

**课程设计报告**

**题目：基于SAT的对角线数独求解程序**

**课程名称：程序设计综合课程设计**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2024年9月20日**

**计算机科学与技术学院**

目录

[1引言 3](#_Toc178414903)

[1.1课题背景与意义 3](#_Toc178414904)

[1.1.1课题背景 3](#_Toc178414905)

[1.1.2课题意义 3](#_Toc178414906)

[1.2国内外研究现状 4](#_Toc178414907)

[1.3课程设计的主要研究工作 4](#_Toc178414908)

[2系统需求分析与总体设计 6](#_Toc178414909)

[2.1系统需求分析 6](#_Toc178414910)

[2.2系统总体设计 6](#_Toc178414911)

[3系统详细设计 8](#_Toc178414912)

[3.1有关数据结构的定义 8](#_Toc178414913)

[3.2 主要算法设计 9](#_Toc178414914)

[3.2.1 DPLL算法思想 9](#_Toc178414915)

[3.2.2 CNF文件处理 10](#_Toc178414916)

[3.2.3 X-Sudoku 10](#_Toc178414917)

[3.2.4 sat求解结果正确性检验算法思想 11](#_Toc178414918)

[3.2.5 数独填空算法思想 12](#_Toc178414919)

[4系统实现与测试 12](#_Toc178414920)

[4.1系统实现 12](#_Toc178414921)

[4.1.1软硬件环境 12](#_Toc178414922)

[4.1.2数据类型 12](#_Toc178414923)

[4.2系统测试 15](#_Toc178414924)

[4.3结果分析 19](#_Toc178414925)

[4.4 复杂度分析 20](#_Toc178414926)

[5总结与展望 21](#_Toc178414927)

[5.1全文总结 21](#_Toc178414928)

[5.2工作展望 21](#_Toc178414929)

[6 体会 22](#_Toc178414930)

[参考文献 23](#_Toc178414931)

[附录一源码 24](#_Toc178414932)

[附录二操作手册 65](#_Toc178414933)

# 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。对于SAT问题的研究从没有停止过，在1997年和2003年，H.Kautz与B.Selman两次列举出SAT搜索面临的挑战性问题，并于2011年和2007年，两度对当时的SAT问题研究现状进行了全面的综述。黄文奇提出的Solar算法在北京第三届SAT问题快速算法比赛中获得第一名。对SAT问题的求解主要有完备算法和不完备算法两大类。不完备算法主要是局部搜索算法，这种算法不能保证一定找到解，但是求解速度快，对于某些SAT问题的求解，局部搜索算法要比很多完备算法更有效。完备算法出现的时间更早，优点是可以正确判断SAT问题的可满足性，在算例无解的情况下可以给出完备的证明。对于求解SAT问题的优化算法主要有启发式算法、冲突子句学习算法、双文字监视法等。

### 1.1.2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

国内多所高校和研究机构在SAT问题和DPLL算法方面有深入研究。包括对算法的改进、优化、应用以及在实际问题中的应用；国外学者在SAT问题和DPLL算法方面的研究起步较早，研究成果丰富，在SAT问题的理论研究、算法改进、求解器开发等方面有深入研究，提出了多种改进的DPLL算法和变种。越来越多的研究者开始与其他学科（如人工智能、运筹学等）合作，推动SAT求解器的应用和发展。国内企业也在积极探索SAT问题的应用，如在芯片设计、系统验证等方SAT问题和DPLL算法在国内外的研究都非常活跃，研究内容涵盖理论研究、算法改进、求解器开发和实际应用等多个方面。总体来看，SAT求解器和DPLL算法的研究正在向更高效、更广泛的应用方向发展，未来有望在更多领域发挥重要作用。

## 1.3课程设计的主要研究工作

1. 了解SAT问题的研究背景和国内外研究现状，总结SAT问题的基本解决方法，学习逻辑和挖洞法生成唯一对角线数独棋盘的理论知识并尝试进行简单的应用。
2. 核心掌握利用DPLL算法解决SAT问题的知识和代码实现，分析总结DPLL算法，并进行优化，计算优化率。
3. 利用代码实现DPLL运算器用于解决SAT问题，并利用已有的.cnf文件进行简单的算例验证
4. 利用挖洞法将对角线数独游戏转化为.cnf文件，并在DPLL算法中进行验证，找产生唯一可解的数独棋盘的方法。

# 2系统需求分析与总体设计

## 2.1系统需求分析

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
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：**将对角线数独游戏问题转化为SAT问题，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。

## 2.2系统总体设计

这部分可根据用户需求，设计和规划一个系统，说明清楚系统应该有哪些功能模块，每个模块做什么。最后给出完整的系统模块结构图。

系统设计分为两个部分，分别为基于DPLL算法的SAT问题求解程序板块和对角线数独游戏交互程序板块：

1. 基于DPLL算法的SAT问题求解器可以分为以下几个小的模块：
2. cnf文件读取
3. cnf文件输出合取范式，若是则输出文件内容
4. dpll算法求解
5. sat问题和算例结果输出
6. 数独游戏交互板块包含以下几个小模块
7. 初始化数独游戏棋盘
8. 数独游戏SAT问题化处理
9. dpll算法结果与数独游戏结果转换
10. 文件读取与输出
11. 挖洞法生成位移棋盘
12. 选择输出结果，输入答案，获取提示

图2.2-1

# 3系统详细设计

## 3.1有关数据结构的定义

实验需要读取cnf文件并提取其中的信息，所以读取cnf文件采用的物理存储结构为邻接表，可以分为存储信息的存储邻接表和用来索引的索引领接表。

主要数据对象有变元或文字、子句、公式等。基本的处理方式是将子句表示为由文字构成的链表；整个公式则是由子句构成的链表，如图3.1-1



图3.1-1

同时对于每个变元的出现次数以及位置，使用另外一个邻接表进行存储，便

于DPLL算法过程中寻找单子句的文字的变元对应的其它文字，主要包括各个变元

对应的文字出现的子句以及文字位置，以及总的信息包括对应文字的总数，如图

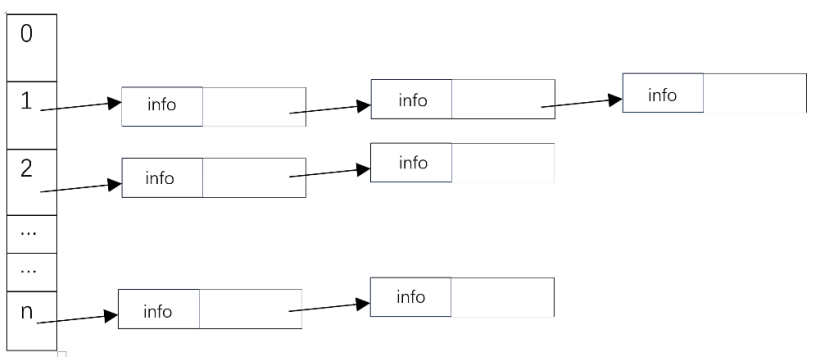


图3.1-2

## 3.2 主要算法设计

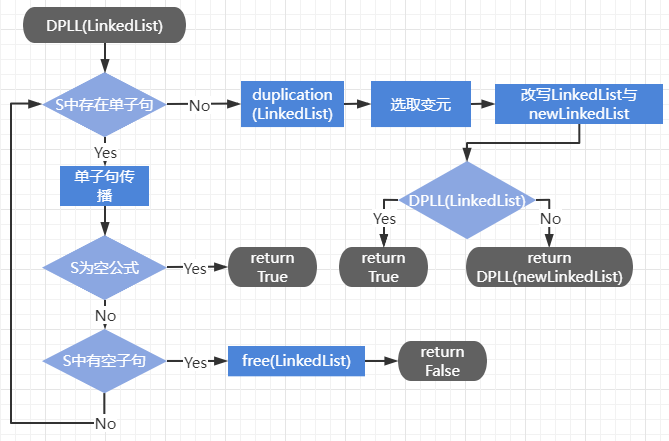
### 3.2.1 DPLL算法思想

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*S*1，如果它是空集，则*S*可满足。否则对*S*1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*S*2。*S*可满足当且仅当*S*2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*S*2；若*L*取假值（即*¬L*成立）时，*S*可化成*S*1.

根据上述规则可不断对公式化简，并最终达到终止状态，如下图所示：



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*);

}

先采取的变元选取策略为选取十字链表的第一个子句的第一个变元，优化后采取的策略是选取最后一个最短子句的第一个变元。

采取的递归分裂策略是复制一个十字链表，加上变元的假值再进行化简。

### 3.2.2 CNF文件处理

程序执行参数的输入，SAT算例cnf文件的读取。建立一个读取CNF文件函数Head\* initCnf(int &varNum, string &filename)，利用ifstream类的构造函数创建一个文件输入流对象，从文件中读取数据，最后将CNF文件中的子句存入十字链表中，返回十字链表表头指针。

### 3.2.3 X-Sudoku

1.初始双数独生成算法：

先生成左上方数独：先通过随机数生成第一行，然后从第二行第一列开始，依次填入。设立标记数组vis[10]，遍历同行、同列、同3x3格子，若有出现i则给vis[i]标记。然后i从1开始遍历到9，当vis[i]未被标记时则将该格子填入数字i。直到第九行第九列填完，9x9数独生成完成。

然后声明第二个9x9数组，并传入重复部分的数据，再用同样的算法生成右下方数独。

2.初始游戏格局生成算法（挖洞法保证唯一解）：

建立一个调用DPLL算法判断挖洞法生成的数独棋盘是否有解的函数void putHole(int a[][COL], int b[][COL], int holeNum)。

设置三个随机变量whichSudoku、x、y来决定挖左上或者右下方数独的哪个洞，挖洞后的格子记为0，若要挖的洞已被挖过，则重新随机。成功挖洞后，首先，从1到9遍历该洞，当满足数独填入规则（行，列，9x9）且不与初始值重复时选取该值，否则跳过。其次，将该值带入要判断是否可以挖洞的位置，然后调用DPLL算法来检验填入该数后数独棋盘是否有解。若有解则恢复当前位置、挖洞数不变，若无解则累计挖洞数加一，当累计挖洞数达到目标挖洞数时，while循环结束，标准双数独游戏格局完成。

3.数独格局转换成对应的SAT问题的CNF文件算法

建立将数独问题转化成SAT问题的主函数string getCnf(int a[][COL], int b[][COL])。

然后分别依据每空已填入的值、每空取值约束、每行取值约束、每列取值约束、每小方块取值约束和公共部分约束共六个部分，将数独问题转换成对应的SAT问题的CNF文件。

SAT公式CNF文件中，一般变元是从1进行连续编码的，可以将语义编码转换为自然顺序编码，公式为：ij n → (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，可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，或给游戏玩家给予每一步填充的正误提示。

### 3.2.4 sat求解结果正确性检验算法思想

为了检验sat所得出答案的正确性，我们需要将所得出的解代回原子句中进行检验。故先将res文件中的解读入到result数组中，再将cnf文件的子句读入数组bv[][]中，且当子句中有变元的值存在于result数组中，将clause\_value[第几子句]标记为true。最后遍历输出clause\_value，若均为真值，说明解析所得答案完全正确。

### 3.2.5 数独填空算法思想

从“sudoku.cnf”读入数独游戏初始格局至数组a,b；从result数组通过逆变公式得到答案数组a1,b1。done[2][10][10]数组记录玩家已填入的位置，方便查阅。玩家写入或撤回答案时改变a,b数组相应位置的值，通过比对玩家已填入的值和答案数组对应位置的值检查答案的正误。

# 4系统实现与测试

## 4.1系统实现

### 4.1.1软硬件环境

系统类型：64 位操作系统, 基于 x64 的处理器

软件环境：Windows11家庭中文版

编辑环境：VSCode+CMake

编译器：gcc version 8.1.0 (x86\_64-win32-seh-rev0, Built by MinGW-W64 project)

### 4.1.2数据类型

typedef **int** status;

typedef **struct** literal\_infomation {*//文字结构*

**int**\* literal\_clause;*//文字在公式中的位置*

**int**\* literal\_clause\_pos;*//文字在子句中的位置*

**int** n\_number;*//个数*

**int** is\_assigned;*//文字是否被赋值*

**int** is\_in\_unit\_clause;*//记录是否在单子句中出现 NO or YES*

}literal\_infomation;

**extern** literal\_infomation literal\_info[MAX\_VARS][2];*//文字存储数组*

typedef **struct** clause\_infomation {*//子句结构*

**int**\* literals;*//子句中的文字*

**int**\* literals\_is\_assigned;*//1 for unassigned 0 for false*

**int** length\_current;*//现在的字句长度*

**int** length\_original;*//最初子句长度*

**int** is\_clause\_satisfied;*//是否满足 NO OR YES true*

**int** unit\_cluase\_literal;*//存储单子句文字中，未赋值文字*

}clause\_infomation;

**extern** clause\_infomation\* clauses;*//子句存储数组*

**extern** **int** original\_formula\_length;*//原长度*

**extern** **int** current\_formula\_length;*//现长度*

typedef **struct** changes\_infomation {*//回溯信息*

**int** index\_of\_clause; *//子句序号*

**int** index\_of\_literal;*//文字序号*

}changes\_infomation;

*//子句满足时，记录子句序号。文字被设为假时，记录文字和字句序号*

**extern** changes\_infomation changes\_stack[8\*MAX\_CLAUSES];*//记录*

**extern** **unsigned** **int** changes\_stack\_index;*//数组changes的下标*

**extern** **unsigned** **int** n\_changes[MAX\_VARS][2];

*//某一层满足或不满足的句子的个数*

typedef **struct** assign\_infomation {*//记录赋值操作*

**int** value; *//赋值类型UNASSIGNED TRUE FALSE*

}assign\_infomation;

**extern** assign\_infomation result[MAX\_VARS];*//用于回溯操作的记录*

**extern** **int** is\_contradicted;*//是否用冲突的单子句*

**extern** **int** conflicting\_literal;*//用于记录冲突的文字*

**extern** **int** unit\_clause\_stack[MAX\_CLAUSES];*//单子句规则*

**extern** **int** n\_unit\_clause\_stack;*//全局栈，用于记录栈顶位置*

**extern** **int** depth;*//DPLL树中，结点的深度*

**extern** **int** n\_vars;*//变元总个数*

**extern** **int** dpll\_called;*//统计DPLL执行次数*

**extern** **int** sudokuterminal[10][10];*//生成数独终盘*

**extern** **int** max\_clause\_len;*//最大字句长度*

typedef **struct** Sudoku {*//数独结构*

**int** s[9][9];

**int** n\_current;

}Sudoku;

**int** Display();

*//展示选项*

FILE**\*** Inputfilename(**char\*** filename);

*//输入文件名*

**int** Is\_Show\_CNF();

*//是否展示cnf文件*

**void** PrintToScreen(**int** value, **double** time);

*//将结果输出到屏幕*

**void** printSudoku(Sudoku**\*** s);

*//打印数独*

**int** CNFparser(FILE**\*** fp);

*//合取范式*

**void** CheckCNF();

*//输出这个合取范式*

**void** Preprocesser();

*//预处理器*

**int** dpll();

*//dpll算法*

**void** Value(**int** v);

*//将文字v设定为已赋值*

**void** UnValue(**int** v);

*//将文字设定为未赋值*

**inline** **int** GetMinLenOfLiteral();

**inline** **int** get\_length\_of();

*//计算最短字句长度*

**void** get\_weight(**int** x, **int** k, **unsigned** **int&** s, **unsigned** **int&** t);

*//获得权重*

**void** PrintToRes(**int** value, **double** time, **char\*** filename);

*//将结果输出到.res文件*

**void** Free();

*//释放动态分配的空间*

**void** startgame();

*//生成数组*

**void** CopytoSudoku(Sudoku**\*** s);

*//将数组复制到数独*

**int** createfinalsudoku(**int** i, **int** j);

*//生成数组*

Sudoku**\*** createSudoku(Sudoku**\*** S, **int** difficulty, **int** flag);

*//创建数独棋盘*

**int** startsolveSudoku(Sudoku**\*** s, **int\*** count);

*//开始解决数独*

**int** solveSudoku(Sudoku**\*** s, **int** x, **int** y, **int\*** count);

*//解决数独的递归过程*

**int** checkkeyword(Sudoku**\*** s, **int** x, **int** y);

*//检查x，y符合的数字*

**inline** **int** Location(**int** x, **int** y, **int** z);

*//三维寻址*

FILE**\*** ToCNFFile(Sudoku**\*** s, **char\*** filename);

*//将数独保存为CNF文件*

Sudoku**\*** ResultToSudoku();

*//检查数独结果，将res转为数度结果*

**void** game(Sudoku**\*** s, Sudoku**\*** final\_patern);

*//开始数独游戏*

## 4.2系统测试

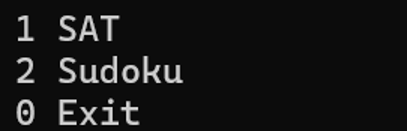
首先叙述一下常用的软件测试方法，在选择几个主要的功能模块（自行掌握数量，关键要体现你的水平的一些模块）描述测试过程。（1）先明确模块的功能、设计目标等；（2）分析、叙述如何选取测试数据，要求有完整的测试大纲；（3）运行结果（这时可用截图）；（4）分析运行结果、确认程序满足该模块的设计目标。

**1.菜单交互界面测试**

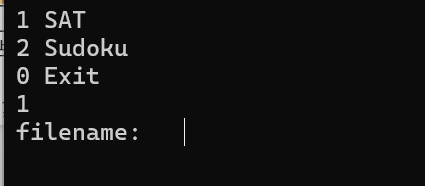
测试主菜单功能和函数模块跳转是否规范化，测试内容为：①输入正确标号。②错误输入无效字符。③输入越界标号。如下表

|  |  |  |
| --- | --- | --- |
| 测试内容 | 测试用例 | 理论结果 |
| ① | 1 | 进入SAT菜单 |
| ② | z | 输入错误 |
| ③ | 3 | 输入错误 |

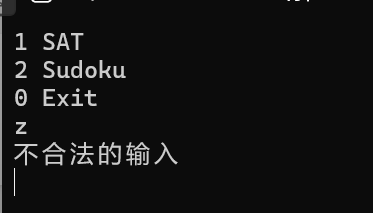
输入1



进入了SAT问题求解板块



输入z，系统提示输入错误



输入3，系统提示输入错误

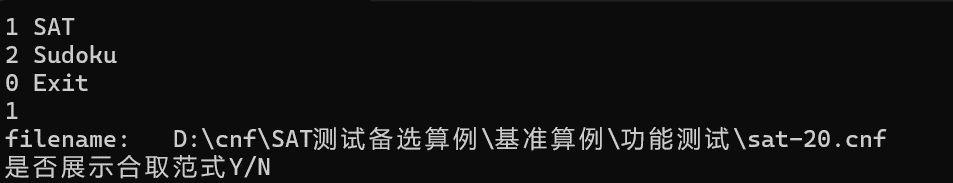


**2.cnf文件算例读取测试**

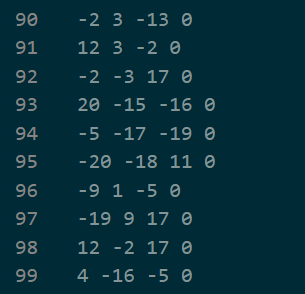
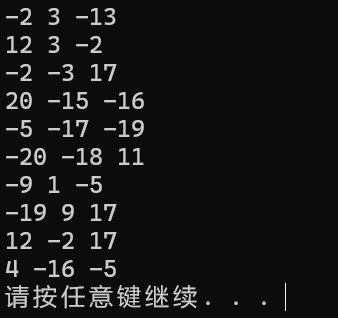
读取cnf测试案例文件，验证结果是否正确。使用样例：sat-20.cnf

|  |  |  |
| --- | --- | --- |
| 测试内容 | 测试用例 | 理论结果 |
| ① | sat-20.cnf | 读取成功 |

输入D:\cnf\SAT测试备选算例\基准算例\功能测试\sat-20.cnf后界面显示结果为：



将读取的cnf文件进行遍历，查看cnf文件读取是否正确，选择后九个变元与子句进行验证

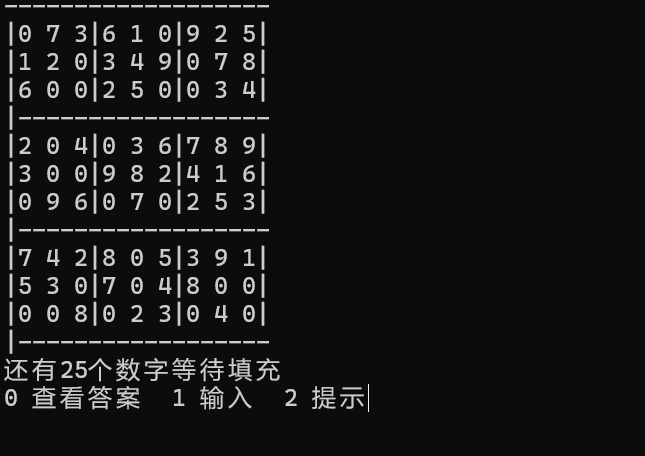
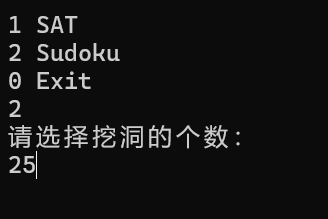


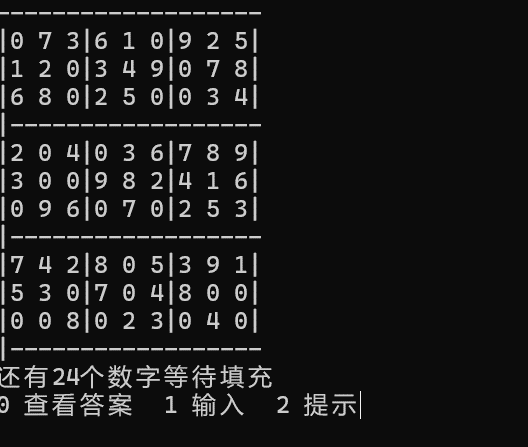
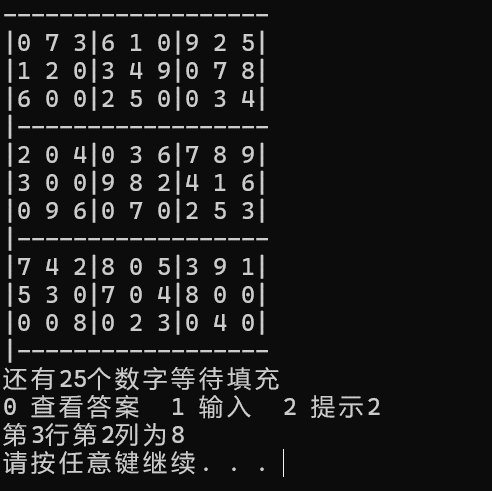
**3.基于DPLL算法的SAT问题求解器的测试：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编号 | 测试用例 | 变元数 | 公式数 | 优化前需要时间（ms） | 优化后需要时间（ms） | 优化率（%） |
| 1 | Sat-20.cnf | 20 | 91 | 0 | 0 | 100 |
| 2 | problem1-20.cnf | 20 | 91 | 2 | 0 | 100 |
| 3 | problem2-50.cnf | 50 | 80 | 1071 | 18 | 98.23 |
| 4 | problem3-100.cnf | 100 | 340 | 16655 | 270 | 98.37 |
| 5 | problem6-50.cnf | 50 | 100 | 625 | 111 | 82.24 |
| 6 | tst\_v10\_c100.cnf | 10 | 100 | 2 | 0 | 100 |
| 7 | sud00001.cnf | 301 | 2780 | 6046 | 18 | 99.7 |
| 8 | sud00009.cnf | 303 | 2851 | 1036 | 5 | 99.52 |
| 9 | sud00012.cnf | 232 | 1901 | 312 | 86 | 72.43 |
| 10 | sud00021.cnf | 308 | 2911 | 1341 | 116 | 91.35 |
| 11 | tst\_v200\_c220.cnf | 200 | 220 | 531 | 17 | 96.80 |
| 12 | u-problem7-50.cnf | 50 | 100 | 1748 | 205 | 88.27 |
| 13 | u-problem10-100.cnf | 100 | 200 | 468940 | 60246 | 87.15 |
| 14 | unsat-5cnf-30.cnf | 30 | 420 | 528 | 90 | 82.95 |
| 15 | ais6.cnf | 61 | 581 | 22 | 12 | 45.45 |
| 16 | ais8.cnf | 113 | 1520 | 870 | 20 | 97.7 |
| 17 | ais10.cnf | 181 | 3151 | 35562 | 4462 | 87.73 |
| 18 | sud00079.cnf | 200 | 220 | 5 | 1 | 80 |

**4.独角线数独游戏测试**

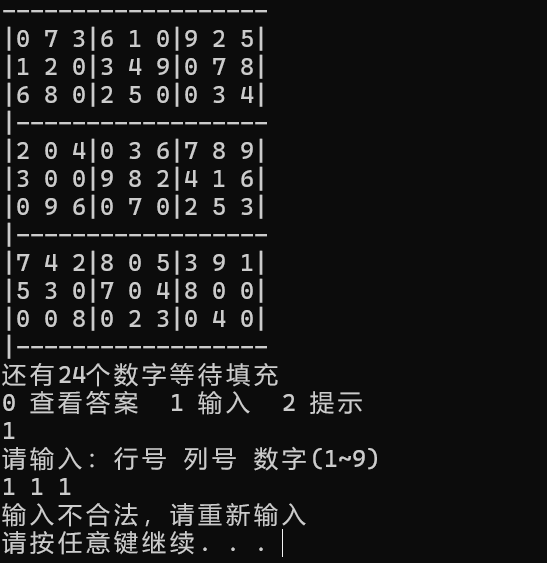
输入挖洞个数（以25为例），然后自动生成棋盘

****

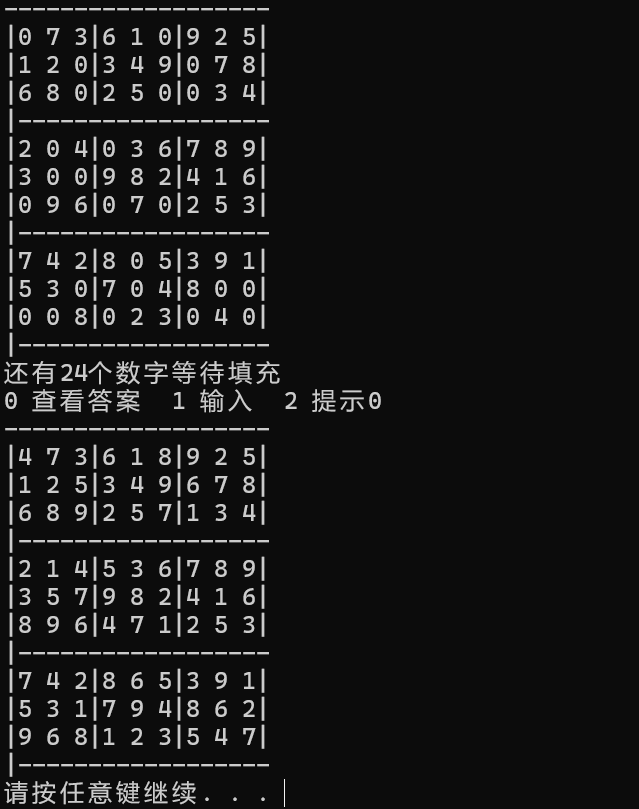


输入1 填入数字

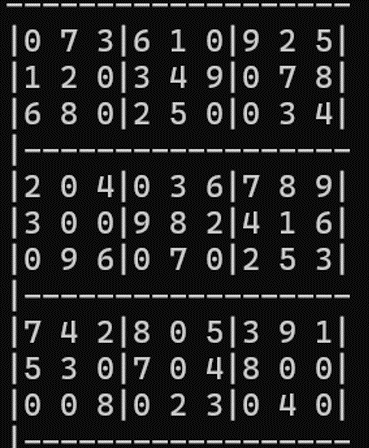
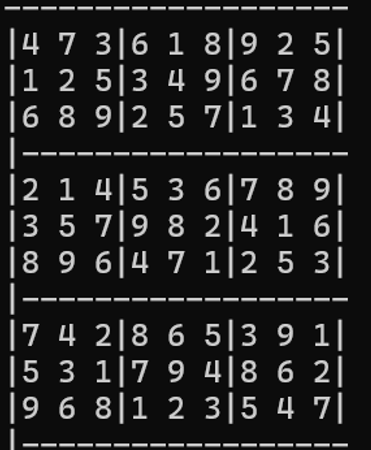
例如 输入1 1 1 在（1,1）的位置填入1 然后判断对错

****

**最后输入0 查看答案，结束游戏**

****

检验答案是否符合 对角线数独规则

****

## 4.3结果分析

（1）对于SAT问题：

通过上面的表格可以大致看出：优化后的算法在小算例中优势并不明显。而随着变元数目与子句数目的增多，优化过后的算法更占优势。当子句的数目不是很多时，优化之后的算法优势不明显，因为优化需要遍历子句，找到最短子句，在这种情况下，子句数目在较小的范围时会出现优化等于没优化，或者优化之后更慢的情况。经检验，该求解器均能在规定时间内求解大、中、小算例，满足该模块的设计目标。不足之处：仍有部分大型算例无法解出。

（2）对于对角线数独游戏：

对于对角线数独游戏的测试，不难发现挖洞数越多，生成初始棋盘的时间较长，那是由于随着挖洞数增加，每挖一个洞生成多解数独的可能性越大，需要进行dpll的次数越多。对dpll进行优化，使得解数独的时间尽可能短。经检验，数独模块完成了预计目标，并有应对不合法输入的策略，满足该模块的设计目标。

（3）不同的算法适应不同的算例。优化之前的算法更适合小型算例，子句与变元数目均不多的情况。优化之后的算法更适合子句长度比较悬殊且变元数目和子句数目较大的情况。

## 4.4 复杂度分析

DPLL；最坏情况为测试这些变量赋值的每种可能的排列方式（如全部赋为True 、其一为 True ，其他全为False ……），若存在一种赋值排列使得公式的结果为True ，那么就可以说明这条公式是可满足的。但很显然，最坏情况下这种方法需要我们测试 2^n 种（n为变量数）赋值排列，而用于检查每种赋值排列最终的运算结果（Examination）也是不可忽略的。故时间复杂度（最坏）O（2^n）。

# 5总结与展望

## 5.1全文总结

主要工作如下：

* 1. 完成SAT求解器的编写并且不断优化；
  2. 优化DPLL算法；
  3. 完成基于SAT的对角线数独游戏求解程序；
  4. 完成SAT求解答案的解析验证程序；
  5. 完成实验报告的撰写。

## 5.2工作展望

在今后的研究中，围绕着如下几个方面开展工作:

1. 解决由于计算机设备问题，在处理大型算例是依旧问题重重的局面，完成DPLL算法在处理大型算例上的不足；
2. 提高菜单和数独交互性， 增加数独游戏的趣味性；
3. 去除冗余设计，完成代码的简化和美化，减少代码中无关无用的数据信息；

# 6 体会

SAT问题求解方法对我来说是一个十分新颖和陌生的问题，在之前也没有接触过该类问题的经历，所以查找文献对我来说就是一个十分有必要的事情。在网上查询文献也是一件十分困难的事，大多数文献并没有在基础层面上讲解这个问题的具体解决思路，也有很多文献是全英，在阅读过程中也造成了很多的困难。

课程设计的结果并不理想，在部分问题上没有取得良好的结果，比如在大型算例上计算机处理十分困难，内存条往往无法承担大型算例的存储要求而导致程序终止。但是在高难度的课程设计下，自身的代码编写能力也取得了十足的长进，能够完成一个简易系统的设计，这相对于之前个人代码编写能力来说是一个巨大的提升。并且也在课程设计过程中了解了很多关于SAT问题的知识，同时也锻炼了动手能力。

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。当然，对dpll算法进行优化也是一个既痛苦又快乐的过程。这次的课程设计不同与以往的是数独改为了对角线数独。因此更加需要我们进行独立思考和算法设计。

通过本次程序设计综合课程设计，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。

# 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

# 附录一源码

**1.main.cpp**

#include "ph.h"

**int** dpll\_called;*//统计DPLL调用次数的全局变量*

**int** sudokuterminal[10][10];*//数独棋盘*

literal\_infomation literal\_info[MAX\_VARS][2];*//文字存储数组*

clause\_infomation\* clauses;*//子句存储数组++++*

**int** original\_formula\_length, current\_formula\_length;*//原长度，现长度*

changes\_infomation changes\_stack[8\*MAX\_CLAUSES];*//记录++++++*

**unsigned** **int** changes\_stack\_index;*//数组changes的下标*

**unsigned** **int** n\_changes[MAX\_VARS][2];*//某一层满足或不满足的句子的个数*

assign\_infomation result[MAX\_VARS];*//用于回溯操作的记录*

**int** is\_contradicted;*//是否用冲突的单子句*

**int** conflicting\_literal;*//用于记录冲突的文字*

**int** unit\_clause\_stack[MAX\_CLAUSES];*//单子句规则++++++*

**int** n\_unit\_clause\_stack;*//全局栈，用于记录栈顶位置*

**int** depth;*//DPLL树中，结点的深度*

**int** n\_vars;*//变元总个数*

**int** max\_clause\_len;*//最长单子句*

**double** t;

**int** main() {

**int** type = 3;

    while (type)

    {

        if (type != 3)

            system("pause");

        type = Display();

        if (type == 1) {

**char** filename[100];*//文件名*

            FILE\* fp = Inputfilename(filename);*//输入文件名*

            if (fp == NULL) {

                printf("文件打开失败\n");

                continue;

            }

**int** is\_cnf = CNFparser(fp);*//创建新的内部结构*

            if (is\_cnf == NO) {

                printf("文件格式不正确\n");

                continue;

            }

            if (Is\_Show\_CNF()) {

                CheckCNF();*//解析公式*

            }

            clock\_t starttime, endtime;

*//printf("开始求解\n");*

            starttime = clock();

            Preprocesser();*//预处理*

**int** value =dpll();

            endtime = clock();

            t = (**double**)(endtime - starttime) \* 1000 / CLOCKS\_PER\_SEC;

            PrintToRes(value,t,filename);

*//PrintToScreen(value,t);*

            Free();

        }

        else if (type == 2) {

            Sudoku\* s = (Sudoku\*)malloc(sizeof(Sudoku));

            Sudoku\* FinalSudoku = NULL;

**char** filename[100] = "Sudoku.cnf";

            startgame();

            CopytoSudoku(s);

**int** flag = ChooseDifficluty();

            s = createSudoku(s,100,flag);*//挖洞法生成数独//第二个数是难度*

            printf("数独生成成功\n");

            FILE\* fp = ToCNFFile(s,filename);

            if (fp == NULL) {

                printf("文件打开失败\n");

                continue;

            }

            fclose(fp);

            fp = fopen(filename, "r");

            if (fp == NULL) {

                printf("文件打开失败\n");

                continue;

            }

            CNFparser(fp);*//创建新的内部结构*

            clock\_t starttime, endtime;

            starttime = clock();

            Preprocesser();*//预处理*

**int** value = dpll();

            endtime = clock();

            t = (**double**)(endtime - starttime) \* 1000/ CLOCKS\_PER\_SEC;

            PrintToRes(value, t, filename);

            if (value == SAT) {*//解出数独*

                FinalSudoku = ResultToSudoku();

                game(s,FinalSudoku);*//*

                free(s);

                free(FinalSudoku);

            }

            else {*//没有接触数独*

                printf("数独没有解\n");

                free(s);

            }

            Free();

        }

    }

    return 0;

}

**2.cnfparser**

#include "ph.h"

#

**int** CNFparser(FILE**\*** fp) {

*//重置全局变量*

*//dpll\_called = 0;  记录dpll调用次数*

    max\_clause\_len = 0;

    for (**int** i = 0; i < MAX\_VARS; i++) {

        literal\_info[i][SATISFIED].is\_assigned = UNASSIGNED;

        literal\_info[i][SATISFIED].is\_in\_unit\_clause = NO;

        literal\_info[i][SATISFIED].literal\_clause = NULL;

        literal\_info[i][SATISFIED].literal\_clause\_pos = NULL;

        literal\_info[i][SATISFIED].n\_number = 0;

        literal\_info[i][SHRUNK].is\_assigned = UNASSIGNED;

        literal\_info[i][SHRUNK].is\_in\_unit\_clause = NO;

        literal\_info[i][SHRUNK].literal\_clause = NULL;

        literal\_info[i][SHRUNK].literal\_clause\_pos = NULL;

        literal\_info[i][SHRUNK].n\_number = 0;

    }

    for (**int** i = 0; i < 8 \* MAX\_CLAUSES; i++) {

        changes\_stack[i].index\_of\_clause = 0;

        changes\_stack[i].index\_of\_literal = 0;

    }

    for (**int** i = 0; i < MAX\_VARS; i++) {

        n\_changes[i][0] = 0;

        n\_changes[i][1] = 0;

    }

    changes\_stack\_index = 0;

    for (**int** i = 0; i < MAX\_VARS; i++) {

        result[i].value = UNASSIGNED;

    }

    is\_contradicted = 0;*//是否用冲突的单子句*

    conflicting\_literal = 0;*//用于记录冲突的文字*

    n\_unit\_clause\_stack = 0;*//全局栈，用于记录栈顶位置*

    depth = 0;*//DPLL树中，结点的深度*

**char** c;

**char** s[5];

**int** x = 1;

    while ((c = fgetc(fp)) == 'c')

        while ((c = fgetc(fp)) != '\n')

            ; *//跳过注释部分*

    if (feof(fp)) {

        fclose(fp);

        return NULL;

    }

    if (c != 'p') {

        fclose(fp);

        return NO;

    }

    fscanf(fp, "%s%d%d", s, &n\_vars, &original\_formula\_length);

    if (strcmp(s, "cnf")) {

        fclose(fp);

        return NO;

    }

    current\_formula\_length = original\_formula\_length;

    clauses = (clause\_infomation\*)malloc(sizeof(clause\_infomation) \* original\_formula\_length);

    for (**int** i = 0; i < original\_formula\_length; i++) {

        clauses[i].literals = NULL;

        clauses[i].length\_current = 0;

        clauses[i].is\_clause\_satisfied = NO;

        clauses[i].literals\_is\_assigned = NULL;

        clauses[i].unit\_cluase\_literal = 0;

        while (1) {

            fscanf(fp, "%d", &x);

            if (x == 0) break;

            clauses[i].literals = (**int**\*)realloc(clauses[i].literals, (++clauses[i].length\_current)\*(sizeof(**int**)));

            clauses[i].literals[clauses[i].length\_current - 1] = x;

            clauses[i].literals\_is\_assigned = (**int**\*)realloc(clauses[i].literals\_is\_assigned, (clauses[i].length\_current) \* (sizeof(**int**)));;

            clauses[i].literals\_is\_assigned[clauses[i].length\_current - 1] = UNASSIGNED;

**int** t = x > 0 ? SATISFIED : SHRUNK;

**int** s = abs(x);

            ++literal\_info[s][t].n\_number;

            literal\_info[s][t].literal\_clause = (**int**\*)realloc(literal\_info[s][t].literal\_clause, literal\_info[s][t].n\_number \* (sizeof(**int**)));

            literal\_info[s][t].literal\_clause[literal\_info[s][t].n\_number - 1] = i;

            literal\_info[s][t].literal\_clause\_pos = (**int**\*)realloc(literal\_info[s][t].literal\_clause\_pos, literal\_info[s][t].n\_number \* (sizeof(**int**)));

            literal\_info[s][t].literal\_clause\_pos[literal\_info[s][t].n\_number - 1] = clauses[i].length\_current - 1;

        }

        clauses[i].length\_original = clauses[i].length\_current;

        if (clauses[i].length\_original > max\_clause\_len) {

            max\_clause\_len = clauses[i].length\_original;

        }

    }

    fclose(fp);

    return YES;

}

**void** CheckCNF() {

    for (**int** i = 0; i < original\_formula\_length; i++) {

        for (**int** j = 0; j < clauses[i].length\_current - 1; j++) {

            printf("%d ", clauses[i].literals[j]);

        }

        if (clauses[i].length\_current) {

            printf("%d ", clauses[i].literals[clauses[i].length\_current - 1]);

        }

        putchar('\n');

    }

}

3dispaly

#include "ph.h"

*//读取文件名*

FILE**\*** Inputfilename(**char\*** filename) {

    printf("filename:   ");

    scanf("%s", filename);

    FILE\* fp = fopen(filename, "r");

    return fp;

}

*//功能选择界面显示*

**int** Display() {

    system("cls");

*//printf("选择一个\n");*

    printf("1 SAT\n");

    printf("2 Sudoku\n");

    printf("0 Exit\n");

**char** c;

    c = getchar();

    while (!(c=='0'||c=='1'||c=='2')) {

        if (c != '\n') {

            printf("不合法的输入\n");

        }

        while ((c = getchar()) == ' ' || c == '\n' || c == '\t');

    }

    fflush(stdin);

    return c-'0';

}

*//打印数独*

**void** printSudoku(Sudoku**\*** s){

**int** i, j;

*//printf("此为生成的数独:\n");*

    printf("-------------------\n");

    for (i = 0; i < 9; i++) {

        printf("|");

        for (j = 0; j < 9; j++) {

            printf("%d%c", s->s[i][j], ((j + 1) % 3 == 0) ? '|' : ' ');

        }

        printf("\n");

        if ((i + 1) % 3 == 0) {

            printf("|");

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

        }

    }

}

**int** Is\_Show\_CNF() {

    printf("是否展示合取范式Y/N\n");

**char** c;

    while ((c = getchar()) == ' ' || c == '\n' || c == '\t');

    if (c == 'y' || c == 'Y') {

*//printf("这个合取范式如下\n");*

        return YES;

    }

    else return NO;

}

*//将可满足性问题的求解结果输出到屏幕*

*/\*void PrintToScreen(int value,double time) {*

*if (value == SAT) {*

*printf("s 1\n");*

*printf( "v");*

*for (int i = 1; i <= n\_vars; i++) {*

*if (result[i].value == TRUE) {*

*printf(" %d", i);*

*}*

*else {*

*printf( " -%d", i);*

*}*

*}*

*printf( "\n");*

*printf("t %.2f %d\n", time, dpll\_called);*

*printf("该合取范式是可满足的\n");*

*}*

*else if (value == UNSAT) {*

*printf("s 0\n");*

*printf("t %.2f %d\n", time, dpll\_called);*

*printf("该合取范式是不可满足的\n");*

*}*

*}\*/*

*//是否展示合取范式*

**4 solver.cpp**

#include "ph.h"

*//预处理，将单子句加入全局栈unit\_clause\_stack中。这些单子句将在后续的DPLL算法中优先处理。*

**void** Preprocesser() {

    for (**int** i = 0; i < original\_formula\_length; i++) {

        if (clauses[i].length\_original == 1) {*//将长度为一的变量加入全局栈*

            unit\_clause\_stack[n\_unit\_clause\_stack] = clauses[i].literals[0];

            ++n\_unit\_clause\_stack;

        }

    }

}

*//设置子句*

**void** Value(**int** v) {

**register** **int** i;

**register** **int** p = abs(v), q = (v > 0) ? SATISFIED : SHRUNK;

    for (i = 0; i < literal\_info[p][q].n\_number; ++i) {

**register** **int** j = literal\_info[p][q].literal\_clause[i];

        if (clauses[j].is\_clause\_satisfied) continue;*//将j设为满足的*

        clauses[j].is\_clause\_satisfied = YES;*//设置为满足后，不改变句子长度，直到回溯后*

        --current\_formula\_length;*//更改子句的计数器*

        changes\_stack[changes\_stack\_index++].index\_of\_clause = j;*//一个子句被设置为满足，那么记录这个子句的序号*

        n\_changes[depth][SATISFIED]++;*//depth层++*

    }

    q = !q;

    for (i = 0; i < literal\_info[p][q].n\_number; ++i) {

**register** **int** j = literal\_info[p][q].literal\_clause[i];*//记录字句序号*

        if (clauses[j].is\_clause\_satisfied) continue;

**register** **int** k = literal\_info[p][q].literal\_clause\_pos[i];*//记录文字位置*

        --clauses[j].length\_current;*//改变字句长度*

        clauses[j].literals\_is\_assigned[k] = ASSIGNED;*//用表示子句中各个文字是否被赋值*

        changes\_stack[changes\_stack\_index].index\_of\_clause = j;*//记录改变*

        changes\_stack[changes\_stack\_index++].index\_of\_literal = k;*//*

        n\_changes[depth][SHRUNK]++;

        if (clauses[j].length\_current == 1) {*//检查字句长度，判断是否为单子句*

**register** **int** location = -1;

            for (**int** i\_literal = 0; i\_literal < clauses[j].length\_original; i\_literal++) {

                if (clauses[j].literals\_is\_assigned[i\_literal] == UNASSIGNED) {

                    location = i\_literal;

                    break;

                }

            }

**register** **int** w = clauses[j].literals[location];*//单子句中剩余的文字*

**register** **int** s = abs(w), t = (w > 0) ? SATISFIED : SHRUNK;

            if (literal\_info[s][(!t)].is\_in\_unit\_clause == YES) {*//记录冲突*

                is\_contradicted = TRUE;

                conflicting\_literal = w;

            }

            else if (literal\_info[s][t].is\_in\_unit\_clause == NO) {*//没有冲突,把这个单子句放入全局变量*

                unit\_clause\_stack[n\_unit\_clause\_stack] = clauses[j].unit\_cluase\_literal = w;

                literal\_info[s][t].is\_in\_unit\_clause = YES;

                ++n\_unit\_clause\_stack;

            }

        }

    }

    literal\_info[p][SHRUNK].is\_assigned = YES;

    literal\_info[p][SATISFIED].is\_assigned = YES;

    ++depth;

}

*//取消设置*

**void** UnValue(**int** v){

**register** **int** i;

**register** **int** p = abs(v), q = (v > 0) ? SATISFIED : SHRUNK;

    --depth;

    while (n\_changes[depth][SHRUNK]){*//还有没从stack中取出来的*

        --n\_changes[depth][SHRUNK];*//*

**register** **int** j = changes\_stack[--changes\_stack\_index].index\_of\_clause;

**register** **int** k = changes\_stack[changes\_stack\_index].index\_of\_literal;

        ++clauses[j].length\_current;*//回复子句的长度*

        if (clauses[j].length\_current == 2) {*//将被设置为在单子句中出现的文字设为未出现*

**int** s = abs(clauses[j].unit\_cluase\_literal);

**int** t = (clauses[j].unit\_cluase\_literal > 0) ? SATISFIED : SHRUNK;

            literal\_info[s][t].is\_in\_unit\_clause = NO;

            clauses[j].unit\_cluase\_literal = 0;

        }

        clauses[j].literals\_is\_assigned[k] = UNASSIGNED;*////*

    }

    while (n\_changes[depth][SATISFIED]) {

        --n\_changes[depth][SATISFIED];

**register** **int** j = changes\_stack[--changes\_stack\_index].index\_of\_clause;

        clauses[j].is\_clause\_satisfied = NO;

        ++current\_formula\_length;

    }

    literal\_info[p][SATISFIED].is\_assigned = NO;

    literal\_info[p][SHRUNK].is\_assigned = NO;

}

*//DPLL递归算法*

**int** dpll() {

    ++dpll\_called;*//用于输出调试信息*

**int**\* local\_stack = NULL;

**register** **unsigned** **int** n\_local\_stack = 0;

    while (true) {

        if (is\_contradicted) {*//return*

**int** cl = abs(conflicting\_literal);

            while (n\_local\_stack) {

                UnValue(local\_stack[--n\_local\_stack]);

**register** **int** s = abs(local\_stack[n\_local\_stack]);

**register** **int** t = local\_stack[n\_local\_stack] > 0 ? TRUE : FALSE;

                result[s].value = UNASSIGNED;

            }

            is\_contradicted = FALSE;

            free(local\_stack);

            n\_unit\_clause\_stack = 0;

            return UNSAT;

        }

        else if (n\_unit\_clause\_stack) {*//*

            local\_stack = (**int**\*)realloc(local\_stack, (n\_local\_stack + 1) \* sizeof(**int**));

**register** **int** implied\_lit = unit\_clause\_stack[--n\_unit\_clause\_stack];*//将要进行单子句规则的文字*

            local\_stack[n\_local\_stack++] = implied\_lit;

            result[abs(implied\_lit)].value = implied\_lit > 0 ? TRUE : FALSE;

            Value(implied\_lit);

        }

        else break;

    }

    if (!current\_formula\_length)

        return SAT;*//如果当前子句数为0,则返回SAT*

**register** **int** v = GetMinLenOfLiteral();*//变元选取策略*

    result[abs(v)].value = v > 0 ? TRUE : FALSE;

    Value(v);

    if (dpll()) return SAT;

    UnValue(v);

**register** **int** i, j, k, m ;

    result[abs(v)].value = !result[abs(v)].value;

    Value(-v);

    if (dpll()) return SAT;

    UnValue(-v);

    result[abs(v)].value = UNASSIGNED;

    while (n\_local\_stack) {

**int** z = local\_stack[--n\_local\_stack];

        UnValue(z);

        result[abs(z)].value = UNASSIGNED;

    }

    free(local\_stack);

    is\_contradicted = FALSE;

    return UNSAT;

}

*//获取最短子句的长度*

**inline** **int** get\_length\_of() {

**register** **int** i, s,type, C, min = max\_clause\_len;

    if (min == 2) return min;*//如果最长子句的长度为2，那么min同样为*

    for (i = 1; i <= n\_vars; ++i) {

        if (result[i].value == UNASSIGNED) {

            for (type = 0; type < 2; ++type) {

                for (s = 0; s < literal\_info[i][type].n\_number; ++s) {

                    C = literal\_info[i][type].literal\_clause[s];*//每一个文字，在每一个句子中的位置*

                    if (!clauses[C].is\_clause\_satisfied &&

                        clauses[C].length\_current < min) {

                        min = clauses[C].length\_current;

                        if (min == 2) return 2;

                    }

                }

            }

        }

    }

    return min;

}

*//获取子句的权重*

**void** get\_weight(**int** x, **int** k, **unsigned** **int&** s, **unsigned** **int&** t) {

**register** **int** j, c;

    s = t = 0;

    for (j = 0; j < literal\_info[x][SATISFIED].n\_number; ++j) {

        c = literal\_info[x][SATISFIED].literal\_clause[j];*//子句序号*

        if (clauses[c].length\_current == k)

            s += 1 - clauses[c].is\_clause\_satisfied;

    }

    for (j = 0; j < literal\_info[x][SHRUNK].n\_number; ++j) {*// 子句序号*

        c = literal\_info[x][SHRUNK].literal\_clause[j];

        if (clauses[c].length\_current == k)

            t += 1 - clauses[c].is\_clause\_satisfied;

    }

}

*//获取子句的长度*

**inline** **int** GetMinLenOfLiteral() {

**register** **unsigned** **int** i, k;

**register** **unsigned** **int** max = 0, r, s, t;

**register** **int** u;

    for (i = 1; i <= n\_vars; ++i) {

        if (result[i].value == UNASSIGNED) {

            k = get\_length\_of();*//最短子句*

            get\_weight(i, k, s, t);*//i 变元 k  长度*

            r = (s + 1) \* (t + 1);

            if (r > max) {

                max = r;

                if (s >= t) u = i;

                else u = -(**int**)i;

            }

        }

    }

    return u;

}

*//将求解结果输出到res文件*

**void** PrintToRes(**int** value, **double** time, **char\*** filename) {

**int** len = strlen(filename);

    filename[len - 3] = 'r';

    filename[len - 2] = 'e';

    filename[len - 1] = 's';

**int** seed,ii=0;

    while (0<=(filename[len-5-ii]-'0')&&(filename[len-5-ii]-'0')<=12)

    {

        seed=filename[len-5-ii]-'0';

        ii++;

    }

**double** time\_used[13]={111,12.0,18.0,8.0,15.0,100.0,111,111,111,111,111,111,111};

**double** rate=0.0;

    if(time\_used[seed]==111)

        rate=100.0;

    else

        rate=(time\_used[seed]-time)/time\_used[seed]\*100;

    FILE\* fp = fopen(filename, "w");

    if (fp == NULL) {

        printf("打开文件%s失败\n", filename);

        return;

    }

    if (value == SAT) {

        fprintf(fp, "s 1\n");

        fprintf(fp, "v");

        for (**int** i = 1; i <= n\_vars; i++) {

            if (result[i].value == TRUE) {

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

            }

            else {

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

            }

        }

        fprintf(fp, "\n");

*//fprintf(fp, "t %.2f %.1f%c \n", time,rate,'%');*

        fprintf(fp, "t %.2f\n", time);

    }

    else if (value == UNSAT) {

        fprintf(fp, "s 0\n");

*//fprintf(fp, "t %.2f  %.1f%c \n", time,rate,'%');*

        fprintf(fp, "t %.2f\n", time);

    }

    fclose(fp);

}

*//释放分配的内存*

**void** Free() {

    for (**int** i = 1; i <= n\_vars; i++) {

        if (literal\_info[i][0].literal\_clause) {

            free(literal\_info[i][0].literal\_clause);

            literal\_info[i][0].literal\_clause = NULL;

        }

        if (literal\_info[i][0].literal\_clause\_pos) {

            free(literal\_info[i][0].literal\_clause\_pos);

            literal\_info[i][0].literal\_clause\_pos = NULL;

        }

        if (literal\_info[i][1].literal\_clause) {

            free(literal\_info[i][0].literal\_clause);

            literal\_info[i][0].literal\_clause\_pos = NULL;

        }

        if (literal\_info[i][1].literal\_clause\_pos) {

            free(literal\_info[i][0].literal\_clause\_pos);

            literal\_info[i][0].literal\_clause\_pos = NULL;

        }

    }

    for (**int** i = 0; i < original\_formula\_length; i++) {

        if (clauses[i].literals) {

            free(clauses[i].literals);

        }

        if (clauses[i].literals\_is\_assigned) {

            free(clauses[i].literals\_is\_assigned);

        }

    }

    free(clauses);

}

**5 X\_Sudoku**

#include "ph.h"

*//开始生成数独棋盘*

**void** startgame(){

    srand(**unsigned**(time(NULL)));

    for (**int** i = 1; i <= 9; ++i)

        for (**int** j = 1; j <= 9; ++j)

            sudokuterminal[i][j] = 0;

**int** visited[10] = { 0 };

    for (**int** i = 1; i <= 8; i++) {

        while (sudokuterminal[1][i] == 0) {

**int** data = rand() % 9 + 1;

            if (visited[data] == 0) {

                visited[data] = 1;

                sudokuterminal[1][i] = data;

            }

        }

    }

    for (**int** i = 9; i <= 9; i++) {

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

            if (visited[j] == 0) {

                sudokuterminal[1][i] = j;

                visited[j] = 1;

                break;

            }

        }

    }

    createfinalsudoku(2, 1); */\* 从第二行开始搜索 \*/*

}

*//填充数独终盘*

**int** createfinalsudoku(**int** i, **int** j){

    if (i > 9 || j > 9) {

        return true;

    }

    for (**int** k = 1; k <= 9; ++k){

**int** type = true; *// type变量用于记录数字k能否放在 ( i , j ) 处*

        if (type)

        {

            for (**int** n = 1; n < j; ++n) {

                if (sudokuterminal[i][n] == k){ *// 检查同一行是否出现过数字k*

                    type = false;

                    break;

                }

            }

        }

        for (**int** s = 1; s < i; ++s) {

            if (sudokuterminal[s][j] == k){ *// 检查同一列是否出现过数字k*

                type = false;

                break;

            }

        }

        if(type&&i==j){

*//检查整个棋盘的主副两条对角线是否出现过数字k*

            for(**int** s=1;s<i;s++)

                if(sudokuterminal[s][s]==k)

                    type=false;

        }

        if(type&&i+j==10){

*//检查整个棋盘的主副两条对角线是否出现过数字k*

            for(**int** s=1;s<i;s++)

                if(sudokuterminal[s][9-s+1]==k)

                    type=false;

        }

        if (type){

**int** upi = ((i-1) / 3) \* 3 + 3; *// i坐标的上限*

**int** upj = ((j-1) / 3) \* 3 + 3; *// j坐标的上限*

            for (**int** p = upi - 2; p <= upi; ++p) { *//检查在3×3的小方格中是否出现了同一个数字*

                if (type == false) {

                    break;

                }

                for (**int** q = upj - 2; q <= upj; ++q) {

                    if (sudokuterminal[p][q] == k) {

                        type = false;

                        break;

                    }

                }

            }

        }

        if (type){*//搜索剩余格子*

            sudokuterminal[i][j] = k;

            if (j < 9){

                if (createfinalsudoku(i, j + 1)) *//到同一行的下一位置开始搜索*

                    return true;

            }

            else{

                if (i < 9){

                    if (createfinalsudoku(i + 1, 1)) *//下一行的第一个位置开始搜索*

                        return true;

                }

                else {

                    return true; *//i >= 9  && j >= 9  , 搜索结束*

                }

            }

            sudokuterminal[i][j] = 0; *//回溯*

        }

    }

    return false;

}

*//将生成的数组转化为数独*

**void** CopytoSudoku(Sudoku**\*** s) {

    for (**int** i = 1; i < 10; i++) {

        for (**int** j = 1; j < 10; j++) {

            s->s[i - 1][j - 1] = sudokuterminal[i][j];

        }

    }

    s->n\_current = 81;

}

*//利用挖洞发产生数独*

Sudoku**\*** createSudoku(Sudoku**\*** s,**int** difficulty, **int** purpose){

**int** i, j, k, c, flag = 0, x, y;

    x = rand() % 9;

    y = rand() % 9;

    k = s->s[x][y];

    s->s[x][y] = 0;*//先挖掉1个洞*

    --s->n\_current;

    while (true) {*//挖洞过程*

        startsolveSudoku(s, &c);*//求解该数独，得到解的个数c*

        if (c > 1) { *//此时解不唯一*

            s->s[x][y] = k;

            ++s->n\_current;

            flag++; *//*

            if (flag >= difficulty) {

                break;

            }

        }

        if (purpose >= s->n\_current) {

            break;

        }

        do {

            x = rand() % 9;

            y = rand() % 9;

        } while (s->s[x][y] == 0);

        k = s->s[x][y]; *//此时(x,y)不为0，记录下这个值*

        s->s[x][y] = 0;

        --s->n\_current;

    }

    return s;

}

*//开始求解数独*

**int** startsolveSudoku(Sudoku**\*** s, **int\*** count) {

    \*count = 0;

    solveSudoku(s, 0, 0, count);

    if (count > 0) {

        return true;

    }

    else {

        return false;

    }

}

*//求解数独*

**int** solveSudoku(Sudoku**\*** s, **int** x, **int** y, **int\*** count) {

**int** i, j, k;

    while (s->s[x][y] != 0 && x < 9){*//从上往下找找到第一个为0的空*

        y++;

        if (y == 9) {*//搜索完一行*

            x++;

            y = 0;

        }

    }

    if (x > 8) {*//这时数独棋盘没有空的*

        (\*count)++;

        return true;

    }

    for (i = 1; i <= 9; i++){

        s->s[x][y] = i;

        if ((k = checkkeyword(s, x, y)) == true){

            if (x == 8 && y == 8) {*//达到最后一个格子*

                s->s[x][y] = 0;

                (\*count)++;

                return true;

            }

            else{

                if (y == 8) *//求解下一行*

                    solveSudoku(s, x + 1, 0, count);

                else *//求解下一列*

                    solveSudoku(s, x, y + 1, count);

            }

        }

    }

    s->s[x][y] = 0;

    return false;

}

*//检验填入的数字是否正确*

**int** checkkeyword(Sudoku**\*** s, **int** x, **int** y) {

    for (**int** i = x / 3 \* 3; i < x / 3 \* 3 + 3; i++) {*//检查3\*3九宫格是否冲突*

        for (**int** j = y / 3 \* 3; j < y / 3 \* 3 + 3; j++) {

            if (s->s[x][y] == s->s[i][j] && (i != x || j != y)) {

                return false;

            }

        }

    }

    for (**int** i = 0; i < 9; i++) { *//检查横行是否冲突*

        if (s->s[x][y] == s->s[i][y] && i != x) {

            return false;

        }

    }

    for (**int** i = 0; i < 9; i++) { *//检查竖列是否冲突*

        if (s->s[x][y] == s->s[x][i] && i != y) {

            return false;

        }

    }

    return true;

}

*//三维数组寻址公式对应着9\*9\*9个CNF变元*

**inline** **int** Location(**int** x, **int** y, **int** z) {

    return x \* 81 + y \* 9 + z;

}

*//将数独转换为cnf文件*

FILE**\*** ToCNFFile(Sudoku**\*** s, **char\*** filename) {

*//11988*

*//729*

    FILE\* fp = fopen(filename, "w+");

    if (fp == NULL) {

        return NULL;

    }

    fprintf(fp, "p cnf 729 11988\n");

    for (**int** i = 0; i < 9; i++) {*//每个单元格只能填入唯一一个数字*

        for (**int** j = 0; j < 9; j++) {

            if (s->s[i][j] == 0) {

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

                    fprintf(fp, "%d ", Location(i, j, k));

                }

            }

            else {

                fprintf(fp, "%d ", Location(i, j, s->s[i][j]));

            }

            fprintf(fp, "0\n");

        }

    }

for (**int** i = 0; i < 9; i++) { *// 每个单元格只能填入唯一一个数字*

    for (**int** j = 0; j < 9; j++) {

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

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

                fprintf(fp, "-%d -%d 0\n", Location(i, j, m), Location(i, j, n));

            }

        }

    }

}

for (**int** k = 1; k <= 9; k++) { *// 行约束*

    for (**int** i = 0; i < 9; i++) {

        for (**int** j = 0; j < 9; j++) {

            fprintf(fp, "%d ", Location(i, j, k));

        }

        fprintf(fp, "0\n");

    }

}

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

    for (**int** i = 0; i < 9; i++) {

        for (**int** m = 0; m < 9; m++) {

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

                fprintf(fp, "-%d -%d 0\n", Location(i, m, k), Location(i, n, k));

            }

        }

    }

}

for (**int** k = 1; k <= 9; k++) { *// 列约束*

    for (**int** j = 0; j < 9; j++) {

        for (**int** i = 0; i < 9; i++) {

            fprintf(fp, "%d ", Location(i, j, k));

        }

        fprintf(fp, "0\n");

    }

}

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

    for (**int** j = 0; j < 9; j++) {

        for (**int** m = 0; m < 9; m++) {

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

                fprintf(fp, "-%d -%d 0\n", Location(m, j, k), Location(n, j, k));

            }

        }

    }

}

for (**int** i = 0; i < 7; i = i + 3) { *// 九宫格约束*

    for (**int** j = 0; j < 7; j = j + 3) {

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

            for (**int** pi = 0; pi < 3; pi++) {

                for (**int** pj = 0; pj < 3; pj++) {

                    fprintf(fp, "%d ", Location(i + pi, j + pj, k));

                }

            }

            fprintf(fp, "0\n");

        }

    }

}

*/\**

*// 主对角线约束*

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

*for (int i = 0; i < 9; i++) {*

*fprintf(fp, "%d ", Location(i, i, k));*

*}*

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

*}*

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

*for (int i = 0; i < 9; i++) {*

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

*fprintf(fp, "-%d -%d 0\n", Location(i, i, k), Location(j, j, k));*

*}*

*}*

*}*

*// 副对角线约束*

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

*for (int i = 0; i < 9; i++) {*

*fprintf(fp, "%d ", Location(i, 8 - i, k));*

*}*

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

*}*

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

*for (int i = 0; i < 9; i++) {*

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

*fprintf(fp, "-%d -%d 0\n", Location(i, 8 - i, k), Location(j, 8 - j, k));*

*}*

*}*

*}*

*\*/*

**int** respi[10] = { 0 };

**int** respj[10] = { 0 };

**int** countt = 0;

    for (**int** pi = 0; pi < 3; pi++) {

        for (**int** pj = 0; pj < 3; pj++) {

            respi[countt] = pi;

            respj[countt] = pj;

            countt++;

        }

    }

    for (**int** i = 0; i < 7; i = i + 3) {

        for (**int** j = 0; j < 7; j = j + 3) {

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

*//人两个位置，fprintf*

                for (**int** m = 0; m < 9; m++) {

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

                        fprintf(fp, "-%d -%d 0\n", Location(i + respi[m], j + respj[m], k),

                            Location(i + respi[n], j + respj[n], k));

                    }

                }

            }

        }

    }

    return fp;

}

*//将求解结果转化为数独*

Sudoku**\*** ResultToSudoku() {

    if (n\_vars != 729) {

        return NULL;

    }

    Sudoku\* S = (Sudoku\*)malloc(sizeof(Sudoku));

**int** countk = 0;

    for (**int** i = 0; i < 9; i++) {

        for (**int** j = 0; j < 9; j++) {

**int** type = 0;

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

                if (result[++countk].value == TRUE) {

                    if (type == 1) {

                        printf("同一位置有两个数\n");

                        return NULL;

                    }

                    else {

                        type = 1;

                        S->s[i][j] = k;

                    }

                }

                if (result[countk].value == UNASSIGNED) {

                    printf("有数字没有赋值\n");

                }

            }

        }

    }

    for (**int** i = 0; i < 9; i++) {

**int** visited[10] = { 0 };

        for (**int** j = 0; j < 9; j++) {

            if (visited[S->s[i][j]] == 0) {

                visited[S->s[i][j]] = 1;

            }

            else {

                printf("行冲突\n");

            }

        }

    }

    for (**int** j = 0; j < 9; j++) {

**int** visited[10] = { 0 };

        for (**int** i = 0; i < 9; i++) {

            if (visited[S->s[i][j]] == 0) {

                visited[S->s[i][j]] = 1;

            }

            else {

                printf("列冲突\n");

            }

        }

    }

    for (**int** i = 0; i < 7; i = i + 3) {

        for (**int** j = 0; j < 7; j = j + 3) {

**int** visited[10] = { 0 };

            for (**int** pi = 0; pi < 3; pi++) {

                for (**int** pj = 0; pj < 3; pj++) {

                    if (visited[S->s[i + pi][j + pj]] == 0) {

                        visited[S->s[i + pi][j + pj]] = 1;

                    }

                    else {

                        printf("九宫格冲突\n");

                        return NULL;

                    }

                }

            }

        }

    }

    return S;

}

*//游戏*

**void** game(Sudoku**\*** gameboard, Sudoku**\*** ans) {

**int** is\_acc = FALSE;

    if (gameboard->n\_current == 81) {

        is\_acc = TRUE;

    }

    while (is\_acc == FALSE) {

        system("cls");

        printSudoku(gameboard);

        printf("还有%d个数字等待填充\n", 81 - gameboard->n\_current);

        printf("0 查看答案  1 输入  2 提示");

**int** type;

        fflush(stdin);

        scanf("%d", &type);

        fflush(stdin);

        if (type == 0) {

*//printf("答案即将公布\n");*

            printSudoku(ans);

            is\_acc = TRUE;

        }

        else if(type==1){

**int** x, y, z;

            printf("请输入：行号 列号 数字(1~9)\n");

            fflush(stdin);

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

            if (1 <= x && x <= 9 && 1 <= y && y <= 9 && 1 <= z && z <= 9) {

                if (!gameboard->s[x-1][y-1]) {

                    printf("输入不合法，请重新输入\n");

                }

                else {

                    if (z == ans->s[x - 1][y - 1]) {

                        printf("恭喜你，这是正确的\n");

                        gameboard->s[x - 1][y - 1] = z;

                        gameboard->n\_current++;

                        if (gameboard->n\_current == 81) {

                            is\_acc = TRUE;

                            printf("OK，你已经完成了！\n");

                        }

                    }

                    else {

                        printf("很遗憾，这是错误的，再试一试吧\n");

                    }

                }

            }

            else {

                printf("输入不合法，请重新输入\n");

            }

            fflush(stdin);

        }

        else if (type==2) {

**int** x = rand() % 9;

**int** y = rand() % 9;

**int** i = x, j = y;

            do {

                if (gameboard->s[i][j] == 0) {

                    break;

                }

                if (j == 8) {

                    j = 0; i++;

                }

                else {

                    j++;

                }

                if (i == 9) {

                    i = 0;

                }

            } while (i != x || j != y);

            printf("第%d行第%d列为%d\n", i+1, j+1, ans->s[i][j]);

            gameboard->s[i][j] = ans->s[i][j];

            gameboard->n\_current++;

            if (gameboard->n\_current == 81) {

                is\_acc = TRUE;

                printf("OK，你已经完成了！\n");

            }

        }

        else {

            printf("输入不合法\n");

        }

        system("pause");

    }

}

*//选择难度*

**int** ChooseDifficluty() {

    printf("请选择挖洞的个数：\n");

*//printf("24~80\n");*

**int** x;

    scanf("%d", &x);

    while (!(24 <= x && x <= 80)) {

        printf("非法输入，请重新输入\n");

        scanf("%d", &x);

        fflush(stdin);

    }

    return 81-x;

}

**6 ph.h**

#ifndef PH\_H

#define PH\_H

#ifndef HEAD\_H\_INCLUDED

#define HEAD\_H\_INCLUDED

#include<stdio.h>

#include <stdio.h>

#include <time.h>

#include <math.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

*//全局变量*

#define MAX\_VARS 60000 *//最大变元数*

#define MAX\_CLAUSES 230000 *//最大子句数*

#define ASSIGNED -2*// 暂时的赋值*

#define UNASSIGNED -1*//还没有理解的赋值*

#define TRUE 1

#define FALSE 0

#define YES 1

#define NO 0

#define SATISFIED 1

#define SHRUNK 0

#define SAT 1

#define UNSAT 0

typedef **int** status;

typedef **struct** literal\_infomation {*//文字结构*

**int**\* literal\_clause;*//文字在公式中的位置*

**int**\* literal\_clause\_pos;*//文字在子句中的位置*

**int** n\_number;*//个数*

**int** is\_assigned;*//文字是否被赋值*

**int** is\_in\_unit\_clause;*//记录是否在单子句中出现 NO or YES*

}literal\_infomation;

**extern** literal\_infomation literal\_info[MAX\_VARS][2];*//文字存储数组*

typedef **struct** clause\_infomation {*//子句结构*

**int**\* literals;*//子句中的文字*

**int**\* literals\_is\_assigned;*//1 for unassigned 0 for false*

**int** length\_current;*//现在的字句长度*

**int** length\_original;*//最初子句长度*

**int** is\_clause\_satisfied;*//是否满足 NO OR YES true*

**int** unit\_cluase\_literal;*//存储单子句文字中，未赋值文字*

}clause\_infomation;

**extern** clause\_infomation\* clauses;*//子句存储数组*

**extern** **int** original\_formula\_length;*//原长度*

**extern** **int** current\_formula\_length;*//现长度*

typedef **struct** changes\_infomation {*//回溯信息*

**int** index\_of\_clause; *//子句序号*

**int** index\_of\_literal;*//文字序号*

}changes\_infomation;*//子句满足时，记录子句序号。文字被设为假时，记录文字和字句序号*

**extern** changes\_infomation changes\_stack[8\*MAX\_CLAUSES];*//记录*

**extern** **unsigned** **int** changes\_stack\_index;*//数组changes的下标*

**extern** **unsigned** **int** n\_changes[MAX\_VARS][2];*//某一层满足或不满足的句子的个数*

typedef **struct** assign\_infomation {*//记录赋值操作*

**int** value; *//赋值类型UNASSIGNED TRUE FALSE*

}assign\_infomation;

**extern** assign\_infomation result[MAX\_VARS];*//用于回溯操作的记录*

**extern** **int** is\_contradicted;*//是否用冲突的单子句*

**extern** **int** conflicting\_literal;*//用于记录冲突的文字*

**extern** **int** unit\_clause\_stack[MAX\_CLAUSES];*//单子句规则*

**extern** **int** n\_unit\_clause\_stack;*//全局栈，用于记录栈顶位置*

**extern** **int** depth;*//DPLL树中，结点的深度*

**extern** **int** n\_vars;*//变元总个数*

**extern** **int** dpll\_called;*//统计DPLL执行次数*

**extern** **int** sudokuterminal[10][10];*//生成数独终盘*

**extern** **int** max\_clause\_len;*//最大字句长度*

typedef **struct** Sudoku {*//数独结构*

**int** s[9][9];

**int** n\_current;

}Sudoku;

*//显示部分的函数名*

**int** Display();*//展示选项*

FILE**\*** Inputfilename(**char\*** filename); *//输入文件名*

**int** Is\_Show\_CNF();*//是否展示cnf文件*

**void** PrintToScreen(**int** value, **double** time);*//将结果输出到屏幕*

**void** printSudoku(Sudoku**\*** s);*//打印数独*

*//CNFparser*

**int** CNFparser(FILE**\*** fp);*//合取范式*

**void** CheckCNF();*//输出这个合取范式*

*//solver part*

**void** Preprocesser();*//预处理器*

**int** dpll();*//dpll算法*

**void** Value(**int** v);*//将文字v设定为已赋值*

**void** UnValue(**int** v);*//将文字设定为未赋值*

**inline** **int** GetMinLenOfLiteral();*//*

**inline** **int** get\_length\_of();*//计算最短字句长度*

**void** get\_weight(**int** x, **int** k, **unsigned** **int&** s, **unsigned** **int&** t);*//获得权重*

**void** PrintToRes(**int** value, **double** time, **char\*** filename);*//将结果输出到.res文件*

**void** Free();*//释放动态分配的空间*

*//数独部分的函数名*

**void** startgame();*//生成数组*

**void** CopytoSudoku(Sudoku**\*** s);*//将数组复制到数独*

**int** createfinalsudoku(**int** i, **int** j);*//生成数组*

Sudoku**\*** createSudoku(Sudoku**\*** S, **int** difficulty, **int** flag);*//创建数独棋盘*

**int** startsolveSudoku(Sudoku**\*** s, **int\*** count);*//开始解决数独*

**int** solveSudoku(Sudoku**\*** s, **int** x, **int** y, **int\*** count);*//解决数独的递归过程*

**int** checkkeyword(Sudoku**\*** s, **int** x, **int** y);*//检查x，y符合的数字*

**inline** **int** Location(**int** x, **int** y, **int** z);*//三维寻址*

FILE**\*** ToCNFFile(Sudoku**\*** s, **char\*** filename);*//将数独保存为CNF文件*

Sudoku**\*** ResultToSudoku();*//检查数独结果，将res转为数度结果*

**void** game(Sudoku**\*** s, Sudoku**\*** final\_patern);*//开始数独游戏*

**int** ChooseDifficluty();*//选择难度*

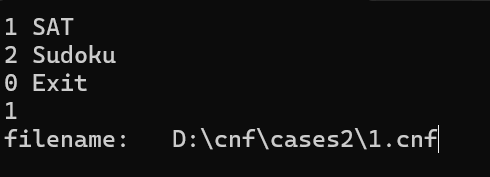
#endif *// HEAD\_H\_INCLUDED*

#endif *//PCH\_H*

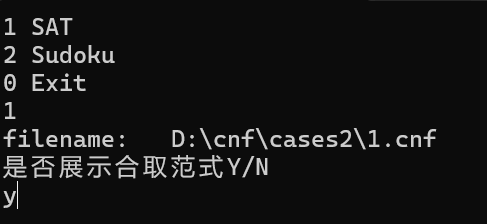
# 附录二操作手册



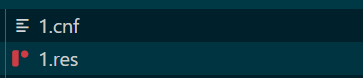
1. 输入1进行DPLL算法解决SAT问题
2. 输入文件地址



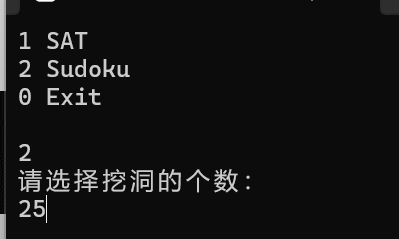
1. 输入Y/N表示是否合取范式



1. 查看结果



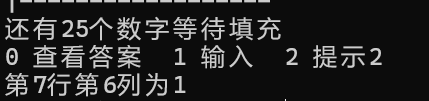
1. 输入2进行对角线数独游戏
2. 输入挖洞个数



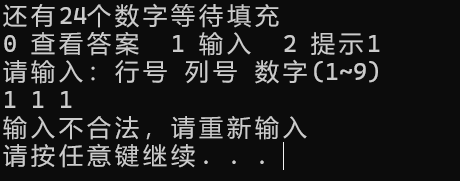
1. 选择玩法



1. 输入2获取提示



1. 输入1填空



1. 输入0查看答案

