

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 程序设计综合课程设计**

**专业班级： 计算机2003班**

**学 号：**

**姓 名：**

**指导教师： 卢萍**

**报告日期： 2021年9月20日**

**计算机科学与技术学院**

目录

**[任 务 书 2](#_Toc27423)**

[设计内容 2](#_Toc22417)

[设计要求 2](#_Toc16053)

**[1.引 言 4](#_Toc26954)**

[1.1 问题背景 4](#_Toc22843)

[1.2 研究意义 4](#_Toc19273)

[1.3 本文摘要 5](#_Toc15530)

**[2.系统需求分析与总体设计 6](#_Toc10839)**

[2.1系统需求分析 6](#_Toc4759)

[2.2系统总体设计 6](#_Toc8692)

**[3. SAT问题与数独问题思想 7](#_Toc18356)**

[3.1总体构想 7](#_Toc5725)

[3.2 SAT问题思想 7](#_Toc30811)

[3.2.1DPLL算法思想 7](#_Toc32613)

[3.2.2 DPLL改进思路 8](#_Toc24169)

[3.3数独思想 9](#_Toc13161)

**[4. 系统详细设计 12](#_Toc13040)**

[4.1有关常量和变量定义 12](#_Toc31411)

[4.2有关数据结构的定义： 12](#_Toc19088)

[4.3所有函数的定义 13](#_Toc14073)

[4.4一些主要函数的算法设计 14](#_Toc10609)

[4.5程序源代码 19](#_Toc2018)

**[5. 算例测试 20](#_Toc29748)**

[5.1 SAT部分 20](#_Toc23740)

[5.1.1SAT部分操作界面 20](#_Toc3420)

[5.1.2SAT算例测试及其解的正确性验证 20](#_Toc9408)

[5.2数独部分 22](#_Toc27383)

[5.2.1数独部分操作界面 22](#_Toc6810)

[5.2.2生成数独功能测试 22](#_Toc20110)

[5.3分析 23](#_Toc19726)

[5.3.1SAT部分分析 23](#_Toc6569)

[5.3.2数独部分分析 23](#_Toc22004)

**[6. 感悟与总结 24](#_Toc2926)**

**[7. 致谢 错误！未定义书签。](#_Toc31200)**

**[参考文献 25](#_Toc22553)**

**[附录 26](#_Toc32301)**

# 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问题求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

# 

# 1 引 言

## 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。

## 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。

## 本文摘要

依据Davis和Putnam在1960年提出的DPLL算法来求解合取范式，并基于原始的DPLL算法提出优化方案，做出一些思考。本文首先介绍关于SAT问题的背景、研究意义等，接着介绍系统需求与总体的设计，紧接着对系统的思想进行简要说明，包括SAT问题的思想与数独问题的思想，其中SAT问题思想包括未改近的DPLL算法和改进之后的DPLL算法思路进行介绍，接着是系统详细设计，包括有关常量、数据结构、函数及函数主要思想的介绍，还有算例测试，包括SAT与数独的测试最后是感悟与分析以及致谢。

# 

# 

# 2.系统需求分析与总体设计

## 2.1系统需求分析

本系统致力于实现两大功能，一是关于cnf文件的读取与SAT问题的求解，二是有关数独游戏的设计，实现具有一定交互性的数独游戏。

## 2.2系统总体设计

本系统分为两大模块，第一模块为数独部分，第二模块为SAT部分，数独部分又分为4个模块，分别为：生成数独、开始游戏、查看答案、利用DPLL求解数独，其中生成数独实现随机生成一个9\*9的数独的功能，开始游戏实现进行填写数字并判断所填写数字正确与否的功能，查看答案实现查看数独答案的功能，利用DPLL求解数独实现利用DPLL算法求解由该数独转化成的SAT问题。SAT部分为对于cnf文件进行读取，输出cnf文件的数据，随后求解相关的SAT问题并输出该SAT问题的解与求解时间。其模块结构图如图2-1所示：



图2-1程序模块结构图

# **3. SAT问题与数独问题思想**

## 3.1总体构想

程序所有函数中关于求解SAT问题的函数有InitSat、Sat、LoadCnf、GetNum、DPLL、PrintAnswer、PutClause、Deduce、SingleClauseDeduce、NextBranch、Analyse\_conflict、ChectAnswer。其中InitSat用于数据元素类型等的初始化、Sat用于调用函数、LoadCnf用于cnf文件的调用、GetNum以读取字符的形式得到cnf文件的内容如变元数子句数等、DPLL用于求解SAT问题的框架、PrintAnswer是用来打印和保存结果的、PutClause用于添加子句、Deduce和SingleClauseDeduce类似于一颗树和树上的节点便于回溯、NextBranch是确定下一分支变量、Analyse\_conflict、ChectAnswer是用来回溯的。

关于数独游戏的函数主要有Sudoku、NewSudoku、GenerateSudoku、DigHole、SolveSudoku。其中Sudoku用于函数调用、NewSudoku 和GenerateSudoku用于生成新的数独、DigHole用于挖洞、SolveSudoku用于将数独转化为cnf文件并求解。

## 3.2 SAT问题思想

**3.2.1DPLL算法思想**

DPLL算法是一种基于树的回溯算法，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.

根据上述规则可不断对公式化简，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3-1所示。



图3-1 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 ),为了优化执行效率，一般用非递归实现。

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);

return DPLL(S ∪¬v);

}

**3.2.2 DPLL改进思路**

我们首先假定，在初始状态下，不存在已经被赋值的变量，我们称未被赋值的变量为自由变量。主循环通过给一个自由变量赋值，从而进入一个分支，这种对变量进行赋值的操作被称为判定，这个被赋值的变量也就有了一个与之对应的判定级。判定级从1开始，随之后的分支变量出现而增长。这些操作由迭代算法中的 NextBranch()方法完成。

进入分支后，问题可以简化为由这一判定及其推论决定。deduce()方法完成一些必要的推理，以便得出在当前的判定级下，为了使被判定实例满足而需要进行的变量赋值。在分支后由该分支判定的推论得到的被赋值变量，被定义为与该分支变量处于同一判定级。

推理结束后，如果所有变量已经被赋值且所有子句都被满足，则返回 SATISFIABLE：如果在推理后的赋值状态下，存在矛盾子句，则当前判定分支及其子分支都不可能是能够使实例被满足的变量赋值，这时求解器会进行回退操作。回退到0判定级则表示即使未进入任何一个判定分支，该输入实例仍然是不可满足的，在这种情况下，求解器将给出该被求解实例不可满足的结论。其中回溯操作回退到上一判定级，如果选定的赋值变量的两个文字均已经推理完毕，则再次回退到上一判定级。

迭代的DPLL算法中的NextBranch()方法决定下一个分支方向。文字统计启发式算法，该算法对给定的变量在两种赋值状态(true或false)下，会导致的未被满足子句数量进行统计。在实际应用中，这种启发式算法，即通过计算同一变量文字的不同逻辑值的动态最大组合总和来选定分支变量的启发式算法，在标准测试数据上可以得到很好的结果。

关于NextBranch()方法的具体算法描述如下

(1)每一变量的两个文字(true和false)都对应一个计数器，它们的初始值为0。

(2)当一个子句被加入到子句库中时，子句中每个文字对应的计数器值增加。

(3)每次判定时，拥有最高计数器值的未被赋值变量及它的分支逻辑值会被选出作为下一个分支

(4)默认情况下，这种数值会被随机破坏。这一点是可以改变默认配置的。

(5)所有计数器的值会被周期性地减少一个常量。

**3.3数独思想**

数独游戏要求在9×9的网格中每个单元（cell）填入1至9的一个数字，必须满足三种约束：每一行、每一列及9个3×3的盒子中的数字都不重复。一个数独游戏初始时已经提供了一些提示数，如图3-2中的左图，要求在剩下的空格中填满数字。初始游戏格局要求只有唯一解（一般至少要有17个提示数），基于推理可以求解。如何生成一个有效的数独游戏格局？本程序采用设计一种算法自动生成，采用从完整合法填充开始，基于挖洞法生成。

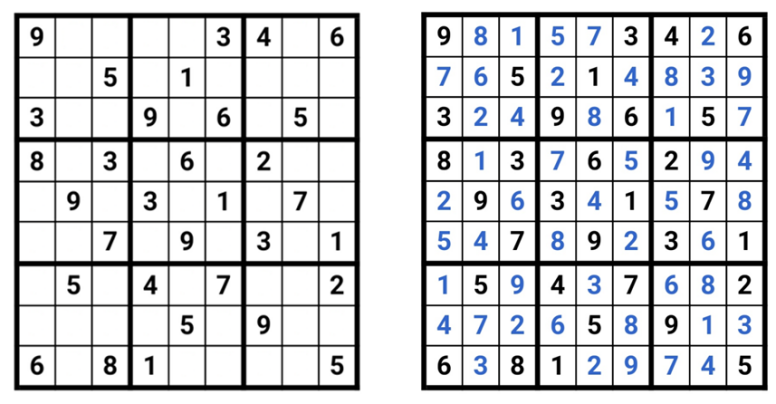


图3-2一个数独格局及其解

本程序利用DPLL求解算法对数独游戏进行求解，因此首先必须将数独游戏转化（归约）为SAT问题，并把它表示为CNF公式的形式。这里要考虑三个问题：（1）如何定义问题的BOOL变元？（2）如何用CNF的子句集表示数独游戏的三种约束？（3）如何表示游戏格局中的提示数条件？

变元可按语义编码为1～9之间数字构成的三位整数*ijk*，*i*, *j*, *k*∈{1,2,…,9}，其中*i*表示单元格的行号，*j*表示单元格的列号，*k*表示单元格<*i*, *j*>填入的数字为*k*。如163变元表示第1行6列填入3；负文字 -452表示第4行5列不填入2。这样编码共有729个变元，10287个子句。

数独游戏的基本要求是：每个单元格只能填入唯一一个数字。以单元格<1,1>例，这可以表示为如下子句：

111 112 113 114 115 116 117 118 119 0

-111 -112 0

-111 -113 0

……

-118 -119 0

上述表示中，每个子句的末尾的0表示结束标记；第一个子句的含义是单元格<1,1>可填入至少一个数字；后面的子句集共同表示只能填入一个数字，子句-111 -112 0表示不能同时填1与2；其它类推。按这种方式需要对81个单元格进行类似表示，得到对应的子句集。

行约束要求每行需要填入1～9中的每个数字，且每个数字只出现一次。以第1行为例可表示为（此处在每个子句后加入注释，说明子句的含义）：

111 121 131 141 151 161 171 181 191 0 第1行含有1

112 122 132 142 152 162 172 182 192 0 第1行含有2

… …

119 129 139 149 159 169 179 189 199 0 第1行含有9

-111 -121 0 前两格不同时为1

-111 -131 0 第1与第3格不同时为1

… …

-111 -191 0 第1与第9格不同时为1

… …

列约束仿照行约束易于表示为对应子句集，可写出。

对于3×3的盒子约束，以左上角的盒子为例进行说明，其子句集可表示如下：

111 121 131 211 221 231 311 321 331 0 包含1

112 122 132 212 222 232 312 322 332 0  包含2

… …

119 129 139 219 229 239 319 329 339 0 包含9

-111 -211 0 11格与21格不同时为1

-111 -311 0 11格与31格不同时为1

-111 -121 0 11格与12格不同时为1

… …

最后，对于每个具体的数独游戏，已经填入了部分提示数，如图2.3中的左图，每个提示数可表示为一个单子句，如第2行3列填入5，对应单子句如下：

235 0

SAT公式CNF文件中，一般变元是从1进行连续编码的，可以将上述语义编码转换为自然顺序编码，公式为：*ijn* → (*i*-1)\*81+(*j*-1)\*9+*n*；当按自然编码对数独游戏对应的CNF公式求解后，设计逆变换公式将解解析为对应的游戏填充方案，完成填充。

# 

# 4. 系统详细设计

## 4.1有关常量和变量定义

有关常量的定义：

#define MaxNumVar 4000

#define PreassignVar 18

//预先分配的变元数目

#define CONFLICT 0

#define SATISFIABLE 1

#define UNSATISFIABLE 0

#define OTHERS 2

#define SINGLE -1

//UNKNOWN表示该变元未知，NONE表示该变元不存在，FALSE表示该变元为假，TRUE表示该变元为真

#define FALSE -1

#define TRUE 1

#define UNKNOWN 0

#define NONE 2

有关变量的定义：

int numVar;

int knownVar;

int numBranch;

int firstBranch[MaxNumVar];

int position\_number;

int sudokuanswer[9][9];

## 4.2有关数据结构的定义：

该程序数据结构定义及注释如下：

//该结构参考双文字监视方法，可看作两张表格

typedef struct varWatch { //文字邻接表，正文字邻接表，负文字邻接表

struct varList \*pos;

struct varList \*neg;

} Var\_watch;

typedef struct varList {

struct clause \*p; //指向一个子句

struct varList \*next; //指向下一个包含该文字的子句

} VarList;

typedef struct clause {

struct clauseLiteral \*p; //指向一个子句中的下一个文字

struct clause \*nextClause;//这个在邻接表中无意义，要从读取CNF文件时看出效果

} Clause;

typedef struct clauseLiteral {

int data;//文字的值

struct clauseLiteral \*next; //指向子句中的下一个文字

} ClauseLiteral;

typedef struct satAnswer {

int branchLevel[MaxNumVar + 1]; //赋值时的决策树高度

int value[MaxNumVar + 1]; //TRUE or FALSE or UNKNOWN or NONE

int searched[MaxNumVar + 1]; //已被搜索的情况数

int singleClause[MaxNumVar + 1]; //标记是否存在该变量的单子句

} SatAnswer;

## 4.3所有函数的定义

本程序所有函数定义及注释如下：

int InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision);//相关数据初始化

int Sat();//SAT问题操作界面

int LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp);//读取cnf文件

int GetNum(FILE \*fp);//获取数据（快读）

int DPLL(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int op, int firstBranch);//DPLL算法（优化后）

int PrintAnswer(SatAnswer \*answer, int result, char filename[100], int duration);//输出答案并且存档答案为res文件

int PutClause(Clause \*ctemp, int var, Var\_watch var\_watch[]);

int Deduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*root);

int SingleClauseDeduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*\*vp);

int NextBranch(int branchDecision[], SatAnswer \*answer);

int Analyse\_conflict(int \*blevel, int var, SatAnswer \*answer);

int Sudoku();//数独操作界面

int NewSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);//生成新的数独

int GenerateSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]);//生成数独并将数独数据写入sudoku\_rule.cnf文件

int DigHole(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);//挖空法实现数独的生成

int SolveSudoku(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);//利用DPLL算法求解数独问题

int dig\_watch(int sudokuTable[9][9]);//输出数独游戏盘

## 4.4一些主要函数的算法设计

(1)函数名称：InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision)

操作结果：初始化数据元素类型。

设计思想：逐行清空，直到指针为空，其流程图如图3-1所示。



图3-1 InitSat简易流程图

(2)函数名称：LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp)

操作结果：加载cnf文件中的内容，读取变元个数，子句数目等

设计思想：先找到标志cnf文件内容开始的符号“p cnf ”，接着调用函数GetNum，直到文件结尾标志出现。流程图如图3-2所示。



图3-2 LoadCnf简易流程图

(3)函数名称：PrintAnswer(SatAnswer \*answer, int result, char filename[100], int duration);

操作结果：将求解SAT问题的结果打印并保存

设计思想：首先设置文件指针创建一个与源文件同名不同后缀的文件，紧接着根据value值的不同选择打印的值为正数还是负数，直到文件结束。流程图如图3-3所示。



图3-3 PrintAnswer简易流程图

(4)函数名称：NextBranch(int branchDecision[], SatAnswer \*answer)

操作结果： 选出下一个决策变量。

设计思想： 遍历子句，统计每个文字出现次数并记录到对应的计数器数组中。

(5)函数名称：ChectAnswer(Clause \*S, SatAnswer \*answer)

操作结果：检查计算结果是否正确，并将结果保存。

设计思想：传入计算结果和子句结构S，将计算结果代入检查，输出检查结果并创建文本文件用来保存结果。

(6)函数名称：PutClause(Clause \*ctemp,int var, Var\_watch var\_watch[])

操作结果：在文字邻接表最后添加子句。

设计思想：先找到文字邻接表的末尾，传入子句，指针初始化。流程图如图3-4 所示。



图3-4 PutCluse流程图

1. 函数名称：NewSudoku()

操作结果：随机生成新的数独。

设计思想：调用GenerateSudoku函数与DigHole函数。

1. 函数名称：GenerateSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[])

操作结果：生成数独数据并且把数据以cnf形式写入sudoku\_rule.cnf文件中。

设计思想：利用数独的性质在每个位置填写数字，将数据写入sudoku\_rule.cnf文件中，对相关数据进行赋值，并且调用DPLL函数确保所填写数字符合实际情况。

1. 函数名称：DigHole(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9])

操作结果：将GenerateSudoku函数所产生的数独进行随机挖洞，并且将数据写入sudoku\_rule.cnf文件中，此函数确保挖洞后的数独具有唯一解。

设计思想：随机生成挖洞顺序，不断调用DPLL函数进行求解，确保生成的数独具有唯一解，其中挖洞顺序采用随机数的形式生成，当生成的数独具有唯一解后，将数据写入sudoku\_rule.cnf文件中。

1. 函数名称：dig\_watch(int sudokuTable[9][9])

操作结果：输出数独盘。

设计思想：由于数独数据是以数组的形式保存的，因此顺序输出数组数据即可，但是数组数据中有很多0，代表未知数据，因此输出“\_”代替0。

**4.5程序源代码**

程序源代码详见《附录》。

# 

# 5. 算例测试

程序采用简易界面，如图所示：



图5-1 程序主界面

## **5.1 SAT部分**

**5.1.1SAT部分操作界面**

SAT部分操作界面如图5-2所示：

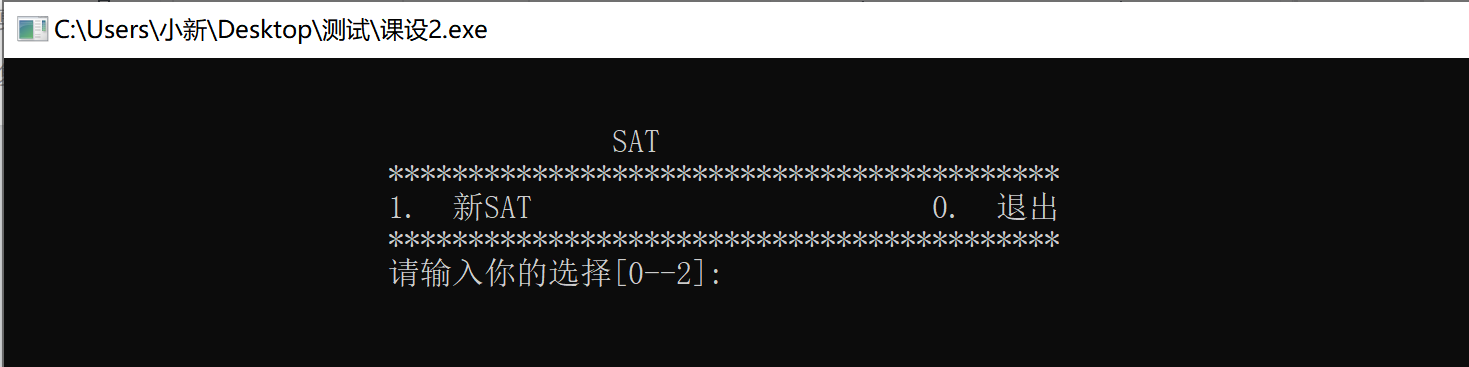
****

图5-2SAT部分操作界面

**5.1.2SAT算例测试及其解的正确性验证**

对于测试集中s级算例，选择了三个算例进行测试，测试数据如表5-1所示：

表5-1s级算例测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 算例名称 | 是否有解 | 求解时间（ms） | 验证解正确性 |
| problem8-50.cnf | 有解 | 0ms | 正确  QQ截图20210911100131 |
| problem9-100.cnf | 有解 | 0ms | 正确  QQ截图20210911100131 |
| problem11-100.cnf | 有解 | 1ms | 正确  QQ截图20210911100131 |

其中三个算例的运行截图分别如图5-3、5-4、5-5所示：

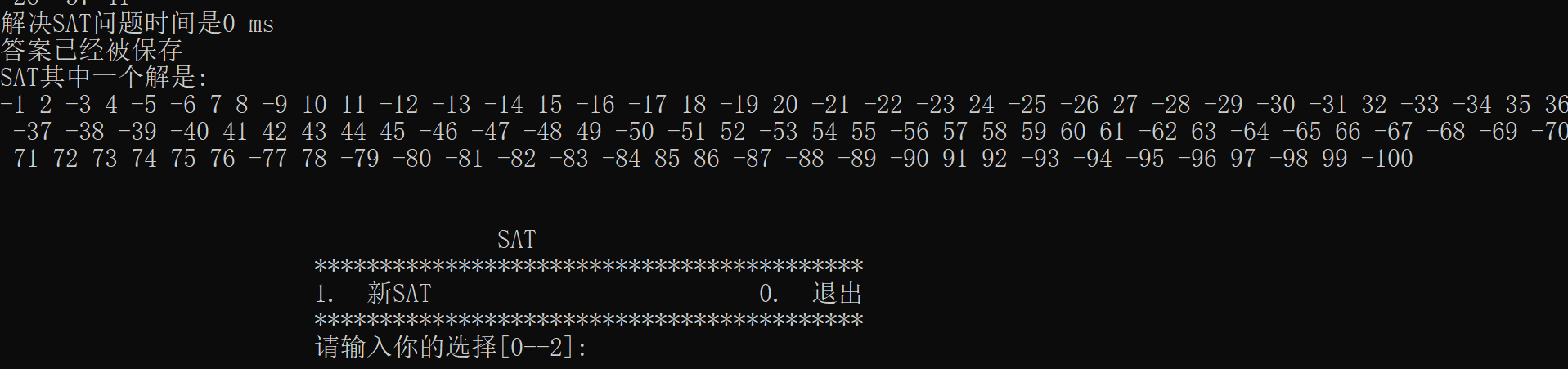


图5-3算例problem8-50.cnf的求解截图

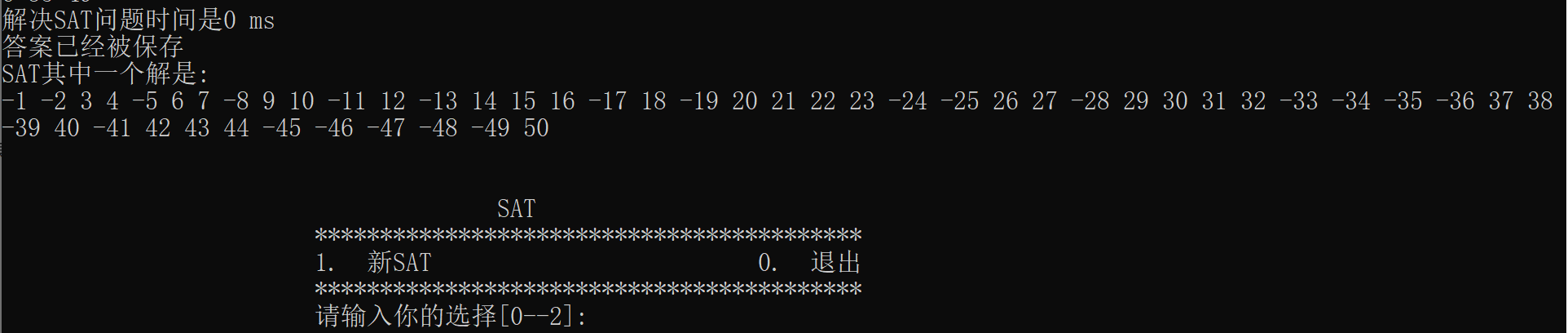


图5-4算例problem9-100.cnf的求解截图

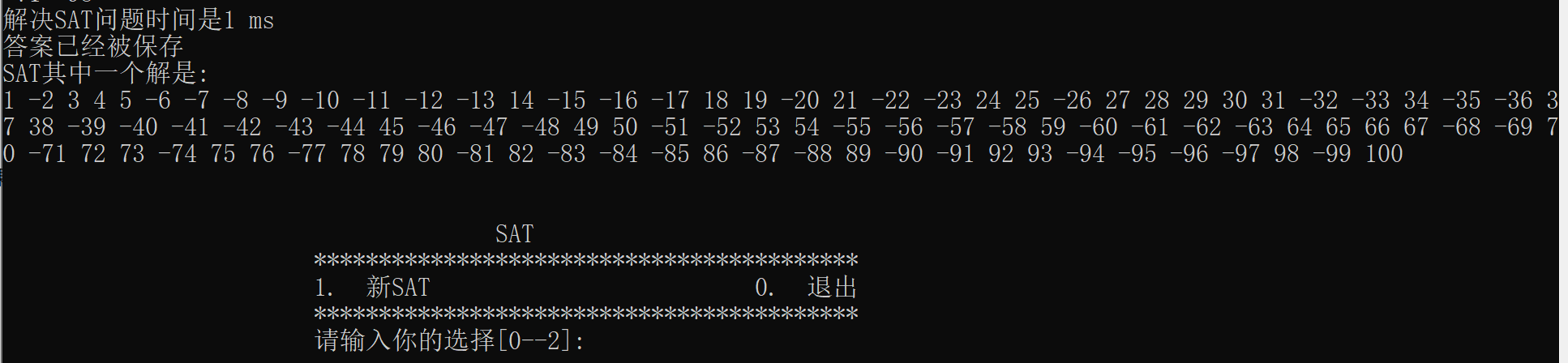


图5-5算例problem11-100.cnf的求解截图

对于测试集中M级算例，选择了三个算例进行测试，测试数据如表5-2所示：

表5-2M级算例测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 算例名称 | 是否有解 | 求解时间 | 验证解正确性 |
| sud00079.cnf | 有解 | 3ms | 正确  QQ截图20210911100131 |
| sud00082.cnf | 有解 | 6ms | 正确  QQ截图20210911100131 |
| sud00861.cnf | 有解 | 1ms | 正确  QQ截图20210911100131 |

其中三个算例的运行截图分别如图5-6、5-7、5-7所示：

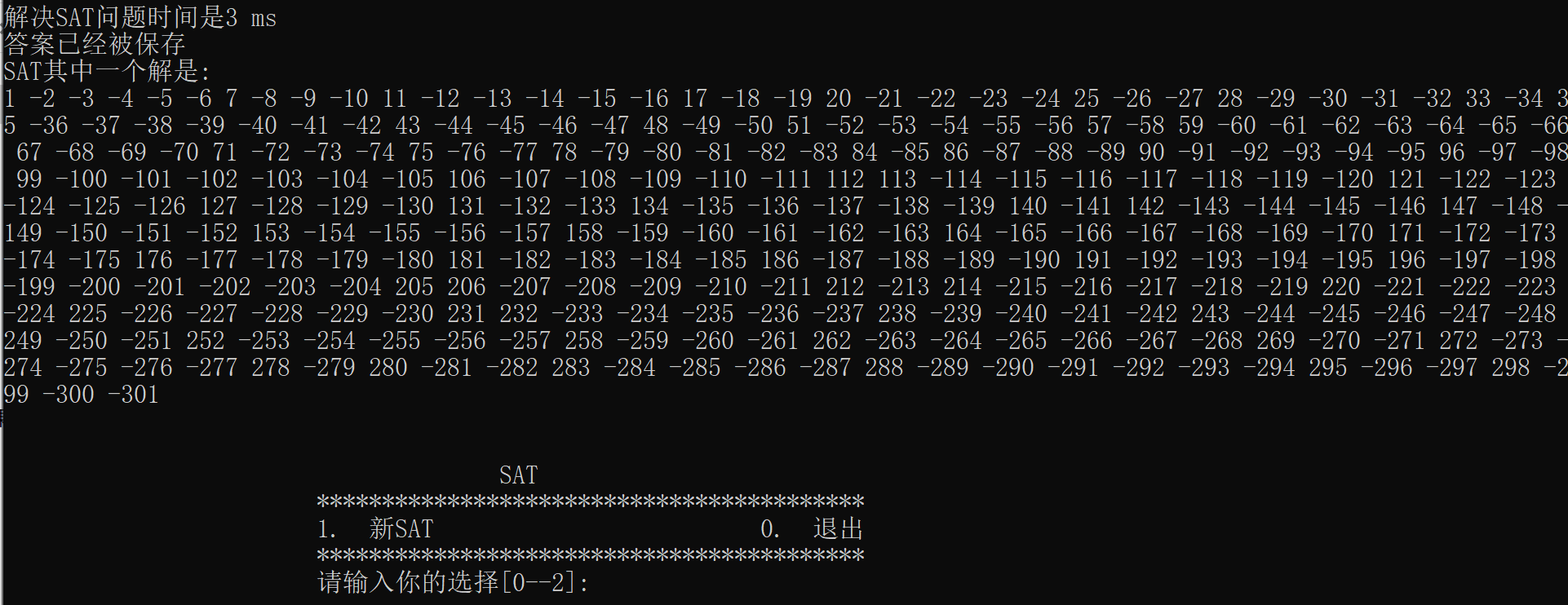


图5-6算例sud00079.cnf的求解截图

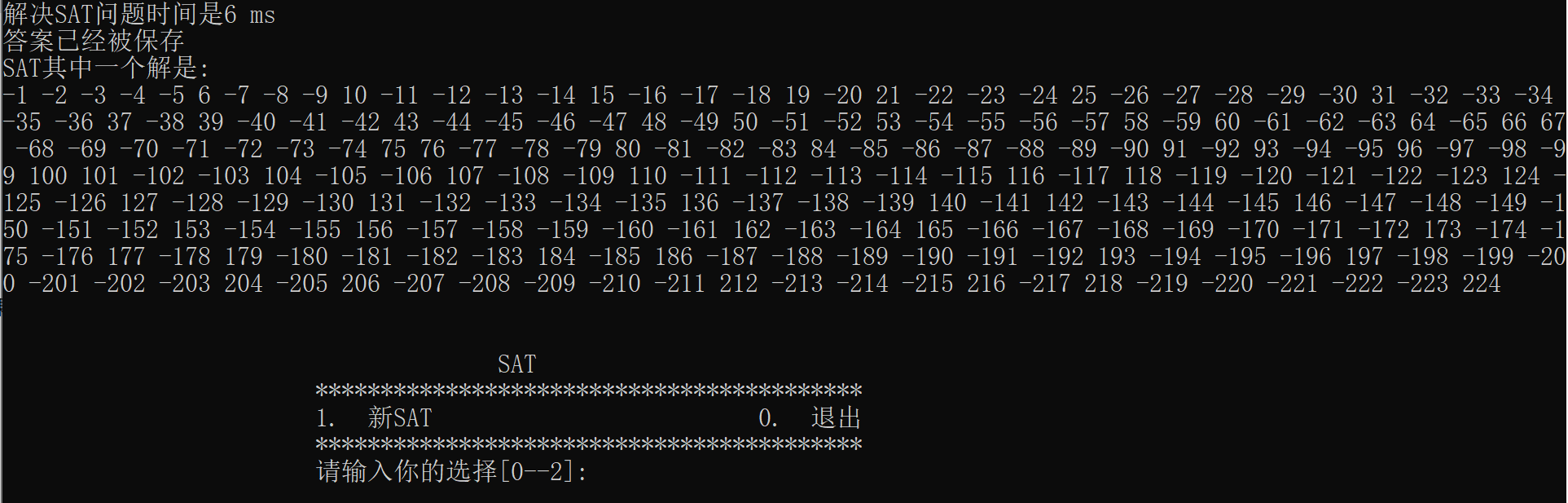


图5-7算例sud00082.cnf的求解截图

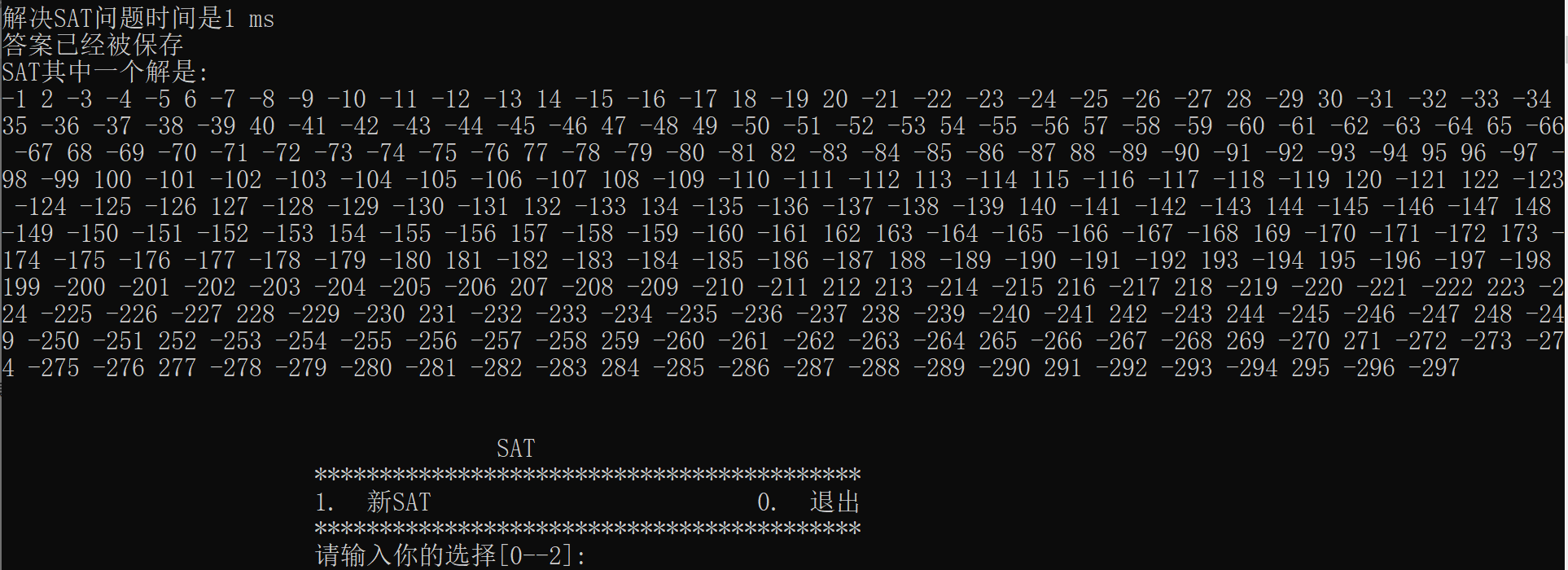


图5-8算例sud00861.cnf的求解截图

## 5.2数独部分

**5.2.1数独部分操作界面**

数独部分操作界面如图5-9所示：

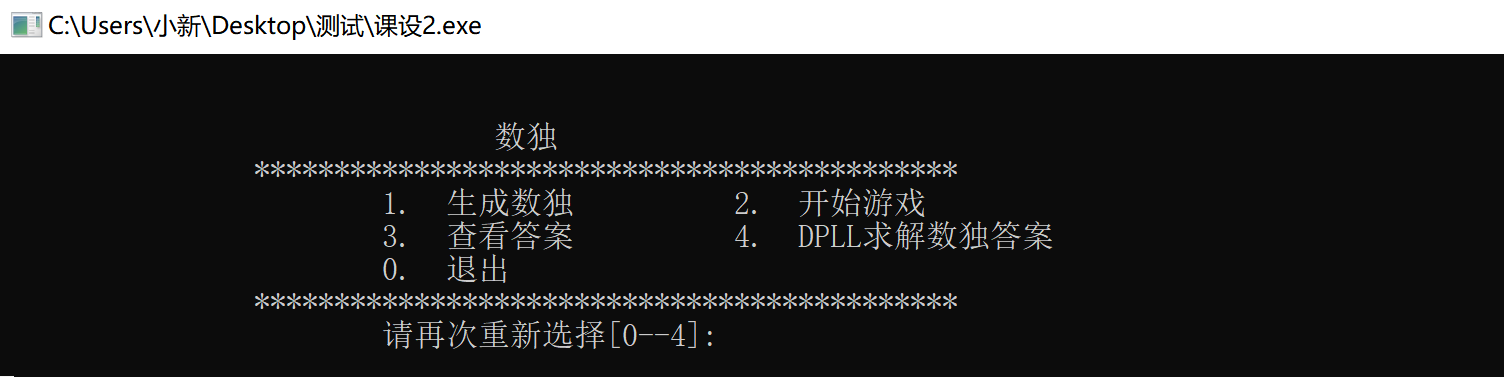


图5-9数独部分操作界面截图

**5.2.2生成数独功能测试**

该功能为生成具有唯一解的数独，并且输出生成该数独的时间，测试三次，统计时间，计算平均时间。测试表如表5-3所示：

表5-3生成数独功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试次数 | 生成的数独截图 | 是否有唯一解 | 时间（ms） |
| 第一次 | QQ截图20210911103017 | 有唯一解 | 29.358s |
| 第二次 | QQ截图20210911103301 | 有唯一解 | 29.852s |
| 第三次 | QQ截图20210911103529 | 有唯一解 | 9.507s |

三次测试生成数独均具有唯一解，平均时间为22.91s。

## 5.3分析

**5.3.1SAT部分分析**

通过上表5-1、5-2可以看出，优化后的DPLL算法对于求解中小规模的SAT问题效率很高，但是我改进后的DPLL算法仍然无法求解大规模的SAT问题，仍然有需要改进的地方。

**5.3.2数独部分分析**

通过表5-3可以看出，对于数独部分，该程序可以生成具有唯一解的数独，并且效率较高，但是美中不足是数独难度固定，无法改变，有进一步改进的空间。而对于数独部分的开始游戏功能，具有一定的交互性，使得该游戏可玩，具有趣味性。

# 6. 感悟与总结

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。

课程设计这门课程只进行了八次，共两个星期，但我觉得这门课的重量不止这些学时，在我看来虽然学时少，但是收获大。这次课设与以往的实验不同，以往都是一个文件就可以搞定，但是这次试验更偏向于实际，注重工程中的模块化编程，这一点是我的收获之一。

我感觉这次的课程设计课程的难度超过以往实验难度。很多人都是从暑假看到题目就开始构思撰写的，我当时在暑假看到题目的时候是真的很懵，看不懂要干嘛，所以只有上网查询资料并且积极与同学交流。在学习的过程中，有一点是必须要具有的，那就是勤于请教，不懂就要问，我在这些天里问了很多人关于课设中遇到的难题，他们也都乐于回答，这使我很感激。

若问收获，我想我最大的收获就是意识到自己代码能力薄弱，需要锻炼吧。此外，我也意识到提升自己的阅读能力至关重要，我也读了很多文献，但是并没有含好的理解，也许是自己的只是储备还不够，需要继续的学习专业的知识。我更加意识到算法的重要性，自己也可以在闲暇时间多多学习算法的相关知识，提升自己的专业能力。孔子说学而不思则罔，思而不学则殆。过去的学习中，我总是拘泥于书本，并没有过多重视实践，这次课设使我变得清醒，毕竟以后的工作也要和代码打交道，代码能力不能是短板。这次课设也使我意识到，学习的过程中需要同伴，互相监督学习，互相鼓励支持，一个人是很难克制自己的，希望以后寻找一些学习上的同伴，共同进步。

最终的课设仍然不算完美，老师的14算例只可以跑出8个，对于数独问题，自己的程序生成的数独难度唯一，不可调，程序仍然可以改进，变得更加完美。

最后，希望自己谨记经验教训，坚持下来，能在平时多多练习自己的编码能力并且提升自己的阅读理解能力以及交流能力。我相信，在日后的学习生活中，有了充足的动力与海量的实践，自己一定会有很大的进步。

# 

# 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>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic forProgramming, 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 InternationalSymposium 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] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

[11] 赵伟楠. 对可满足性（SAT）问题求全解的算法研究及实现. 硕士学位论文. 北京交通大学

# 附录

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MaxNumVar 4000

#define PreassignVar 18

//预先分配的变元数目

#define CONFLICT 0

#define SATISFIABLE 1

#define UNSATISFIABLE 0

#define OTHERS 2

#define SINGLE -1

//UNKNOWN表示该变元未知，NONE表示该变元不存在，FALSE表示该变元为假，TRUE表示该变元为真

#define FALSE -1

#define TRUE 1

#define UNKNOWN 0

#define NONE 2

//该结构参考双文字监视方法，可看作两张表格

typedef struct varWatch {

struct varList \*pos; //文字邻接表，正文字邻接表，负文字邻接表，

struct varList \*neg;

} Var\_watch;

typedef struct varList {

struct clause \*p; //指向一个子句

struct varList \*next; //指向下一个包含该文字的子句

} VarList;

typedef struct clause {

struct clauseLiteral \*p;//指向一个子句中的下一个文字

struct clause \*nextClause;//这个在邻接表中无意义，要从读取CNF文件时看出效果

} Clause;

typedef struct clauseLiteral {

int data;//文字的值

struct clauseLiteral \*next; //指向子句中的下一个文字

} ClauseLiteral;

typedef struct satAnswer {

int branchLevel[MaxNumVar + 1]; //赋值时的决策树高度

int value[MaxNumVar + 1]; //TRUE or FALSE or UNKNOWN or NONE

int searched[MaxNumVar + 1]; //已被搜索的情况数

int singleClause[MaxNumVar + 1]; //标记是否存在该变量的单子句

} SatAnswer;

int InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision);

int Sat();

int LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp);

int GetNum(FILE \*fp);

int DPLL(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int op, int firstBranch);

int PrintAnswer(SatAnswer \*answer, int result, char filename[100], int duration);

int PutClause(Clause \*ctemp, int var, Var\_watch var\_watch[]);

int Deduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*root);

int SingleClauseDeduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*\*vp);

int NextBranch(int branchDecision[], SatAnswer \*answer);

int Analyse\_conflict(int \*blevel, int var, SatAnswer \*answer);

int Sudoku();

int NewSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);

int GenerateSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]);

int DigHole(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);

int SolveSudoku(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);

int dig\_watch(int sudokuTable[9][9]);

int numVar;

int knownVar;

int numBranch;

int firstBranch[MaxNumVar];

int position\_number;

int sudokuanswer[9][9];

int main() {

int op = 1;

while (op) {

printf("\n\n");

printf("\t\t\t 主菜单 \n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. 数独 2. SAT\n");

printf("\t\t\t0. 退出 \n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t请选择你的选择[0--2]: \n");

scanf("%d", &op);

system("cls");

switch (op) {

case 1:

Sudoku();

break;

case 2:

Sat();

break;

case 0:

exit(0);

default:

printf("\t\t\t请再次选择你的选择[0--2]:\n");

scanf("%d", &op);

}

}

return 0;

}

int InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]) {

Clause \*cfront = \*S, \*crear; //子句的前后指针，用于清空

ClauseLiteral \*lfront, \*lrear; //文字的前后指针

//子句初始化

while (cfront) {

crear = cfront->nextClause; //这两行可以理解为一个表格从左上角出发

lfront = cfront->p;

while (lfront) {

lrear = lfront->next;

free(lfront);

lfront = lrear; //逐一清空子句中的文字，清空表格中的一行

}

free(cfront);

cfront = crear; //清除完毕之后，由于free了前指针。。。

}

\*S = NULL;

//解初始化

numVar = 0; //变元数目为0

knownVar = 0; //已知变元数目为0

numBranch = 0; //分支数目为0

\*answer = (SatAnswer \*) malloc(sizeof(SatAnswer)); //给answer结构分配空间

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

//解初始化

(\*answer)->value[i] = NONE; //表示变元不存在

(\*answer)->branchLevel[i] = 0; //赋值时决策树高度为0

(\*answer)->searched[i] = 0; //已被搜索的情况数是0

(\*answer)->singleClause[i] = 0; //该变量的单子句数目为0

//文字邻接表初始化

var\_watch[i].pos = (VarList \*) malloc(sizeof(VarList)); //为正文字邻接表指针分配空间

var\_watch[i].pos->next = NULL; //正文字邻接表指针指向空

var\_watch[i].neg = (VarList \*) malloc(sizeof(VarList)); //为负文字邻接表指针分配空间

var\_watch[i].neg->next = NULL; //负文字邻接表指针指向空

}

//分支决策计数器初始化

for (int j = 1; j <= 2 \* MaxNumVar; ++j)

branchDecision[j] = 0; //分支决策计数器初始化为0

}

int Sat() {

Clause \*S = NULL,\*p=NULL; //子句指针

ClauseLiteral \*q=NULL;

SatAnswer \*answer; //SAT解的指针

Var\_watch var\_watch[MaxNumVar + 1]; //最大变元数目加一个空间

FILE \*fp; //文件指针

char filename[100]; //文件名

int branchDecision[2 \* MaxNumVar + 1]; //决策树最大分支 应该是变元数目的二倍加一（一个变元有两种可能）

int op = 1,result;

clock\_t start, finish; //调用的time.h头文件

int duration; //用于表示耗时

while (op) {

printf("\n\n");

printf("\t\t\t\t SAT \n");

printf("\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. 新SAT 0. 退出\n");

printf("\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t请输入你的选择[0--2]:\n " );

scanf("%d", &op);

system("cls");

switch (op) {

case 1://case1意味着要读取cnf文件，首先要初始化，因此有下一行

InitSat(&S, &answer, var\_watch, branchDecision);

printf("请输入文件路径:\n");

scanf("%s", filename);

fp = fopen(filename, "r");

if (fp == NULL) {

printf("未能打开文件!\n ");

break;

} else LoadCnf(&S, answer, var\_watch, fp); //调用LoadCnf 函数，读取文件

printf("该cnf文件中数据为：\n");

p=S;

for(;p!=NULL;p=p->nextClause){

for(q=p->p;q!=NULL;q=q->next)

printf("%d ",q->data);

printf("\n");

}

start = clock(); //调用time.h，记录开始时间

result = DPLL(answer, var\_watch, branchDecision, 1, 1);

finish = clock(); //调用time.h，记录结束时间

duration = (finish - start);//得到解决问题的耗时

if (result == SATISFIABLE)

{

printf("解决SAT问题时间是%d ms\n", duration);

PrintAnswer(answer, 1, filename, duration);

}

else

{

printf("无解!\n");

}

break;

case 0:

return 0;

default:

printf("\t\t\t请再次输入选择[0~2]:\n");

scanf("%d", &op);

}

}

}

int LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp) {//加载CNF文件

char c;

Clause \*ctemp, \*cp = NULL; //两个指针，加载CNF文件中的子句会用到

ClauseLiteral \*lp, \*ltemp; //两个指针，加载CNF文件中的文字会用到

int var; //变量的值

int numClauseVar; //每个子句的变元数

fscanf(fp, "%c", &c);

while (c == 'c') { //注释段

while (c != '\n' && c != '\r') //

fscanf(fp, "%c", &c); //

fscanf(fp, "%c", &c);

if (c == '\n') //如果读取到换行符，意味着要再重复一次

fscanf(fp, "%c", &c);

}

fscanf(fp, " cnf "); //开始读到文件的信息段

numVar = GetNum(fp); //读取变量数，并赋值给numVar

GetNum(fp); //读取子句数

var = GetNum(fp); //再次调用，读取下一行第一个变量值，把值赋给var

while (1) {

numClauseVar = 0;

ctemp = (Clause \*) malloc(sizeof(Clause)); //为ctemp指针分配空间

lp = ctemp->p; //lp为文字指针，，可以想象一下表格长什么样

while (var) {

++numClauseVar; //计数器功能，统计每个子句的变元数目

if (answer->value[abs(var)] == NONE)

answer->value[abs(var)] = UNKNOWN;

ltemp = (ClauseLiteral \*) malloc(sizeof(ClauseLiteral));

ltemp->data = var; //文字的值域改成var，赋值

ltemp->next = NULL; //

if (numClauseVar == 1) { //储存子句中首个变量

ctemp->p = lp = ltemp;

} else { //储存子句中非首个变量

lp->next = ltemp;

lp = lp->next;

}

if (var > 0)

++firstBranch[var]; //初始分支决策计数增加

else

++firstBranch[numVar - var];

PutClause(ctemp, var, var\_watch); //储存各变量的子句地址

var = GetNum(fp);

}

if (numClauseVar == 1) { //输入单子句，则该子句必须满足，无需存入

answer->value[abs(lp->data)] = lp->data / abs(lp->data);

++knownVar; //已知变元数目加1

} else if (\*S == NULL) {

\*S = cp = ctemp;

cp->nextClause = NULL;

} else { //想想表格的形式，是列的头

cp->nextClause = ctemp;

cp = cp->nextClause;

cp->nextClause = NULL;

}

var = GetNum(fp); //若到达文件尾，再执行一次z'z 读文件操作时，设置文件结束标志

if (feof(fp))

break;

}

}

int GetNum(FILE \*fp) {//这个函数用来读取cnf文件时得到变量数、子句数、后面的内容

char c;

int sign = 1, num = 0; //num 用来得到文字的值,sign用来标记文字是正或者负

fscanf(fp, "%c", &c);

if (c == '-') {

sign = -1; //sign变为-1，表示为负文字

fscanf(fp, "%c", &c);

} else if (c == '0') { //表示该条子句结束

fscanf(fp, "%c", &c);

if (c == '\r') //表示换行

fscanf(fp, "%c", &c);

return num; //

} else if (feof(fp)) //如果是结束标记

return 0;

while (c != ' ' && c != '\n' && c != '\r') {

num = num \* 10 + c - '0'; //得到文字的值

fscanf(fp, "%c", &c);

}

if (c == '\r')

fscanf(fp, "%c", &c);

return sign \* num; //用来得到文字（包括正负和值）

}

int DPLL(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int op, int firstBranch) {

int status, var, blevel = 0; //初始判定级为0

VarList \*vp; //邻接表指针

while (1) {

if (numBranch++ == 0) { //第一次分支决策

if (op == 1) //1:SAT求解、2:生成数独的检验求解

var = NextBranch(branchDecision, answer);

else

var = firstBranch;

} else

var = NextBranch(branchDecision, answer); //下一分支决策

++blevel; //判定级加1

answer->value[abs(var)] = var / abs(var); //进入一分支

answer->branchLevel[abs(var)] = blevel;

++answer->searched[abs(var)]; //已被搜索情况数加1

++knownVar; //已知变元数目加1

while (1) {

if (var > 0)

vp = var\_watch[var].neg->next; //var为TRUE，则搜索var为FALSE的子句

else

vp = var\_watch[-var].pos->next; //var为FALSE，则搜索var为TRUE的子句

status = Deduce(blevel, answer, var\_watch, branchDecision, vp);//单子句传播，返回子句的状态

if (status == SATISFIABLE) //满足的情况

return SATISFIABLE;

else if (status == CONFLICT) {

var = Analyse\_conflict(&blevel, var, answer);//var > 0，矛盾，开始回溯

if (blevel == 0)

return UNSATISFIABLE;

else { //进入另一分支，不满足

answer->value[var] = -answer->value[var];//则值进行反转，正变成负，负变成正

++answer->searched[var]; //被搜索情况数目加1

if (answer->value[var] < 0)

var = -var;

}

} else if (status == OTHERS) break; //已知条件不足，进入下一层

}

}

}

int PrintAnswer(SatAnswer \*answer,int result, char filename[100], int duration) {

FILE \*fp;

int p = 0;

while (filename[p] != 0) p++;

while (filename[p] != '.') p--;

p++;

filename[p] = 'r';

p++;

filename[p] = 'e';

p++;

filename[p] = 's';

p++;

filename[p] = 0;

fp = fopen(filename, "w");

if(result== 1)

fprintf(fp, "s 1\r\n");

else

fprintf(fp, "s 0\r\n");

fprintf(fp, "v ");

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

{

if(answer->value[i] == TRUE)

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

else

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

}

fprintf(fp, "\r\n");

fprintf(fp, "t %d\r\n", duration);

fclose(fp);

printf("答案已经被保存\n");

printf("SAT其中一个解是:\n");

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

if (answer->value[i] == TRUE)

{

printf("%d ", i);

}

else if (answer->value[i] == FALSE)

printf("-%d ", i);

printf("\n");

}

int PutClause(Clause \*ctemp, int var, Var\_watch var\_watch[]) {

VarList \*wp;

if (var > 0) //判断var是否大于零，从而归纳到相应的表格中

wp = var\_watch[var].pos;

else

wp = var\_watch[-var].neg;

while (wp->next)

wp = wp->next; //循环，找到VarList的尾部，将var添加到尾部

wp->next = (VarList \*) malloc(sizeof(VarList));//分配空间

wp = wp->next; //

wp->p = ctemp; //将子句放在末尾

wp->next = NULL;

}

int Deduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*root) {

int top = 0, status; //status表示状态

VarList \*stack[MaxNumVar], \*vp = root; //栈，栈的最大长度为最大变元数目

stack[top++] = vp;

while (top) {

vp = stack[top - 1]; //访问栈顶元素

status = SINGLE;

while (status == SINGLE && vp) { //左子树搜索

status = SingleClauseDeduce(blevel, answer, var\_watch, branchDecision, &vp);

stack[top++] = vp; //左孩子入栈

}

--top; //空指针退栈

if (status == CONFLICT)

return CONFLICT;

if (top) { //右子树搜索

vp = stack[--top]; //根节点出栈

if (vp->next)

stack[top++] = vp->next; //右孩子入栈

}

}

if (knownVar < numVar)

return OTHERS;

else return SATISFIABLE;

}

int SingleClauseDeduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*\*vp) {

Clause \*cp; //子句指针

ClauseLiteral \*lp; //文字指针

int unknownNum, firstUnknown, satisfiable; //

//初始化

unknownNum = 0;

firstUnknown = 0;

satisfiable = 0;

cp = (\*vp)->p; //将cp指向vp所指向的子句

lp = cp->p; //文字指针指向cp所指向的文字

if (lp == NULL)

return OTHERS;

while (lp) {

if (lp->data > 0)

++branchDecision[lp->data]; //分支决策计数增加

else

++branchDecision[numVar - lp->data];

if (answer->value[abs(lp->data)] \* lp->data > 0) { //子句中存在值为TRUE的文字，子句成立

satisfiable = 1;

break;

}

if (answer->value[abs(lp->data)] == UNKNOWN) {

++unknownNum; //计数子句中未被赋值的文字，

if (firstUnknown == 0)

firstUnknown = lp->data; //记录第一个未知的文字

}

lp = lp->next;

}

if (unknownNum == 0 && satisfiable == 0) //该子句文字均已知并且都为FALSE，为矛盾句

return CONFLICT;

else if (unknownNum == 1 && satisfiable == 0) { //该子句无值为TRUE的文字，且只有一个未知文字，为单子句

answer->singleClause[abs(firstUnknown)] = 1; //标记，单子句出现的位置

answer->value[abs(firstUnknown)] = firstUnknown / abs(firstUnknown);

answer->branchLevel[abs(firstUnknown)] = blevel;

++knownVar; //已被赋值的变元数目加1,即确定变元数目加1

if (firstUnknown > 0)

\*vp = var\_watch[firstUnknown].neg->next; //var为TRUE，则检索var为FALSE的子句

else

\*vp = var\_watch[-firstUnknown].pos->next; //var为FALSE，则检索var为TRUE的子句

return SINGLE;

} else if (knownVar < numVar) {

\*vp = NULL;

return OTHERS; //判断条件不足，返回OTHERS

} else return SATISFIABLE;

}

int NextBranch(int branchDecision[], SatAnswer \*answer) {//下一分支函数

int maxVar = numVar, maxCount = 0;

int \*branch;

++numBranch;

branch = numBranch == 1 ? firstBranch : branchDecision;

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

if (i <= numVar && answer->value[i] != UNKNOWN)

continue;

if (i > numVar && answer->value[i - numVar] != UNKNOWN)

continue;

if (maxCount <= \*(branch + i)) {

maxVar = i;

maxCount = \*(branch + i);

}

}

return maxVar > numVar ? numVar - maxVar : maxVar;

}

int Analyse\_conflict(int \*blevel, int var, SatAnswer \*answer) {//回溯函数

int fore = abs(var);

while (\*blevel != 0) {

for (int j = 1; j <= numVar; ++j)

if (j != fore && answer->branchLevel[j] == \*blevel) { //将由var赋值产生的单子句重置

answer->value[j] = UNKNOWN;

answer->branchLevel[j] = 0;

answer->searched[j] = 0;

answer->singleClause[j] = 0;

--knownVar;

}

if (\*blevel != 1) {

if (answer->searched[fore] == 2) { //var的TRUE和FALSE分支均搜索过，进行回溯

--(\*blevel);

answer->value[fore] = UNKNOWN;

answer->branchLevel[fore] = 0;

answer->searched[fore] = 0;

--knownVar;

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

if (answer->branchLevel[i] == \*blevel && answer->singleClause[i] == 0) {

fore = i;

break;

}

} else break; //搜索另一分支

} else if (answer->searched[abs(fore)] == 2)//blevel1全部搜索完

--(\*blevel);

else break; //搜索blevel1的另一分支

}

return fore;

}

int Sudoku() {

Clause \*S = NULL;

SatAnswer \*answer;

FILE \*fp;

char filename[100];

Var\_watch var\_watch[MaxNumVar + 1];

int branchDecision[2 \* MaxNumVar + 1];

int sudokuTable[9][9];

int op = 1,x,y,num,option,flag=1;

clock\_t start, finish;

double duration;

srand((unsigned) time(NULL));

while (op) {

printf("\n\n");

printf("\t\t\t 数独 \n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. 生成数独 2. 开始游戏 \n");

printf("\t\t\t3. 查看答案 4. DPLL求解数独答案\n");

printf("\t\t\t0. 退出 \n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t请再次重新选择[0--4]: \n");

scanf("%d", &op);

system("cls");

switch (op) {

case 1:

start = clock();

printf("正在为您生成数独，请稍等！\n");

NewSudoku(&S, &answer, var\_watch, branchDecision, sudokuTable);

finish = clock();

duration = (double) (finish - start) / 1000.0;

printf("产生数独时间是 %.3f s\n", duration);

break;

case 2:

for(int i=1;i<=81-position\_number;i++)

{

dig\_watch(sudokuTable);

printf("请输入希望填数字的位置（例如（1，1））:\n");

scanf("%d%d",&x,&y);

if(sudokuTable[x-1][y-1]||x<1||x>9||y<1||y>9){

printf("该位置已经有数字了，请重新选择！\n");

scanf("%d%d",&x,&y);

}

printf("该位置数字填为:\n");

scanf("%d",&num);

if(sudokuanswer[x-1][y-1]==num){

printf("所填数字正确，请继续！\n");

sudokuTable[x-1][y-1]=num;

}

else

{

printf("所填数字不正确(还有一次机会)，是否查看答案（输入1查看答案，输入2继续作答,输入3放弃作答）\n");

scanf("%d",&option);

if(option==1){

printf("该处正确答案为:%d",sudokuanswer[x-1][y-1]);

sudokuTable[x-1][y-1]=sudokuanswer[x-1][y-1];

}

else if(option==2){

printf("请输入新的答案！\n");

scanf("%d",&num);

if(sudokuanswer[x-1][y-1]==num){

printf("所填数字正确，请继续！\n");

sudokuTable[x-1][y-1]=num;

}

else {

printf("所填数字错误，没有机会了，该处正确答案为%d\n",sudokuanswer[x-1][y-1]);

sudokuTable[x-1][y-1]=sudokuanswer[x-1][y-1];

}

}

else if(option==3){

flag=0;

break;

}

}

}

if(flag){

printf("恭喜你！完成数独！\n");

dig\_watch(sudokuanswer);

}

else printf("请再接再厉！\n");

break;

case 3:

dig\_watch(sudokuanswer);

break;

case 4:

InitSat(&S, &answer, var\_watch, branchDecision);

fp = fopen("sudoku\_rule.txt", "r");

LoadCnf(&S, answer, var\_watch, fp);

fclose(fp);

start = clock();

SolveSudoku(answer, var\_watch, branchDecision, sudokuTable);

finish = clock();

duration = (double) (finish - start) / 1000.0;

printf("解决数独问题时间是%.3f s\n", duration);

break;

case 0:

return 0;

default:

printf("\t\t\t请再次重新选择[0--4]:\n");

scanf("%d", &op);

}

}

}

int NewSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]) {

if (GenerateSudoku(S, answer, var\_watch, branchDecision) == 0)

return 0;

DigHole(\*answer, var\_watch, branchDecision, sudokuTable);

}

int GenerateSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]) {

int x, y, z, i, j, k, l;

int dig\_order[82], index;

FILE \*fp;

fp = fopen("sudoku\_rule.txt", "w");

fprintf(fp, "p cnf 729 10287\r\n");

for (x = 0; x < 9; ++x)

for (y = 0; y < 9; ++y)

for (z = 1; z <= 8; ++z) {

for (i = z + 1; i <= 9; ++i)

fprintf(fp, "%d %d 0\r\n", -(81 \* x + 9 \* y + z), -(81 \* x + 9 \* y + i));//每个位置，数字1~9至多出现一次

}

for (x = 0; x < 9; ++x)

for (z = 1; z <= 9; ++z)

for (y = 0; y < 8; ++y) {

for (i = y + 1; i < 9; ++i)

fprintf(fp, "%d %d 0\r\n", -(81 \* x + 9 \* y + z), -(81 \* x + 9 \* i + z));//每一行，数字1~9至多出现一次

}

for (y = 0; y < 9; ++y)

for (z = 1; z <= 9; ++z)

for (x = 0; x < 8; ++x) {

for (i = x + 1; i < 9; ++i)

fprintf(fp, "%d %d 0\r\n", -(81 \* x + 9 \* y + z), -(81 \* i + 9 \* y + z));//每一列，数字1~9至多出现一次

}

for (z = 1; z <= 9; ++z)

for (i = 0; i < 3; ++i)

for (j = 0; j < 3; ++j) {

for (x = 0; x < 3; ++x)

for (y = 0; y < 3; ++y)

fprintf(fp, "%d ", 81 \* (3 \* i + x) + 9 \* (3 \* j + y) + z);//数字1~9在每个3×3数独中至少出现一次

fprintf(fp, "0\r\n");

for (x = 0; x < 3; ++x) {

for (y = 0; y < 3; ++y) {

for (k = x + 1; k < 3; ++k)

for (l = 0; l < 3; ++l)

if (l != y)

fprintf(fp, "%d %d 0\r\n", -(81 \* (3 \* i + x) + 9 \* (3 \* j + y) + z),

-(81 \* (3 \* i + k) + 9 \* (3 \* j + l) + z));//数字1~9在每个3×3数独中至多出现一次

}

}

}

fclose(fp);

do {

fp = fopen("sudoku\_rule.txt", "r");

if (fp == NULL) {

printf("Opening sudoku\_rule.txt\" failed.\n ");

return 0;

}

InitSat(S, answer, var\_watch, branchDecision);

LoadCnf(S, \*answer, var\_watch, fp);

fclose(fp);

for (j = 1; j <= 81; ++j)

dig\_order[j] = j;

for (j = 81; j > 1; --j) { //随机生成初始化顺序

index = rand() % j + 1;

if (j != index) {

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

dig\_order[index] = dig\_order[index] ^ dig\_order[j];

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

}

}

for (k = 0; k < 11;) { //在棋盘中随机选11个格子随机填入1~9

x = (dig\_order[j] - 1) / 9;

y = (dig\_order[j] - 1) % 9;

z = rand() % 9 + 1;

for (l = 1; l <= 9; ++l)

if (l == z)

(\*answer)->value[81 \* x + 9 \* y + l] = TRUE;

else

(\*answer)->value[81 \* x + 9 \* y + l] = FALSE;

++k;

}

knownVar = k;

} while (DPLL(\*answer, var\_watch, branchDecision, 2, -(rand() % 729 + 1)) == UNSATISFIABLE);

return 1;

}

int DigHole(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]) {

int x, y, z, i, j, k;

int dig\_order[82], index, dig = 1;

int firstBranch;

FILE \*fp;

clock\_t a, b;

a = clock();

for (i = 1; i <= numVar; ++i) {

answer->branchLevel[i] = 0;

answer->searched[i] = 0;

answer->singleClause[i] = 0;

}

for (x = 0; x < 9; ++x) { //得到终盘结果

for (y = 0; y < 9; ++y) {

for (z = 1; z <= 9; ++z)

if (answer->value[81 \* x + 9 \* y + z] == TRUE) {

sudokuTable[x][y] = z;

sudokuanswer[x][y]=z;

break;

}

}

}

for (j = 1; j <= 81; ++j)

dig\_order[j] = j;

for (j = 81; j > 1; --j) { //随机生成挖洞顺序

index = rand() % j + 1;

if (j != index) {

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

dig\_order[index] = dig\_order[index] ^ dig\_order[j];

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

}

}

for (j = 1; j <= 81 - PreassignVar && dig <= 81;) {

dig\_watch(sudokuTable);

x = (dig\_order[dig] - 1) / 9;

y = (dig\_order[dig++] - 1) % 9;

z = sudokuTable[x][y];

if (z <= 0) //该位置不可挖，寻找下一个位置

continue;

knownVar = 9 \* (81 - j); //已经挖掉j个洞

numBranch = 0;

for (i = 1; i <= 9; ++i) //挖去该位置

answer->value[81 \* x + 9 \* y + i] = UNKNOWN;

if (j < 4) { //挖去个数小于4，解必定唯一

++j;

sudokuTable[x][y] = 0;

for (k = 1; k <= 9; ++k) //挖去该位置

answer->value[81 \* x + 9 \* y + k] = UNKNOWN;

continue;

}

for (i = 1; i <= 9; ++i) { //检测挖去该位置解是否唯一

if (i == z)

continue;

firstBranch = 81 \* x + 9 \* y + i;

answer->searched[81 \* x + 9 \* y + i] = 1; //锁定i的另一分支

if (DPLL(answer, var\_watch, branchDecision, 2, firstBranch) == SATISFIABLE) //挖去该位置有其他解

break;

knownVar = 9 \* (81 - j); //已经挖掉j个洞

numBranch = 0;

for (k = 1; k <= numVar; ++k) { //重置终盘

if (!answer->branchLevel[k]) //决策级为0，为初始化条件，不重置

continue;

answer->value[k] = UNKNOWN;

answer->branchLevel[k] = 0;

answer->searched[k] = 0;

answer->singleClause[k] = 0;

}

}

if (i == 10) { //挖去该位置解仍唯一

++j;

sudokuTable[x][y] = 0;

} else { //挖去该位置解不唯一

if (dig > 81)

break;

sudokuTable[x][y] = -sudokuTable[x][y]; //该位置不可挖去

for (k = 1; k <= numVar; ++k) { //重置终盘

if (!answer->branchLevel[k]) //决策级为0，为初始化条件，不重置

continue;

answer->value[k] = UNKNOWN;

answer->branchLevel[k] = 0;

answer->searched[k] = 0;

answer->singleClause[k] = 0;

}

for (k = 1; k <= 9; ++k) //填入原来的数

if (k == z)

answer->value[81 \* x + 9 \* y + k] = TRUE;

else

answer->value[81 \* x + 9 \* y + k] = FALSE;

}

}

fp = fopen("sudoku\_rule.txt", "a+");

printf("数独被保存至\"sudokuTable.txt\"\n");

printf("有%d个已知数字:\n", 81 - j + 1);

position\_number=81-j+1;

for (x = 0; x < 9; ++x) { //得到生成数独

for (y = 0; y < 9; ++y) {

sudokuTable[x][y] = abs(sudokuTable[x][y]);

if (sudokuTable[x][y] != 0) {

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

if (i != sudokuTable[x][y])

fprintf(fp, "%d 0\r\n", -(81 \* x + 9 \* y + i));

else

fprintf(fp, "%d 0\r\n", 81 \* x + 9 \* y + i);

}

}

if (y != 0 && y % 3 == 0)

printf("| ");

if(sudokuTable[x][y])printf("%d ", sudokuTable[x][y]);

else printf("\_ ");

}

printf("\n");

if (x != 8 && x % 3 == 2)

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

}

fclose(fp);

}

int SolveSudoku(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]) {

int x, y, z;

DPLL(answer, var\_watch, branchDecision, 1, 1);

for (x = 0; x < 9; ++x) { //得到终盘结果

for (y = 0; y < 9; ++y) {

for (z = 1; z <= 9; ++z)

if (answer->value[81 \* x + 9 \* y + z] == TRUE) {

sudokuTable[x][y] = z;

break;

}

}

}

dig\_watch(sudokuTable);

}

int dig\_watch(int sudokuTable[9][9]) {

int x, y;

for (x = 0; x < 9; ++x) { //打印数独答案

for (y = 0; y < 9; ++y) {

sudokuTable[x][y] = abs(sudokuTable[x][y]);

if (y != 0 && y % 3 == 0)

printf("| ");

if(sudokuTable[x][y])printf("%d ", sudokuTable[x][y]);

else printf("\_ ");

}

printf("\n");

if (x != 8 && x % 3 == 2)

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

}

printf("\n\n");

}