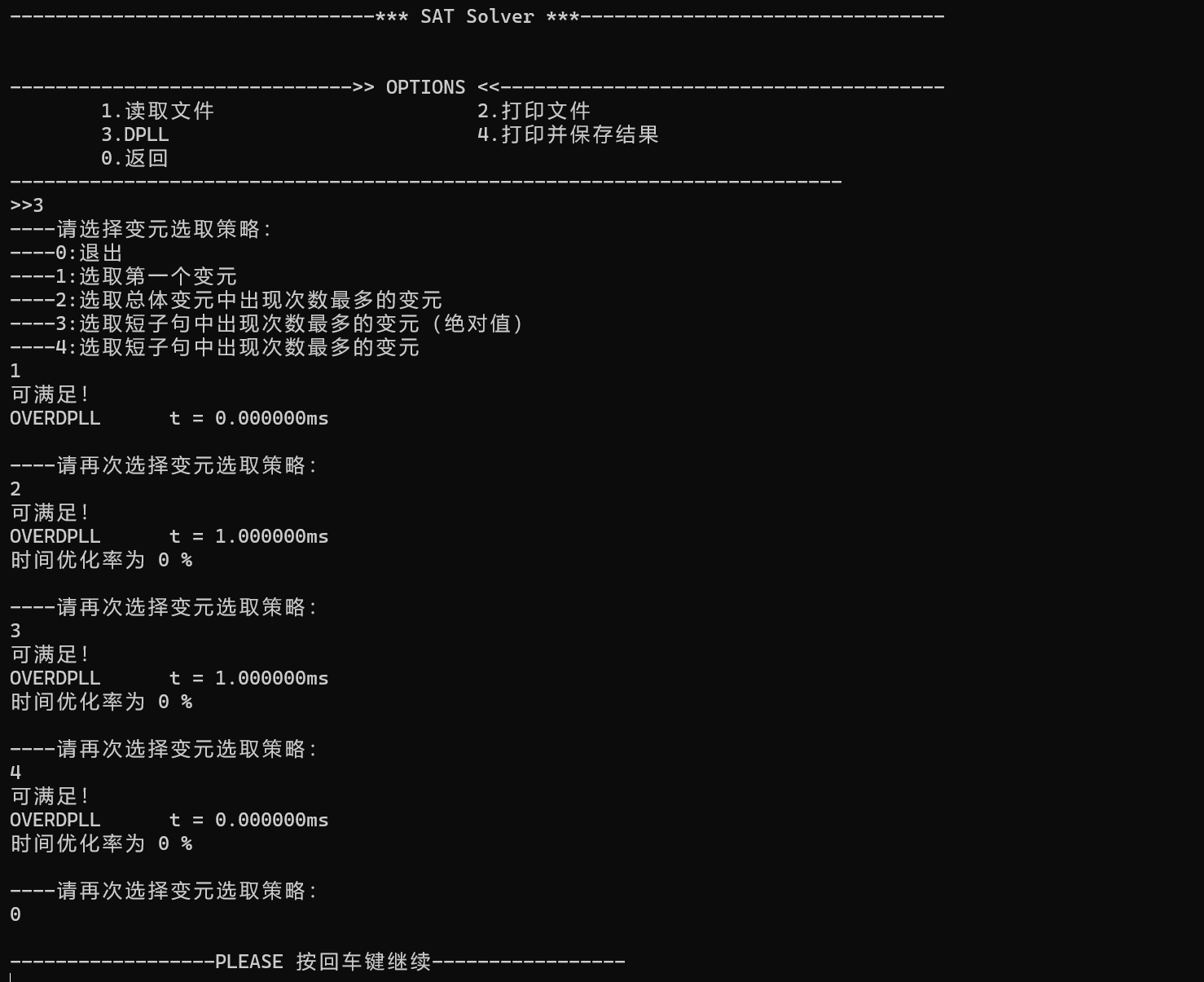


图4-7 可满足算例DPLL功能测试

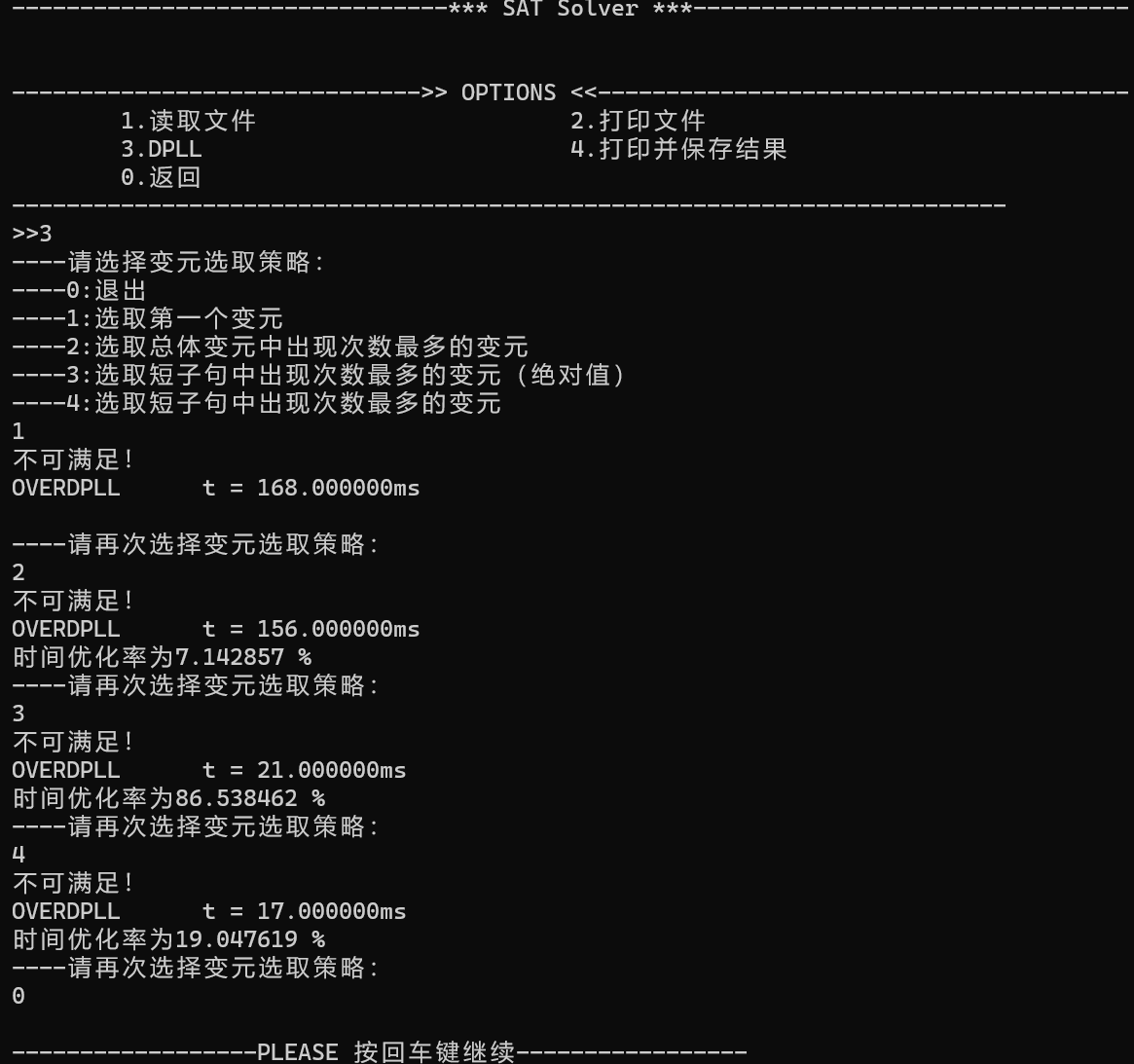


图4-8 不可满足算例DPLL功能测试

之后进行DPLL过程性能测试，依次选取小、中、大规模算例，运行结果如图4-9、4-10、4-11所示

（7cnf20\_90000\_90000\_7.shuffled-20.cnf）

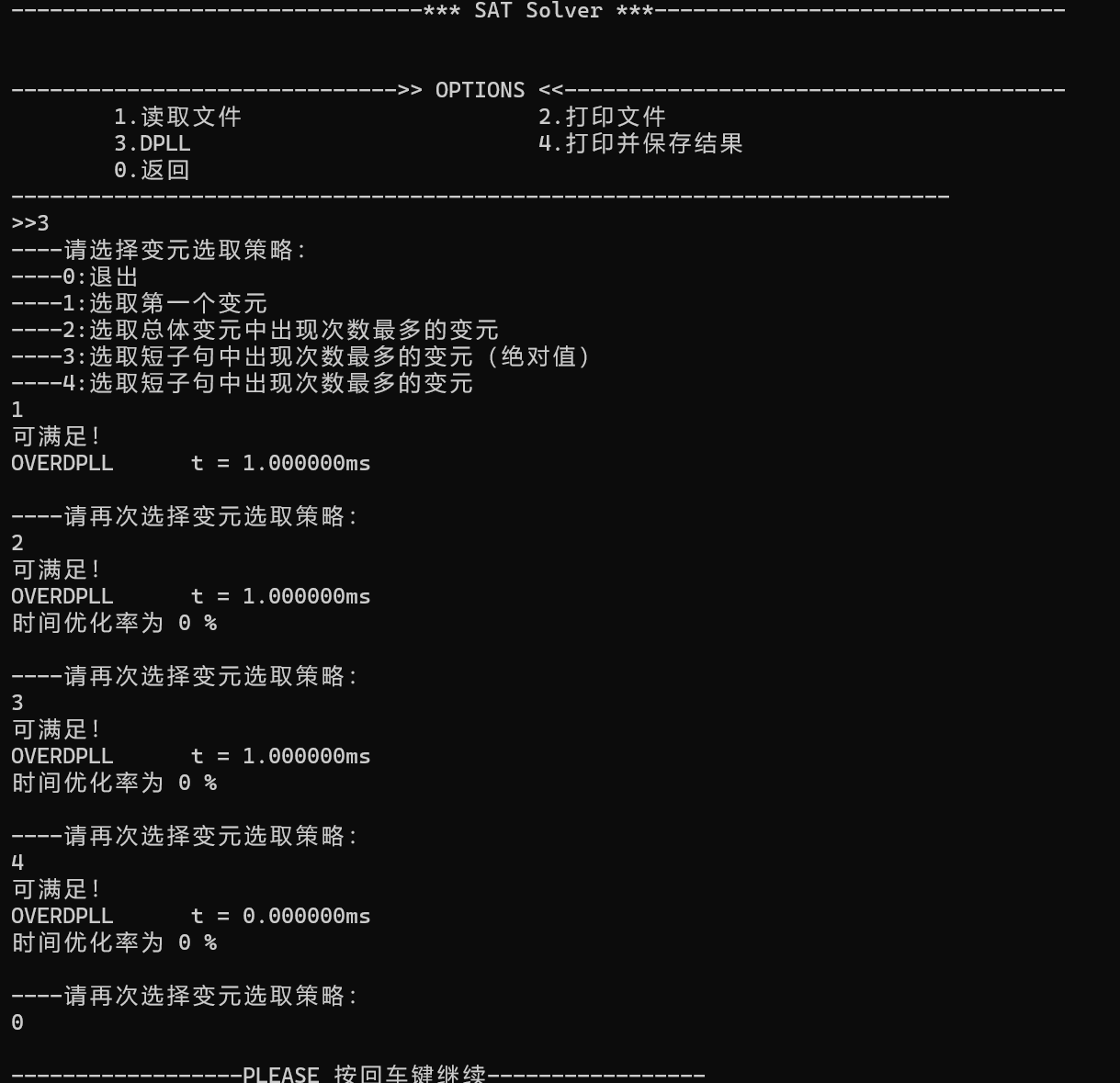


图4-9 小规模算例DPLL性能测试

（bart17.shuffled-231.cnf）

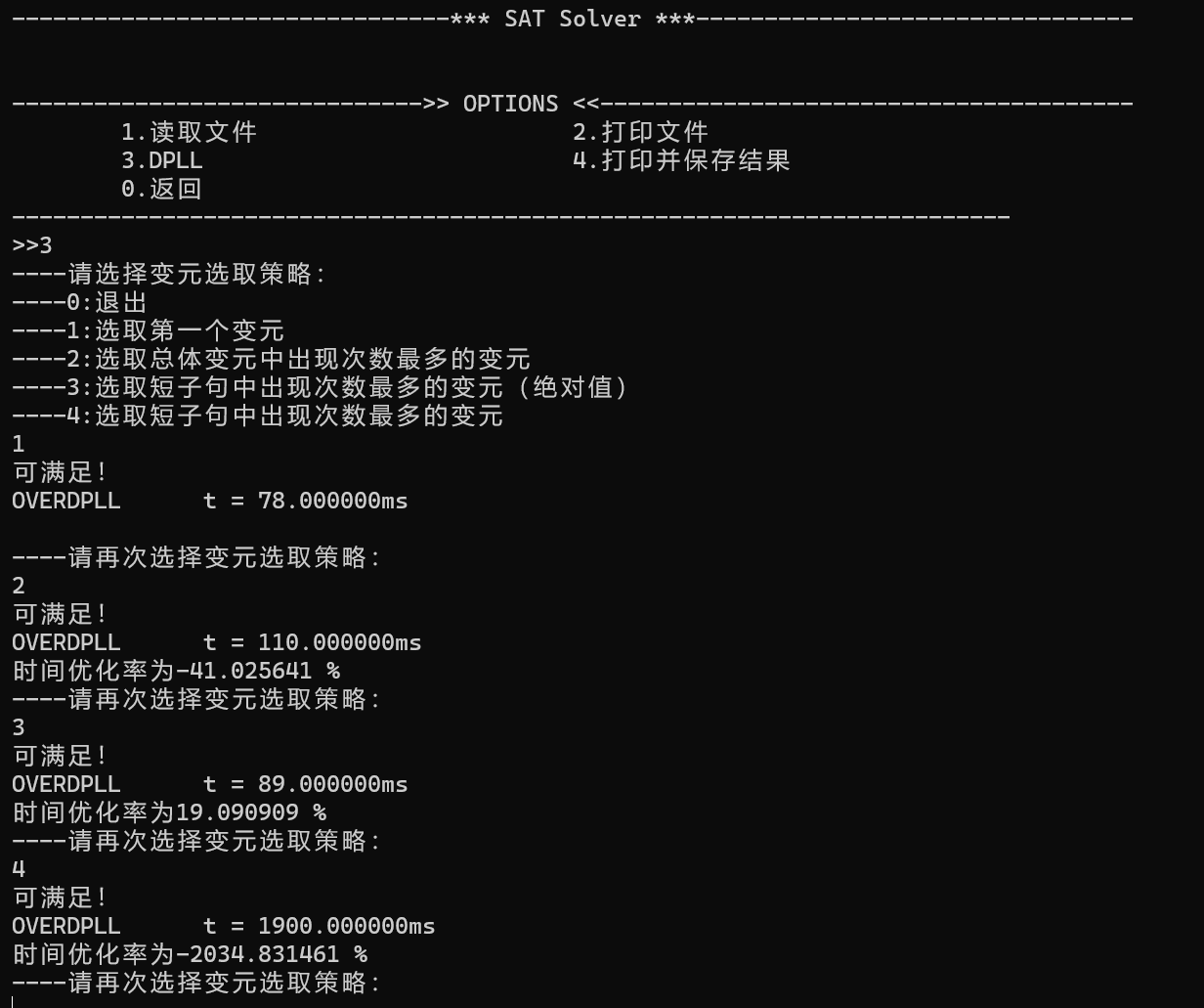


图4-10 中规模算例DPLL性能测试

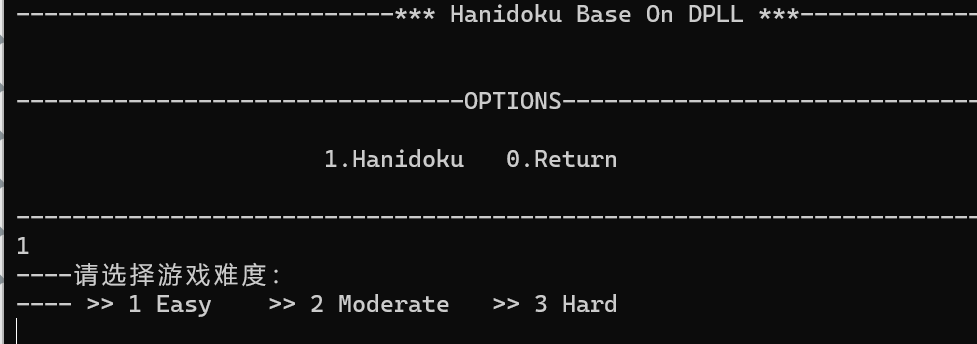
（sud00861.cnf）

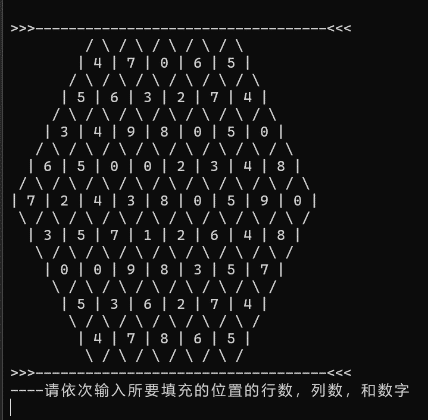
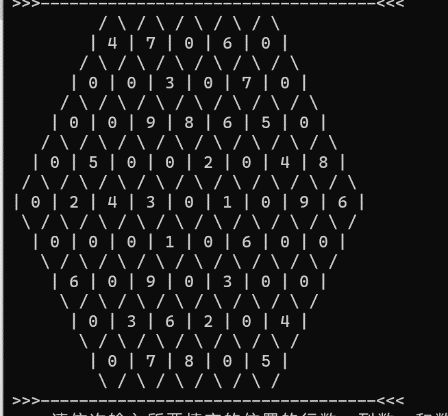


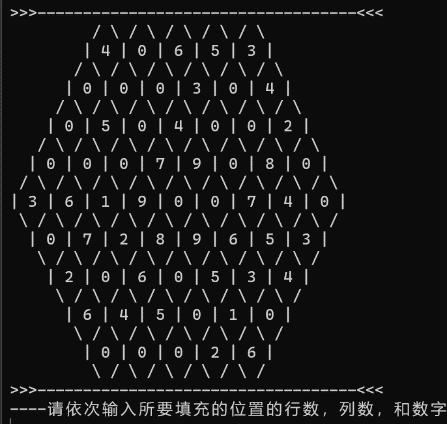
图4-11 大规模算例DPLL性能测试

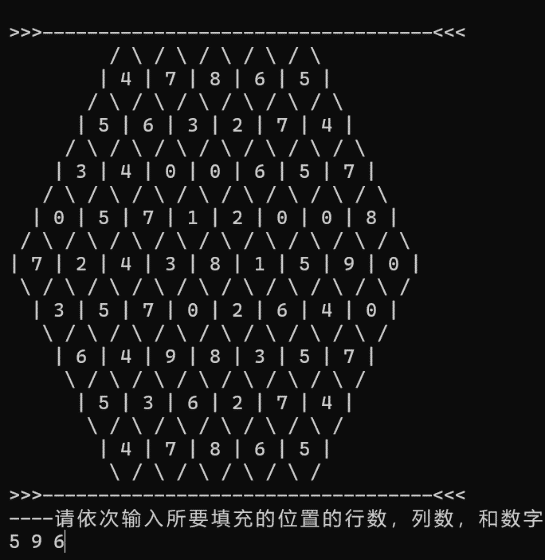
（eh-dp04s04.shuffled-1075.cnf）

最后进行蜂窝数独游戏的功能测试，检验数独游戏的生成、填写、输出答案，可行性验证的功能，运行结果如图4-12、4-13、4-14所示





图4-12 数独游戏生成功能测试（三种不同难度）



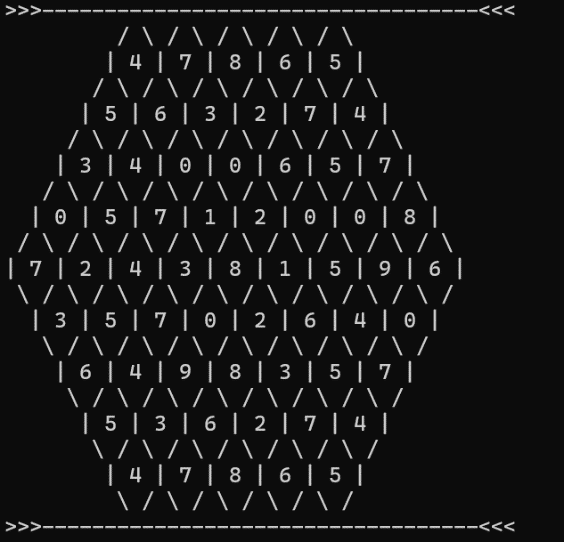


图4-13 数独游戏填写功能测试（填写前、后）



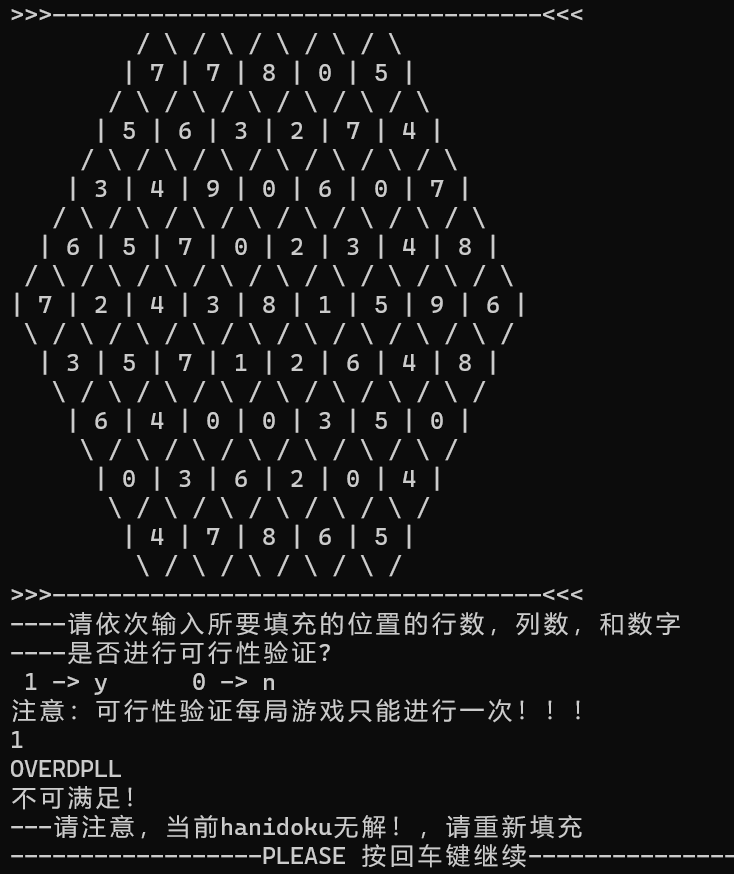


图4-14 数独游戏输出答案功能测试

4.2.5运行结果分析

根据运行结果，该系统读入、打印cnf文件功能正常，能够对SAT问题进行求解且可以将结果保存在res文件中，通过进一步的性能测试，该系统能够在合适的时间内对小、中、大规模的算例验证其满足性并求解。在数独游戏中，根据运行结果，该系统能够基于挖洞法生成不同难度的棋盘，可以让用户进行填写或者输出答案，棋盘可以相应更新。综上所述，该系统各功能正常完备，具有较好的性能，完成了相应的设计目标。

**5总结与展望**

**5.1全文总结**

本次实验的主要工作如下：

（1）读取cnf文件，建立相应的数据结构；

（2）学习并优化DPLL算法，对相应的SAT问题进行求解，判断其满足性并记录求解所需时间；

（3）设计蜂窝数独游戏，挖洞法生成蜂窝数独棋盘,将蜂窝数独游戏的规则转换为SAT问题进行求解。

成功完成了输入输出功能，公式解析与验证，DPLL过程，时间性能的测量，程序优化，SAT应用：蜂窝数独游戏的功能。

**5.2工作展望**

在今后的研究中，将围绕着如下几个方面开展工作

1. 优化DPLL算法中的回溯策略，避免因为选取副本进行回溯占用太多内存，使之能够在合理的时间内化简大算例。
2. 优化变元选取策略，提升求解器的适应性。使求解大规模，不同类型算例的能力更强。

（3）加强蜂窝数独棋盘的图形化，使之更容易被用户接受。

（4）提升蜂窝数独cnf文件的生成效率，同时试着给出用户正确答案的提示，提升程序可玩性。

# 

# 6体会

通过课程设计教学与实践环节，我进一步正确理解与应用了专业知识，增强和提高了分析问题与解决问题的综合能力，加深了对于求解实际问题的基本科研步骤的体会与理解，增强和提升了信息搜索和分析技能，培养了技术总结的基本技能，锻炼了课程设计报告的撰写的能力。

这次的课程设计课程我认为是我学习计算机以来做过的最大型的项目，当时在寒假看到题目的时候还没有学完离散，无法透彻的研究问题。在学习的过程中，有一点是必须要具有的，那就是勤于请教，我与朋友一起探讨之后，才对程序设计有了基本的思路。

在设计DPLL算法的优化时，我发现针对不同算例，其最优变元选取策略也有所不同。最初的算法中，我的代码可以求解37kb的算例，但却无法求解5kb的算例。这就要求我们不断尝试，突破思维定式，寻找到合适的变元选取策略。

在蜂窝数独的设计与求解中，我巧妙的利用了蜂窝的几何旋转特性，在对每一横行进行求解后，利用结构体将棋盘进行翻转，顺利得到了cnf文件。同时在连续性约束代码表达时，我发现往往复杂的规律可以找到简洁的数学表达形式，这启发我面对一些难以一眼得到思路的问题时，不妨静下心来，由特殊到一般，慢慢寻找数学关系。

士不可以不弘毅，任重而道远。

**附录**

**define.h**

#include <math.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<string.h>

#define OK 1

#define ERROR 0

#define Hanidoku 1

#define SAT 0

#define ROW 20

#define COL 40

typedef struct node           //保存文字

{

    int value;                //文字名，小于0表示其在句子中的带有非

    node\* next;               //下一变量

} node;

typedef struct clause           //保存子句

{

    int num;                    //标记子句中文字的个数

    node\* head;                 //指向下一个文字

    clause\* next;              //指向下一个句子

} clause;

typedef struct numNode

{

    int x;

    int y;

    int v;

}numNode;

typedef int status;

int ABS(int a);                                                   //辅助计算函数

int read(FILE \*fp);                                               //快读

status ReadCNF(clause\*& cnf, int satORsudoku);                    //读取cnf文件

void destroyClause(clause\*& cnf);                                 //删除SAT问题

void removeClause(clause\*& cnf, clause\*& cl);                     //删除子句

void removeNode(clause\* cl, node\*& nd);                           //删除文字

status addClause(clause\* cl, clause\*& cnf);                       //加入单子句

bool isUnitClause(clause\* cl);                                    //子句判断是否为单子句

void delelteSingleClause(clause\* s, clause\*& cnf);                //单子句规则

bool emptyClause(clause\* cnf);                                    //子句判空

void getForgery(clause\*& forgery, clause\* cnf);                   //获取副本

int getMaxBool(clause\* cnf);                                      //获取出现次数最多的变元

int getRandBool(clause \*cnf);                                     //随机获取一个变元

bool DPLL(clause\*& cnf, int\* v, int satORsudoku);                 //DPLL算法

status printCNF(int s, int\* v, double t1, double t2);             //创建res文件

void ReadSatissudo(char a[][COL], FILE \*fp);                      //从标准棋盘中读取蜂窝数独

void CreatMod(char a[][COL], char b[][COL], int numDigits);       //挖洞法生成随机棋盘

void show(char a[][COL]);                                         //打印蜂窝数独棋盘

void TurnToInt(char a[][COL], numNode b[][10]);                   //得到数独每一个元素的位置坐标

void ToCnf(int a[][COL], const char fileName, int holes);         //得到当前棋盘的cnf文件

void hanidoku();                                                  //进入蜂窝数独游戏

void Turn(numNode a[][10], numNode b[][10]);                      //将蜂窝数独的元素沿着对角线进行翻转

void LinProcecc(numNode a[][10], FILE \*fp, int line);             //对每行进行约束条件限制，得到cnf文件

void Non\_Repeat(FILE \*fp, numNode a[][10]);                       //每两格数字不重复

void AddNode(char a[][COL], int x, int y, int z);                 //向蜂窝数独棋盘中进行加点

const int N = 50;

int boolCnt, clauseCnt, sumOfAns, branchRule;//变元数量 子句数量  选择策略 分支变元

char fileName[200], WfileName[100];  //cnf文件文件名

int \*ans=NULL; //结果数组

clause\* CNF=NULL; //SAT问题

char map[N][N];

int jjy;

**cnfparser.cpp**

int ABS(int a)

{

    return a > 0 ? a : -a;

}

int read(FILE \* fp)

{

    int flag = 0;

    int x = 0, f = 1;

    char c;

    c = getc(fp);        //读入字符

    while (c < '0' || c > '9')     //非数字

    {

        if (c == '-')

            f = -1, c = getc(fp);

        else

            return inf;

    }

    while (c >= '0' && c <= '9')    //读入数字

    {

        x = x \* 10 + c - '0';

        c = getc(fp);

        if (c == -1)    //文件结束

            flag = 1;

    }

    //跳出循环时c应为空格 换行 或者EOF

    if (!flag && c != ' ' && c != '\n')

        return inf;

    return x \* f;    //返回读入数字的值

}

status ReadCNF(clause\*& cnf, int satORsudoku)

{

    FILE\* fp;

    if (satORsudoku == SAT)

        fp = fopen(fileName, "r");

    else

        fp = fopen(WfileName, "r");

    if (!fp)

        return ERROR;

    node\* pn;

    clause\* pc;

    char ch;

    char check[5] = { ' ', 'c', 'n', 'f', ' ' };

    while ((ch = getc(fp)) == 'c')

        while ((ch = getc(fp)) != '\n')  //当行首为c时读到换行符

            continue;

    //跳出循环时ch应为‘p’

    if (ch != 'p')

           return ERROR;

    for (int i = 0; i < 5; i++)

        if ((ch = getc(fp)) != check[i])

        {

          //读取‘p cnf ’部分

            return ERROR;

        }

    if ((boolCnt = read(fp)) == inf || (clauseCnt = read(fp)) == inf)//读入变元数和子句数

        return ERROR;

    cnf = (clause\*)malloc(sizeof(clause));

    cnf->next = NULL;

    cnf->head = (node\*)malloc(sizeof(node));

    cnf->head->next = NULL;

    cnf->num = 0;

    pc = cnf;  //pc是一个指向子句的指针

    pn = cnf->head;   //pn是一个指向文字的指

        //初始化问题

    for (int i = 1; i <= clauseCnt; i++)

    {

        int dat;

        if ((dat = read(fp)) == inf)

            return ERROR;

        while (dat)//dat不是终止符0时，循环读取后面的数据

        {

            pc->num++;

            pn->value = dat;

            pn->next = (node\*)malloc(sizeof(node));

            if ((dat = read(fp)) == inf)

                return ERROR;

            if (!dat)

                pn->next = NULL;

            pn = pn->next;

        }

        pc->next = (clause\*)malloc(sizeof(clause));

        pc->next->num = 0;

        pc->next->head = (node\*)malloc(sizeof(node));

        if (i == clauseCnt)

        {

            pc->next = NULL;

            break;

        }

        pc = pc->next;

        pn = pc->head;

    }

    //读取cnf文件

    fclose(fp);

    return OK;printf("WWWW\n");

}

status printCNF(int s, int\* v, double t1, double t2)

{

    FILE\* fp;

    if (!(fp = fopen(fileName, "w+")))//创建res文件

    {

        printf("文件创建失败!\n");

        return ERROR;

    }

    fprintf(fp, "s %d\nv ", s);

    if (s)//如果可满足

        for (int i = 1; i <= boolCnt; i++)//写入结果

        {

            if (v[i])

                fprintf(fp, "%d ", i);

            else

                fprintf(fp, "%d ", -i);

        }

    fprintf(fp, "\nt(优化前) %lf ms", t1 \* 1000);

    fprintf(fp, "\nt(优化后) %lf ms", t2 \* 1000);

    fclose(fp);

    return OK;

}

**solver.cpp**

void destroyClause(clause\*& cnf)

{

    clause\* ppc, \* pc2; //指向子句的指针

    node\* pn1, \* pn2;   //指向文字的指针

    ppc = cnf;

    while (ppc)

    {

        pn1 = ppc->head;   //依次删除子句

        while (pn1)

            pn2 = pn1->next, free(pn1), pn1 = pn2;  //依次删除文字

        pc2 = ppc->next, free(ppc), ppc = pc2;

    }

    cnf = NULL;

}

void removeClause(clause\*& cnf, clause\*& cl)

{

    if (cl == cnf) //如果是头子句

        cnf = cnf->next;  //头子句的next变为头子句

    else

    {

        clause\* ppc = cnf;

        while (ppc && ppc->next != cl)

            ppc = ppc->next;     //定位要删除的子句

        ppc->next = ppc->next->next;

    }

    node\* pn1, \* pn2;

    for (pn1 = cl->head; pn1;)

        pn2 = pn1->next, free(pn1), pn1 = pn2;  //删除文字

    free(cl);

    cl = NULL;

}

void removeNode(node\*& head, node\*& nd)

{

    node\* ppn = head;

    if (ppn == nd)  //如果要删除头节点

        head = head->next;  //头节点的next成为头节点

    else

    {

        while (ppn && ppn->next != nd)

            ppn = ppn->next;  //定位节点

        ppn->next = ppn->next->next;

    }

    free(nd);

    nd = NULL;

}

status addClause(clause\* cl, clause\*& cnf)

{

    if (cl)//首插法插入单子句

    {

        cl->next = cnf;

        cnf = cl;

        return OK;

    }

    return ERROR;

}

bool isUnitClause(clause\* cl)

{

    if (cl->head != NULL) {

        if (cl->head->next == NULL)

            return true;

    }

    return false;

}

void delelteSingleClause(clause\* s, clause\*& cnf)

{

    clause\* tmp;

    int n = s->head->value;//要删除的变元的值

    for (clause\* ppc = cnf; ppc; ppc = tmp) //遍历所有子句

    {

        tmp = ppc->next;//用temp存储当前子句的next 当前子句被删除后可以找到下一子句

        for (node\* ppn = ppc->head; ppn; ppn = ppn->next)

        {

            if (ppn->value == n)

            {

                removeClause(cnf, ppc);//如果包含该变元 删除该子句

                break;

            }

            if (ppn->value == -n)

            {

                removeNode(ppc->head, ppn);//如果包含该变元的反 删除该节点

                ppc->num--;

                break;

            }

        }

    }

}

bool emptyClause(clause\* cnf)

{

    for (clause\* p = cnf; p; p = p->next)

        if (!p->head)

            return true;

    return false;

}

void getForgery(clause\*& forgery, clause\* cnf)

{

    clause\* ppc, \* pc;  //指向子句的指针

    node\* ppn, \* pn;    //指向文字的指针

    forgery = (clause\*)malloc(sizeof(clause));

    forgery->head = (node\*)malloc(sizeof(node));

    forgery->next = NULL;

    forgery->head->next = NULL;

    forgery->num = 0;

    //初始化问题

    for (pc = cnf, ppc = forgery; pc != NULL; pc = pc->next, ppc = ppc->next)//复制当前cnf 为回溯做准备

    {

        for (pn = pc->head, ppn = ppc->head; pn != NULL; pn = pn->next, ppn = ppn->next)

        {

            ppc->num++;

            ppn->value = pn->value;

            ppn->next = (node\*)malloc(sizeof(node));

            ppn->next->next = NULL;

            if (pn->next == NULL)

                free(ppn->next), ppn->next = NULL;

        }

        ppc->next = (clause\*)malloc(sizeof(clause));

        ppc->next->head = (node\*)malloc(sizeof(node));

        ppc->next->next = NULL;

        ppc->next->head->next = NULL;

        ppc->next->num = 0;

        if (pc->next == NULL)

            free(ppc->next->head), free(ppc->next), ppc->next = NULL;

    }

}

int getMaxBool(clause\* cnf)   //选择出现次数最多的变元

{

    int\* cnt = (int\*)malloc(sizeof(int) \* (boolCnt \* 2 + 1));

    for (int i = 0; i <= boolCnt \* 2; i++)  //初始化cnt数组

        cnt[i] = 0;

    for (clause\* pc = cnf; pc; pc = pc->next)

        for (node\* pn = pc->head; pn; pn = pn->next)

        {

            if (pn->value > 0)

                cnt[pn->value]++;

            else

                cnt[boolCnt - pn->value]++;

        }

    int maxBool, maxTimes = 0;

    for (int i = 1; i <= boolCnt; i++)   //选择变元的正面

        if (cnt[i] > maxTimes)

            maxTimes = cnt[i], maxBool = i;

    if (maxTimes == 1)  //只剩下变元的反

    {

        for (int i = boolCnt + 1; i <= boolCnt \* 2; i++)

            if (cnt[i] > maxTimes)

                maxTimes = cnt[i], maxBool = boolCnt - i;

    }

    free(cnt);

    return maxBool;

}

int getMaxBool\_hanidoku(clause\* cnf)  //对于变元数已知的数独问题 不再动态分配cnt数组的存储空间

{

    int cnt[boolCnt] = { 0 };

    int mid = boolCnt / 2 + 1;

    for (clause\* pc = cnf; pc; pc = pc->next)

        for (node\* pn = pc->head; pn; pn = pn->next)

        {

            if (pn->value > 0)

                cnt[pn->value]++;

            else

                cnt[mid - pn->value]++;

        }

    int maxBool, maxTimes = 0;

    for (int i = 1; i <= mid; i++)   //先找变元的反面

        if (cnt[i + mid] > maxTimes)

            maxTimes = cnt[i], maxBool = -i;

    if (maxTimes == 0)     //只剩下变元的正面

    {

        for (int i = 1; i <= mid; i++)

            if (cnt[i] > maxTimes)

                maxTimes = cnt[i], maxBool = i;

    }

    printf("%d\n", 666);

    return maxBool;

}

int getRandBool(clause \*cnf)

{

    srand((unsigned)time(NULL));

    clause \* pc = cnf;

    while(pc -> next) pc = pc -> next;

    node \* pn;

    int r2 = rand() % (pc -> num);

    int j;

    for(pn = pc -> head, j = r2; j; j --) pn = pn -> next;

    return pn -> value;

}

int getMaxBool\_2(clause\* cnf)  //选取总出现次数最多的变元

{

    int\* cnt = (int\*)malloc(sizeof(int) \* (boolCnt \* 2 + 1));

    for (int i = 0; i <= boolCnt \* 2; i++)  //初始化cnt数组

        cnt[i] = 0;

    for (clause\* pc = cnf; pc; pc = pc->next)

        for (node\* pn = pc->head; pn; pn = pn->next)

        {

            if (pn->value > 0)

                cnt[pn->value]++;

            else

                cnt[boolCnt - pn->value]++;

        }

    int maxBool, maxTimes = 0;

    for (int i = 1; i <= boolCnt; i++)   //选择变元的正面

        if (cnt[i] > maxTimes)

            maxTimes = cnt[i], maxBool = i;

    if (maxTimes == 1)  //只剩下变元的反

    {

        for (int i = boolCnt + 1; i <= boolCnt \* 2; i++)

            if (cnt[i] > maxTimes)

                maxTimes = cnt[i], maxBool = boolCnt - i;

    }

    free(cnt);

    return maxBool;

}

bool DPLL(clause\*& cnf, int\* v, int satORsudoku)

{

    int flag = 1;

    clause\* pc;

    while (flag)//能找到单子句

    {

        flag = 0;

        pc = cnf;

        while (pc && !isUnitClause(pc))

        {

            //遍历寻找单子句

            pc = pc->next;

        }

        if (pc != NULL)//找到单子句pc

        {

            if (pc->head->value > 0)   //在结果数组中保存pc的值

                v[pc->head->value] = 1;

            else

                v[-pc->head->value] = 0;

            delelteSingleClause(pc, cnf);  //根据单子句规则化简

            if (cnf == NULL)        //子句已完全化简

                return true;

            else if (emptyClause(cnf)) //存在空子句

                return false;

            flag = 1;   //找到了单子句

        }

    }

    int maxBool;

    if (satORsudoku == Hanidoku)

        maxBool = getRandBool(cnf);

    else

    {

        if (branchRule == 1)    maxBool = getMaxBool(cnf);

        else if(branchRule == 5)

            maxBool = getRandBool(cnf);

    }

    clause\* newSingleClause = (clause\*)malloc(sizeof(clause)), \*forgery;

    newSingleClause->head = (node\*)malloc(sizeof(node));

    newSingleClause->next = NULL;

    newSingleClause->head->value = maxBool;

    newSingleClause->head->next = NULL;

    newSingleClause->num = 1;

    getForgery(forgery, cnf);     //复制当前cnf 为回溯做准备

    addClause(newSingleClause, forgery);      //加入单子句

    if (DPLL(forgery, v, satORsudoku) == true)

    {

        return true;

    }

    //如果结果为false

    destroyClause(forgery);  //删除副本

    newSingleClause = (clause\*)malloc(sizeof(clause));

    newSingleClause->head = (node\*)malloc(sizeof(node));

    newSingleClause->next = NULL;

    newSingleClause->head->value = -maxBool;

    newSingleClause->head->next = NULL;

    newSingleClause->num = 1;

    addClause(newSingleClause, cnf);    //加入相反的单子句

    getForgery(forgery, cnf);

    bool ans = DPLL(forgery, v, satORsudoku); //DPLL

    destroyClause(forgery);   //删除副本

    return ans;  //返回ans

}

**hanidoku.cpp**

void ReadSatissudo(char a[][COL], FILE \*fp)

{

    for (int i = 0; i < ROW; i++)

    {

        fgets(a[i], COL, fp);

    }

    fclose(fp);

    return ;

}

void CreatMod(char a[][COL], char b[][COL], int numDigits)

{

    int i,j,k;

    srand((unsigned)time(NULL));

    for( i = 0; i < ROW ; i ++)

        for( j = 0; j < COL ; j++)

            b[i][j] = a[i][j];

    int c[numDigits][2];

    int m,flag = 0;

    for(i = 0; i < numDigits; i ++)

    {

        flag = 0;

        j = rand()%9 + 1;

        k = rand()%9 + 1;

        if(j % 2 == 1)

        {

            j = j \* 2 - 1;

            k = k \* 4 - 2;

        }

        else

        {

            j = j \* 2 - 1;

            k = k \* 4;

        }

        for(m = 0; m < i ; m++)

            if( j == c[m][0] && k == c[m][1])

            {

                flag = 1;

                break;

            }

        if(flag == 0 && b[j][k] >= '1' && b[j][k] <= '9')

        {

            b[j][k] = '0';

            c[i][0] = j;

            c[i][1] = k;

        }

        else i --;

    }

}

void show(char a[][COL])

{

    system("cls");

    printf("\n\n>>>-----------------------------------<<<\n");

    for(int i = 0; i < ROW - 1; i ++)

    {

        printf("%s", a[i]);

    }

    printf(">>>-----------------------------------<<<\n");

    printf("----请依次输入所要填充的位置的行数，列数，和数字\n");

    return ;

}

void TurnToInt(char a[][COL], numNode b[][10])

{

    memset(b, 0, sizeof(b));

    int cnt;

    for(int i = 0; i < ROW - 1; i ++)

    {

        cnt = 1;

        if(i % 2 == 0) continue;

        for(int j = 0; j < COL; j ++)

        {

            if(a[i][j] >= '0' && a[i][j] <= '9')

            {

                b[i / 2 + 1][cnt ++].v = a[i][j] - '0';

                b[i / 2 + 1][cnt - 1].x = i/2 + 1;

                b[i / 2 + 1][cnt - 1].y = cnt - 1;

            }

        }

        while(cnt <= 9){

            b[i / 2 + 1][cnt ++].v = 0;

            b[i / 2 + 1][cnt - 1].x = i/2 + 1;

            b[i / 2 + 1][cnt - 1].y = cnt - 1;

        }

    }

    return;

}

void Non\_Repeat(FILE \*fp, numNode a[][10])

{

    for(int i = 1; i <= 9; i ++)

        for(int j = 1; j <= 9; j ++)

        {

            if(j <= 5)

            {

                for(int k = 1; k <= 4 + j; k ++)

                {

                    for(int q = 1; q + k <= j + 4; q ++)

                    fprintf(fp, "%d %d %d\n",-(i + a[j][k].x\* 100 + a[j][k].y\*10), -(i + a[j][q + k].x\*100 + a[j][q + k].y \* 10), 0);

                }

            }

            else

            {

                for(int k = 1; k <= 14 - j; k ++)

                {

                    for(int q = 1; q + k <= 14 - j; q ++)

                    fprintf(fp, "%d %d %d\n",-(i + a[j][k].x \* 100 + a[j][k].y\*10), -(i + a[j][q + k].x\*100 + a[j][q + k].y \* 10), 0);

                }

            }

        }

}

void LinProcecc(numNode a[][10], FILE \*fp, int line)

{

    int st, end, l;//必出现的连续数字起始与结束数值，所容纳变元的数量

    //对于确定位置的数字cnf

    if(line <= 5)

    {

        st = 5 - line + 1;

        end = 2 \* 5 - st;

        l = 4 + line;

        for(int j = st; j <= end; j ++)

        {

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

            {

                fprintf(fp, "%d ", a[line][i].x \* 100 + a[line][i].y \* 10 + j);

            }

            fprintf(fp, "%d\n", 0);

        }

        for(int i = 1; i <= 5 - line; i ++)

        {

            for(int j = 5 + line; j < 5 + line + i; j ++)

            {

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

                {

                    fprintf(fp, "%d %d ", 100 + 10 \* k + i, 100 + 10 \* k + j);

                }

                fprintf(fp, "%d\n", 0);

            }

        }

    }

    else

    {

        st = line - 4;

        end = 2 \* 5 - st;

        l = 14 - line;

        int fc = 10 - line;

        for(int j = st; j <= end; j ++)

        {

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

            {

                fprintf(fp, "%d ", a[line][i].x \* 100 + a[line][i].y \* 10 + j);

            }

            fprintf(fp, "%d\n", 0);

        }

        //对于连续性约束条件下的数字进行cnf

        for(int i = 1; i <= 5 - fc; i ++)

        {

            for(int j = 5 + fc; j < 5 + fc + i; j ++)

            {

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

                {

                    fprintf(fp, "%d %d ", 100 + 10 \* k + i, 100 + 10 \* k + j);

                }

                fprintf(fp, "%d\n", 0);

            }

        }

    }

    return;

}

void Turn(numNode a[][10], numNode b[][10])

{

    memset(b, 0, sizeof(b));

    b[1][1] = a[1][5], b[1][2] = a[2][6], b[1][3] = a[3][7], b[1][4] = a[4][8], b[1][5] = a[5][9];

    b[2][1] = a[1][4], b[2][2] = a[2][5], b[2][3] = a[3][6], b[2][4] = a[4][7], b[2][5] = a[5][8], b[2][6] = a[6][8];//2

    b[3][1] = a[1][3], b[3][2] = a[2][4], b[3][3] = a[3][5], b[3][4] = a[4][6], b[3][5] = a[5][7], b[3][6] = a[6][7], b[3][7] = a[7][7];//3

    b[4][1] = a[1][2], b[4][2] = a[2][3], b[4][3] = a[3][4], b[4][4] = a[4][5], b[4][5] = a[5][6], b[4][6] = a[6][6], b[4][7] = a[7][6], b[4][8] = a[8][6];

    b[5][1] = a[1][1], b[5][2] = a[2][2], b[5][3] = a[3][3], b[5][4] = a[4][4], b[5][5] = a[5][5], b[5][6] = a[6][5], b[5][7] = a[7][5], b[5][8] = a[8][5], b[5][9] = a[9][5];

    b[6][1] = a[2][1], b[6][2] = a[3][2], b[6][3] = a[4][3], b[6][4] = a[5][4], b[6][5] = a[6][4], b[6][6] = a[7][4], b[6][7] = a[8][4], b[6][8] = a[9][4];

    b[7][1] = a[3][1], b[7][2] = a[4][2], b[7][3] = a[5][3], b[7][4] = a[6][3], b[7][5] = a[7][3], b[7][6] = a[8][3], b[7][7] = a[9][3];

    b[8][1] = a[4][1], b[8][2] = a[5][2], b[8][3] = a[6][2], b[8][4] = a[7][2], b[8][5] = a[8][2], b[8][6] = a[9][2];

    b[9][1] = a[5][1], b[9][2] = a[6][1], b[9][3] = a[7][1], b[9][4] = a[8][1], b[9][5] = a[9][1];

}

void ToCnf(numNode a[][10], const char fileName[], int holes)

{

    FILE \*fp;

    fp = fopen(fileName, "w");

    if (fp == NULL)

    {

        printf("无法打开文件：%s\n", fileName);

        return;

    }

    fprintf(fp, "%c %s %d %d\n", 'p', "cnf", 549,  5272 - holes);

    for (int x = 1; x <= 9; ++x) //将已知条件转化为cnf

    {

        for (int y = 1; y <= 9; ++y)

            if(a[x][y].v != 0)

            {

                fprintf(fp, "%d %d\n", (a[x][y].x)\*100 + (a[x][y].y)\*10 + a[x][y].v, 0);

            }

    }

    numNode b[10][10];

    for(int i = 1; i <= 9; i ++)    LinProcecc(a, fp, i);

    Non\_Repeat(fp, a);

    Turn(a, b);//生成新的蜂窝数独，将反对角线转化成行

    for(int i = 1; i <= 9; i ++)    LinProcecc(b, fp, i);

    Non\_Repeat(fp, b);

    Turn(a, b); Turn(b, a);

    for(int i = 1; i <= 9; i ++)    LinProcecc(a, fp, i);

    Non\_Repeat(fp, a);

    fclose(fp);

    return ;

}

void AddNode(char a[][COL], int x, int y, int z)

{

    x = x\*2 - 1;

    int cnt = 0;

    for(int i = 0; i < y; )

    {

        if(a[x][cnt] <= '9' && a[x][cnt] >= '0')

        {

            cnt++;  i ++;

        }

        else cnt++;

        if(i == y)

        {

            cnt--;

            a[x][cnt] = z + '0';

            break;

        }

    }

    show(a);

    return ;

}

void hanidoku()

{

    printf("----请选择游戏难度：\n");

    printf("---- >> 1 Easy\t  >> 2 Moderate\t  >> 3 Hard\n");

    int ch, holes;

    scanf("%d", &ch);

    if(ch == 1) holes = 10;

    else if(ch == 2) holes = 25;

    else holes = 30;

    srand((unsigned)time(NULL));

    int r = rand() % 3;

    char FileName[50] = "C:\\Users\\86153\\Desktop\\SAT\\ans\\ans1.txt";

    if(r == 1) strcpy(FileName, "C:\\Users\\86153\\Desktop\\SAT\\ans\\ans2.txt");

    else if(r == 2) strcpy(FileName, "C:\\Users\\86153\\Desktop\\SAT\\ans\\ans3.txt");//三种填充合法布局

    //对棋盘进行初始化

    char a[ROW][COL], b[ROW][COL];

    numNode c[10][10], d;

    FILE \* fp;

    fp = fopen(FileName, "r");

    ReadSatissudo(a, fp);

    CreatMod(a, b, holes);

    show(b);

    ans = (int\*)malloc(sizeof(int) \* (boolCnt + 1));

    int resofhani;

    int Hcnt = 0;

    while(1)

    {

        show(b);

        TurnToInt(b, c);

        int x, y, z;

        scanf("%d%d%d", &x, &y, &z);

        c[x][y].x = x;  c[x][y].y = y;  c[x][y].v = z;

        AddNode(b, x, y, z);

        Hcnt ++;

        printf("----是否进行可行性验证?\n 1 -> y      0 -> n\n");

        printf("注意：可行性验证每局游戏只能进行一次！！！\n");

        int ifcheck;

        scanf("%d", &ifcheck);

        if(ifcheck)

        {

            int flag\_cnf = 0;

            char CnfFileName[100];

                strcpy(CnfFileName, "C:\\Users\\86153\\Desktop\\SAT\\ans\\pretreat.txt");

                ToCnf(c, CnfFileName, holes - Hcnt);

            strcpy(fileName, "C:\\Users\\86153\\Desktop\\SAT\\ans\\pretreat.txt");

            if (ReadCNF(CNF, SAT) == OK)

                flag\_cnf = 0;

            if (flag\_cnf)

                printf("尚未未读取cnf文件!\n ");

            else

            {

                int res;

                branchRule = 5;

                ans = (int\*)malloc(sizeof(int) \* (boolCnt + 1));

                for (int i = 0; i <= boolCnt; i++)

                    ans[i] = 1;

                res = DPLL(CNF, ans, SAT);

                printf("OVERDPLL\n");

                if (!res)

                {

                    resofhani = ERROR;

                    printf("不可满足！\n");

                }

                else

                {

                    resofhani = OK;

                    printf("可满足！\n");

                }

            }

        if(resofhani == ERROR)

        {

            printf("---请注意，当前hanidoku无解！，请重新填充");

        }

        }

        if(Hcnt == holes)

        {

            printf("----恭喜！填充完成！\n");

            break;

        }

        printf("\n------------------PLEASE 按回车键继续-----------------\n");

                    getchar(), getchar();

    }

    return ;

}

**display.cpp**

#include"define.cpp"

#include"cnfparser.cpp"

#include"solver.cpp"

#include"hanidoku.cpp"

int main()

{

    int op = 1, op1 = 1, op2 = 1, flag\_cnf, difficulty;

    clock\_t start, finish;

    double time1, time2, rate;

    bool res;

    while (op)

    {

        system("cls");

        printf("\n----------------------------->> MENUE <<----------------------------------\n");

        printf("\n                     1.DPLL     2.Hanidoku    0.退出                       \n\n");

        printf("--------------------------------------------------------------------------\n>>");

        scanf("%d", &op);

        switch (op)

        {

        case 1:

            flag\_cnf = 1, op1 = 1;

            while (op1)

            {

                system("cls");

                printf("\n--------------------------------\*\*\* SAT Solver \*\*\*--------------------------------\n\n");

                printf("\n------------------------------>> OPTIONS <<---------------------------------------\n");

                printf("\t1.读取文件             2.打印文件\n");

                printf("\t3.DPLL                 4.打印并保存结果\n");

                printf("\t0.返回 \n");

                printf("-------------------------------------------------------------------------\n>>");

                scanf("%d", &op1);

                switch (op1)

                {

                case 1:

                    printf("----是否读取默认路径文件？\n");

                    printf("    1 -> [y] / 2 -> [n] \n");

                    int  DefaultOrNot;

                    scanf("%d", &DefaultOrNot);

                    if(DefaultOrNot == 2 )

                    {

                        printf("请输入测试文件地址:\n");

                        scanf("%s", fileName);

                    }

                    else strcpy(fileName, "C:\\Users\\86153\\Desktop\\SAT\\test\\基准算例\\功能测试\\sat-20.cnf");

                    if (ReadCNF(CNF, SAT) == OK)

                        flag\_cnf = 0, printf("cnf文件读取完毕！\n");

                    else

                        printf("cnf文件读取失败!\n");

                    getchar();

                    printf("\n------------------PLEASE 按回车键继续-----------------\n");

                    getchar();

                    break;

                case 2:

                    if (flag\_cnf)

                        printf("尚未未读取cnf文件!\n ");

                    else

                    {

                        printf("文件中有如下子句：\n");

                        printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

                        for (clause\* lp = CNF; lp; lp = lp->next)

                        {

                            for (node\* tp = lp->head; tp; tp = tp->next)

                                printf("%d ", tp->value);

                            printf("\n");

                        }

                    }

                    printf("\n------------------PLEASE 按回车键继续-----------------\n");

                    getchar(), getchar();

                    break;

                case 3:

                    if (flag\_cnf)

                        printf("尚未未读取cnf文件!\n ");

                    else

                    {

                        branchRule = 5;

                        ans = (int\*)malloc(sizeof(int) \* (boolCnt + 1));

                        for (int i = 0; i <= boolCnt; i++)

                            ans[i] = 1;

                        start = clock();

                        res = DPLL(CNF, ans, SAT);

                        printf("OVERDPLL\n");

                        finish = clock();

                        if (!res)

                            printf("不可满足！\n");

                        else

                        {

                            printf("可满足！\n");

                        }

                        time1 = (double)(finish - start);

                        destroyClause(CNF);

                        ReadCNF(CNF, SAT);

                        branchRule = 1;

                        for (int i = 0; i <= boolCnt; i++)

                            ans[i] = 1;

                        start = clock();

                        res = DPLL(CNF, ans, SAT);

                        printf("OVERDPLL\n");

                        finish = clock();

                        time2 = (double)(finish - start);

                        printf("DPLL过程优化前用时%lfms\n", time1);

                        printf("DPLL过程优化后用时%lfms\n", time2 );

                    }

                    if(time2 == 0) printf("时间优化率为0%%\n");

                    else printf("时间优化率为%lf%%", (100 \* (time1 - time2)) / time1);

                    printf("\n------------------PLEASE 按回车键继续-----------------\n");

                    getchar(), getchar();

                    break;

                case 4:

                    for (int i = 1; i <= boolCnt; i++)

                    {

                        if (ans[i])

                            printf("%d \n", i);

                        else

                            printf("%d \n", -i);

                    }

                    if (flag\_cnf)

                        printf("尚未未读取cnf文件!\n ");

                    else

                    {

                        printf("----是否保存到默认路径？\n");

                        printf("    1 -> [y] / 2 -> [n] \n");

                        int  DefaultOrNot;

                        scanf("%d", &DefaultOrNot);

                        if(DefaultOrNot == 2 )

                        {

                            printf("请输入保存文件地址:\n");

                            scanf("%s", fileName);

                        }

                        else strcpy(fileName, "C:\\Users\\86153\\Desktop\\SAT\\save");

                        if (printCNF(res, ans, time1, time2))

                            printf("结果已保存\n");

                        else

                            printf("结果保存失败!\n");

                    }

                    printf("\n------------------PLEASE 按回车键继续-----------------\n");

                    getchar(), getchar();

                    break;

                case 0:

                    break;

                }

            }

            break;

        case 2:

             op2 = 1;

            while (op2)

            {

                system("cls");

                printf("\n---------------------------\*\*\* Hanidoku Base On DPLL \*\*\*------------------------\n\n");

                printf("\n--------------------------------OPTIONS---------------------------------\n\n");

                printf("                      1.Hanidoku   0.Return                            \n\n");

                printf("------------------------------------------------------------------------\n");

                scanf("%d", &op2);

                switch (op2)

                {

                case 1:

                    hanidoku();

                    printf("\n------------------PLEASE 按回车键继续-----------------\n");

                    getchar(), getchar();

                    break;

                default:

                    break;

                }

            }

            break;

        case 0:

            break;

        }

    }

    return 0;

}