

**课程设计报告**

**题目：基于SAT的对角线数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： cs2306班**

**学 号： U202315719**

**姓 名： 郭金环**

**指导教师： 纪俊文**

**报告日期： 2024年9月20日**

**计算机科学与技术学院**

**任务书**

* **设计内容**

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于DPLL算法实现一个完备SAT求解器，对输入的CNF范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

* **设计要求**

要求具有如下功能：

**（1）输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)

**（2）公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)

**（3）DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)

**（4）时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)

**（5）程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)

**（6）SAT应用：**将数双独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

**目录**

**任务书** I

**1引言** 1

1.1课题背景与意义 1

1.2国内外研究现状 2

1.3课程设计的主要研究工作 3

2系统需求分析与总体设计 4

2.1系统需求分析 4

2.2系统总体设计 4

3系统详细设计 6

3.1有关数据结构的定义 6

3.2 主要算法设计 6

4系统实现与测试 13

4.1系统实现 13

4.2系统测试 15

5总结与展望 21

5.1全文总结 21

5.2工作展望 21

6体会 22

参考文献 23

附录□×××××× 24

**1引言**

**1.1课题背景与意义**

**1.1.1SAT问题**

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。SAT问题也是程序设计与竞赛的经典问题。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。SAT问题也是程序设计与竞赛的经典问题。

DPLL算法是经典的SAT完备型求解算法，对给定的一个SAT问题实例，理论上可判定其是否满足，满足时可给出对应的一组解。针对SAT问题的求解算法大致分为完备性算法和非完备性算法两类。完备性算法出现的时间更早，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。而非完备性算法是局部搜索算法，一次仅能判定一个SAT实例的一组赋值是否满足，这种算法不能保证一定找到解，但是求解速度快。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。

SAT 问题的应用领域非常广泛，例如在数学研究和应用领域，它能用来解决旅行商（Traveling Salesman Problem，TSP）和逻辑算术问题；在计算机和人工智能（Artificial Intelligence）领域中，它能解决 CSP（约束满足问题）问题、语义信息的处理和逻辑编程等问题；在计算机辅助设计领域中，它能很好的解决任务规划与设计、三维物体识别等问题。许多的实际问题如人工智能、积木世界规划问题、数据库检索、Job shop 排工问题、超大规模集成电路设计和图着色都可转换为 SAT 问题进行求解。例如在维护知识库时，当把知识表示成合取范式之后，知识库的一致性检查问题便转化为 CNF 公式上的可满足性问题。谓词逻辑公式的不可满足性问题经过一定的消减技术便可以转化为 CNF 公式的不可满足性问题。不仅如此，一些实际问题通常还具有一定的结构特性，可以据此来对具有某一类结构特性的实际应用问题进行求3解，进而得到解决该类问题的一种通用解法。

**1.1.2对角线数独**

对角线数独是从数独衍生出的变种。在9×9的大[九宫格](https://baike.baidu.com/item/%E4%B9%9D%E5%AE%AB%E6%A0%BC/2216215?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%AF%B9%E8%A7%92%E7%BA%BF%E6%95%B0%E7%8B%AC/_blank)中填入数字1～9，每个数字在每个小[九宫格](https://baike.baidu.com/item/%E4%B9%9D%E5%AE%AB%E6%A0%BC/2216215?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%AF%B9%E8%A7%92%E7%BA%BF%E6%95%B0%E7%8B%AC/_blank)内不能出现一样的数字，在每行、每列和每条大对角线中也不能出现一样的数字，其相对于标准数独来说是多了两个额外区，要求两条对角线也包括数字1-9。国内外许多学者已对数独的求解算法做了深入研究，例如递归法、回溯候选数法、枚举算法等。

**1.2国内外研究现状**

由于 SAT 问题在计算机理论研究和人工智能领域的重要作用，使得许多学者都在 SAT 问题求解领域做了大量的研究，可满足性问题进而也成为了国内外研究的热点问题，并在算法研究和技术实现上取得了较大的突破，这也推动了形式验证和人工智能等领域的发展。

Bart Selman 和 Henry Kautz 分别于 1997 年和 2003 年在人工智能第五届国际合作会议上提出了 SAT 问题面临的十大挑战性问题，并在 2001 年和 2007 年先后对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出对于 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。

最经典的求解 SAT 问题的完备算法是 DPLL 算法，它是由 Davis 和 Putnam 等人在 1960 年提出[3]，其它的完备算法大都是在 DPLL 算法的基础上衍生出来的，是对 DPLL 算法的改进。由于 SAT 问题本身的特性使得其最坏情况下的时间复杂度是指数级别，最初这使得许多的研究者望而却步。而后，S.A.Cook 在 1971 年证明了 SAT 问题是 NP 完全问题，这更加削弱了许多学者研究 SAT 问题的兴趣，从而导致了 SAT 问题在很长的一段时间里都没有得到较好的重视，发展非常缓慢，研究成果较少。但是 1996 年以后，很多国家都相继举办了一些 SAT 竞赛和研讨会，这使得越来越多的人开始关注并研究 SAT 问题，所以这段时间也涌现出了众多新的高效的 SAT 算法如 MINISAT、SATO、CHAFF、POSIT和 GRASP等。国内也涌现出了许多高效的求解算法，如 1998 年作者梁东敏提出了改进的子句加权 WSAT 算法，2000年金人超和黄文奇提出的并行 Solar 算法，2002 年作者张德富提出模拟退火算法。

**1.3课程设计的主要研究工作**

本设计要求精心设计问题中变元、文字、子句、公式等有效的物理存储结构，基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。并能实现如下功能：

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。

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。

DPLL过程：基于DPLL算法框架，实现SAT算例的求解。

时间性能的测量：基于相应的时间处理函数，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。

程序优化：对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供明确的性能优化率结果。

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系统需求分析与总体设计**

**2.1系统需求分析**

本设计基于DPLL的算法与程序框架，实现一个完备的SAT求解器，通过一定的策略在 DPLL 框架的基础上实现优化，并将数独问题归约为 SAT 问题，并基于SAT求解器完成数独游戏的设计。

**2.2系统总体设计**

系统由四个模块组成：CNF解析模块（cnfparser.h）：定义外部变量和存储结构，实现读取cnf文件、解析文件，基于一定的物理结构，建立公式的内部表示，创建res文件保存结果；

核心DPLL模块(solver.h):使用不同的变元选择策略实现dpll算法求解sat问题；

对角线数独模块(X-Sudoku.h)：包括游戏格局生成，归约为sat问题，填写数独，给出数独的正确答案；

主控、交互与显示模块（display.cpp）：调用前三个模块中的内容，显示交互环境。

运行程序后，首先选择处理cnf文件的SAT问题还是进行对角线数独游戏。若选择处理SAT问题，则先读取SAT算例（cnf文件），随即进行DPLL，求解出结果，展示并且保存到一个同名res文件中；若选择进行对角线数独游戏，则输入挖洞数初始化一个数独，再进行对角线数独归约，将该数独求解问题转化为SAT问题。



图2-1 系统总体设计图

**3系统详细设计**

**3.1有关数据结构的定义**

系统中要处理的数据结构有：clausenode结构体：包括literal（文字）、score（分数）和指向下一个clausenode的指针next；clauselist结构体：包括listnum（记录子句中的文字数）、指向clausenode链表头部的nodehead指针，以及指向下一个clauselist的指针next。

表3-1 系统数据结构

|  |  |  |
| --- | --- | --- |
| 结构体 | 包含的数据项 | 数据类型 |
| clausenode  （记录文字） | literal | int型 |
| score | int型 |
| next | clausenode指针型 |
| Clauselist  （记录子句） | listnum | int型 |
| nodehead | clausenode指针型 |
| next | clauselist指针型 |

clausenode结构体记录文字，clauselist结构体记录子句，每个子句含不定个数的文字，两个数据的关系如下图所示。

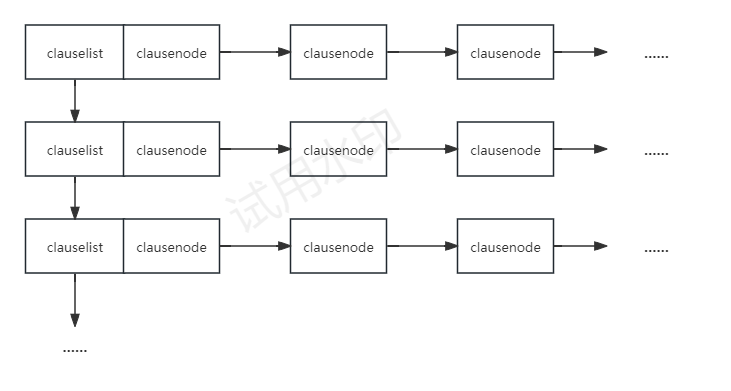


图3-1 存储结构示意图

**3.2 主要算法设计**

**3.2.1DPLL算法思想**

DPLL算法是基于二叉树的回溯搜索算法，，从二叉树到叶子结点的一条包含所有变元的路径表示 CNF 公式对应的一组合理的真值赋值。基于 DPLL 的算法是通过对于这棵二叉树从根节点开始进行深度优先搜索寻找合适的通路，以得到问题的可满足解。主要使用以下两种基本处理策略：

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

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

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



图3-2 DPLL算法搜索树

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

**DPLL( *S*) :**

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

**{**

**while(*S*中存在单子句) {**//单子句传播

**在*S*中选一个单子句*L*；**

**依据单子句规则，利用*L*化简*S*；**

**if *S* = Φ return(TRUE);**

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

**}**//while

**基于某种策略选取变元*v*； //策略对DPLL性能影响很大**

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

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

**}**

DPLL算法中存在变元选择阶段，布尔约束传播阶段和有效回溯阶段。

**3.2.2数独归约成SAT**

cnf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。约束条件按照cnf文件的格式要求写入，具体约束条件包括：

格约束：每个格内只能含有1-9中的一个

行约束：每一行1-9只出现一次

列约束：每一列1-9只出现一次

两对角线约束：对角线1-9只出现一次

3\*3盒子约束：9个3×3的盒子中1-9只出现一次

每个具体的数独游戏，已经填入了部分提示数，每个提示数可表示为一个单子句。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，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

数独约束生成CNF子句集用多重循环结构实现，每个多重循环将生成一种约束的对应子句集。

**3.2.3数独生成的挖洞算法**

本程序数独的生成方法是采用先根据数独规则生成一个符合规则的已解数独，再通过挖洞将需要玩家填充部分的数字移除。

**Generate（各储存初始数独，游戏数独，答案数独的数组）**

**{**

**START：**

**初始化各个数独数组各个格子数为零，默认都是提示数；**

**打乱数组；**

**对角线随机填入1-9，每个数字有且仅有一个；**

**主对角线3个填充3x3的宫格；**

**将数独进行cnf归约；**

**Dpll(); //求解数独**

**goto START; //生成数独无解则重新生成**

**将解填入数独；**

**//挖洞**

**int 挖洞数=81-已知数数量；**

**int 每行挖洞数=remove/9； //先使每行挖洞数一样**

**int 剩下挖洞数=挖洞数-9\*； //每行挖洞数**

**for（遍历9行） //逐行挖洞**

**{**

**while（当前行有未挖的洞）**

**{**

**int 挖洞列=rand()%9+1 //随机选择 一个列**

**if（当前所选单元格未被挖）**

**挖洞，值为0；**

**}**

**}**

**while（有剩下的洞）**

**随机数选未被挖的行列挖洞；**

**Return TRUE;**

**}**

**3.2.4变元选取**

变元选取的策略对于dpll的效率影响极大。本程序采用了三种变元选取策略。

优化前策略：

int varFirst(clauselist \*cnf)

{

clauselist \*pc = cnf;

return abs(pc->nodehead->literal);

}

取第一个子句链表的第一个文字的变元

优化后策略1：

int varMax(clauselist \*cnf)

{

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

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

cnt[i] = 0;

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

for (clausenode\* pn = pc->nodehead; pn; pn = pn->next)

{

if (pn->literal > 0)

cnt[pn->literal]++;

else

cnt[varnum - pn->literal]++;

}

int maxBool, maxTimes = 0;

for (int i = 1; i <= varnum; i++) //选择变元正

if (cnt[i] > maxTimes){

maxTimes = cnt[i];

maxBool = i;

}

if (maxTimes == 1) //变元的反

{

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

if (cnt[i] > maxTimes){

maxTimes = cnt[i];

maxBool = varnum - i;

}

}

free(cnt);

return maxBool;

}

选取总出现次数最多的变元：初始化一个数组cnt，用于统计每个变元出现的次数，初始值为0。遍历整个cnf公式，统计每个变元的出现次数，将负文字映射到对应的正变元。找到出现次数最多的正变元，若有多个出现次数相同的最大值，则选择最小的变元。

优化后策略2：

int varMax2(clauselist\* cnf)

{

double \*weight = (double\*)malloc(sizeof(double) \* (varnum \* 2 + 1));

// 分配内存用于存储每个文字的权重

for (int i = 0; i <= varnum \* 2; i++)// 初始化权重数组中的所有值为0

weight[i] = 0.0;

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

for (clausenode\* pn = pc->nodehead; pn; pn = pn->next)

{

if (pn->literal > 0)

weight[pn->literal] += pow(2.0, (double)(-(pc->listnum)));

else

weight[varnum - pn->literal] += pow(2.0, (double)(-(pc->listnum)));

}

double maxWeight = 0.0;

int maxBool=0;

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

if (weight[i] + weight[i + varnum] > maxWeight)

{

maxWeight = weight[i] + weight[i + varnum];

maxBool = i;

}

if (weight[maxBool] < weight[maxBool + varnum])

maxBool = -maxBool;

free(weight);

return maxBool;

}

选择在短子句中出现次数最多的变元，初始化一个数组weight，用于存储每个文字的权重，初始值为0.0。遍历整个cnf公式，计算每个变元的权重，根据子句长度对权重进行加权，子句越短，权重越大。找到权重之和最大的正变元，若有多个权重和相同的最大值，则选择最小的变元。若正变元的权重小于对应的负变元的权重，则选择对应的负变元。

**4系统实现与测试**

**4.1系统实现**

**4.1.1软件环境**

Windows 规格：Windows 11,version 23H2

编译器：Dev-C++ 5.11

**4.1.2硬件环境**

处理器：13th Gen Intel(R) Core(TM) i7-13700H 2.40 GHz

机带RAM：16.0 GB (15.7 GB 可用)

系统类型：64 位操作系统, 基于 x64 的处理器

根据3.1的设计，用C语言定义各种数据类型；

**4.1.3cnfparser.h模块**

程序代码部分在这里不需要给出来，只需要叙述清楚在系统中包括哪些函数，各函数的说明，如何利用这些函数实现系统各模块的功能，以及函数间的调用关系（可用图表示出来）。

此模块中定义了相关数据结构和外部变量，定义了readcnf函数读取cnf文件并解析，printcnf函数将结果保存到同名res文件中。

程序详见附录。

**4.1.4solver.h模块**

此模块包括函数：

void destroyClause(clauselist\*& cnf)//循环删除子句

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

void removeNode(clausenode \*&head, clausenode \*&nd)//删除节点

int addClause(clauselist \*cl, clauselist \*&cnf)//插入子句

int isUnitClause(clauselist \*cl)//检查是否为单位子句

void simplify(clauselist \*s, clauselist \*&cnf)//根据单位子句来简化 CNF 公式。

int emptyClause(clauselist \*cnf)//检查空子句

void getcopy(clauselist \*&forgery, clauselist\* cnf)//复制一个 CNF 公式的链表结构，以便进行回溯

int varFirst(clauselist \*cnf)//选取第一个变元

int varMax(clauselist \*cnf) //选取总出现次数最多的变元

int varMax2(clauselist\* cnf)  //选取在短子句出现次数最多的变元

int dpll(clauselist \*&list, int \*assignment, int op) //DPLL算法

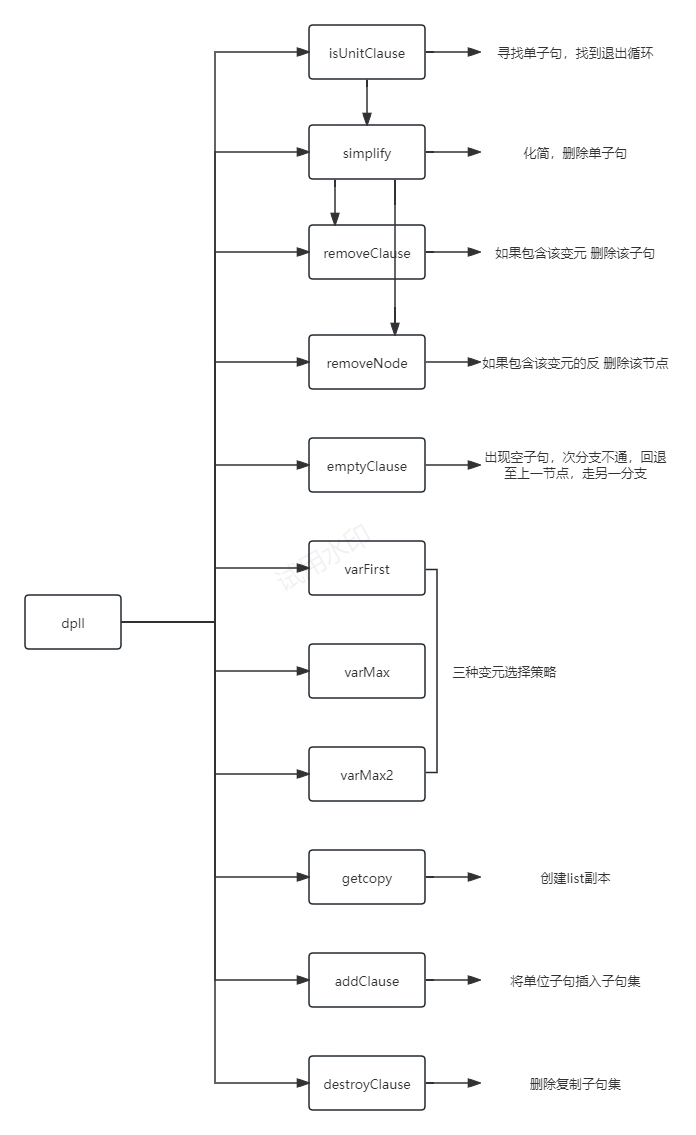


图4-1 dpll函数调用关系

程序详见附录。

**4.1.4X-Sudoku.h模块**

此模块包括函数：

void printmenu();//打印菜单

void Shuffle(int arr[], int n);

//使用随机数来交换数组中的元素，从而实现数组的随机打乱,Fisher-Yates 洗牌算法

int Fill\_Box(int origin[10][10], int player[10][10], int answerecord[10][10], int rowStart, int colStart);//填充一个3X3矩阵 ，数字1-9 各出现一次

int Writecnf(int origin[10][10], int num, char name[]);//将生成的数独归约成cnf

int Generate(int origin[10][10], int player[10][10], int answerecord[10][10], int prompt[10][10], int num, int value[10\*10\*10 + 1]);//随机生成数独

void PrintSudoku(int origin[9 + 1][9 + 1]);//打印数独

int isvalid(int origin[9 + 1][9 + 1], int row, int col, int v);//判断是否符合数独规则

void Play\_Sudoku(int origin[9 + 1][9 + 1], int prompt[9 + 1][9 + 1]);//数独游玩

int Slove(int origin[9 + 1][9 + 1], int value[9 \* 9 \* 9 + 1]);//生成数独答案

void Sudoku();//数独游戏生成菜单

程序详见附录。

**4.2系统测试**



图4-2 初始交互界面

选择cnf文件求解：输入1。



图4-3 cnf文件求解界面



图4-4 cnf文件求解读取文件功能

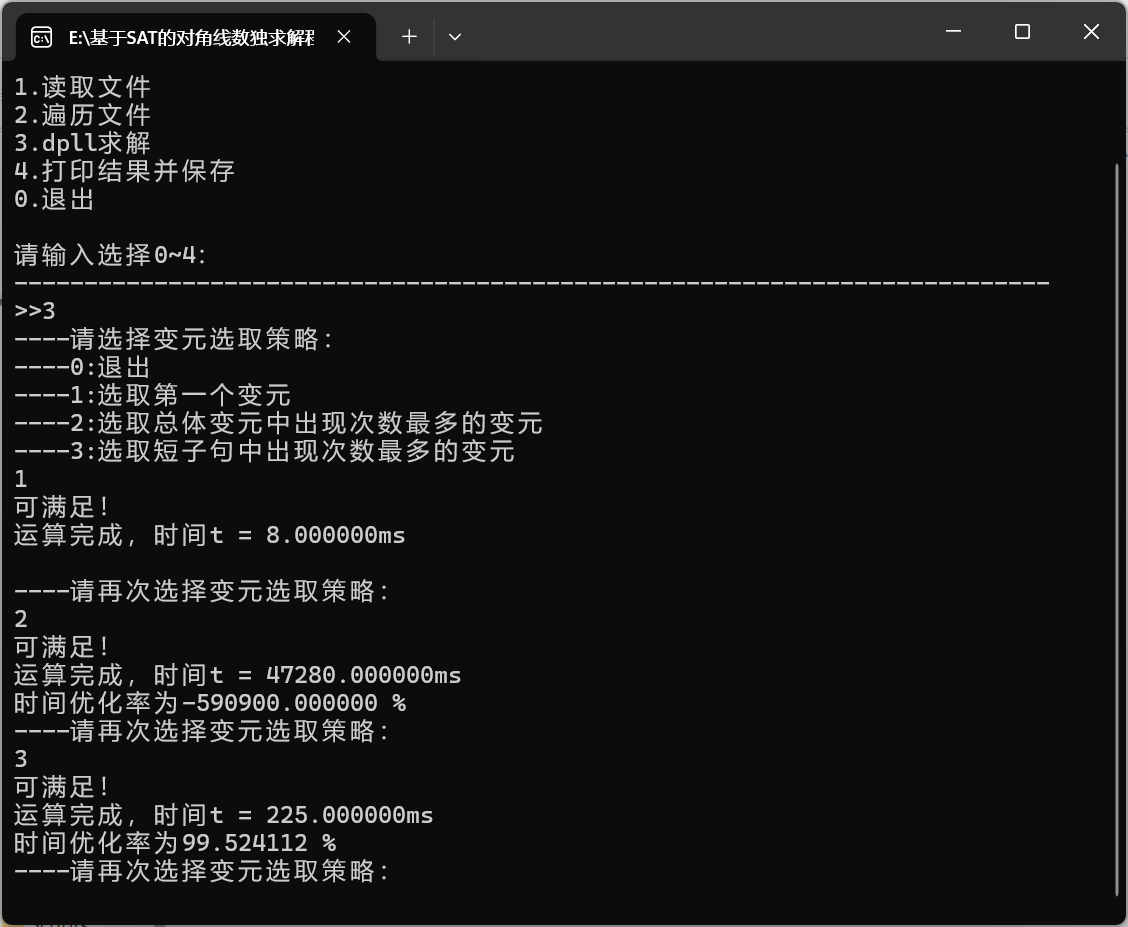


图4-5 cnf文件求解dpll功能

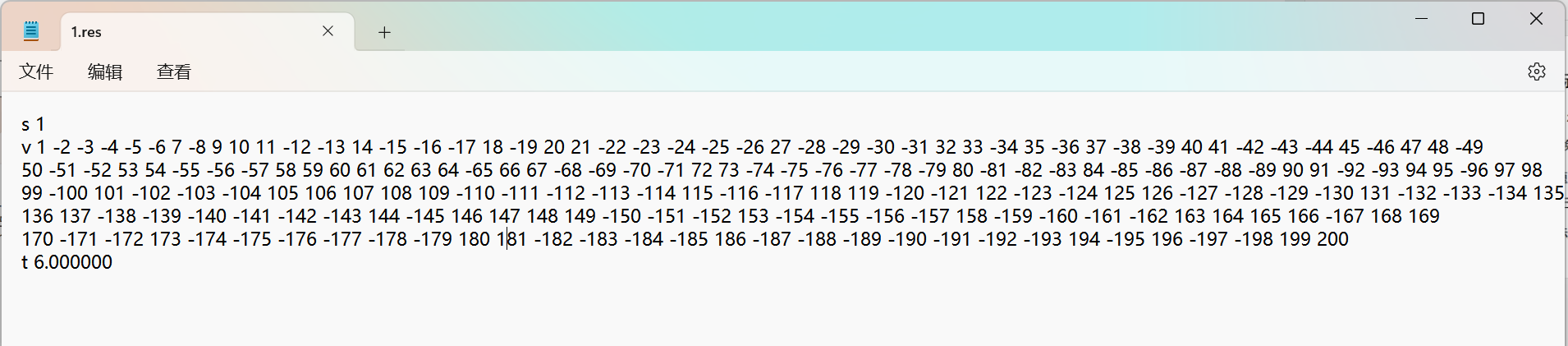


图4-6 cnf文件求解保存功能

根据说明，可在cmd中使用verify.exe验证求解是否正确

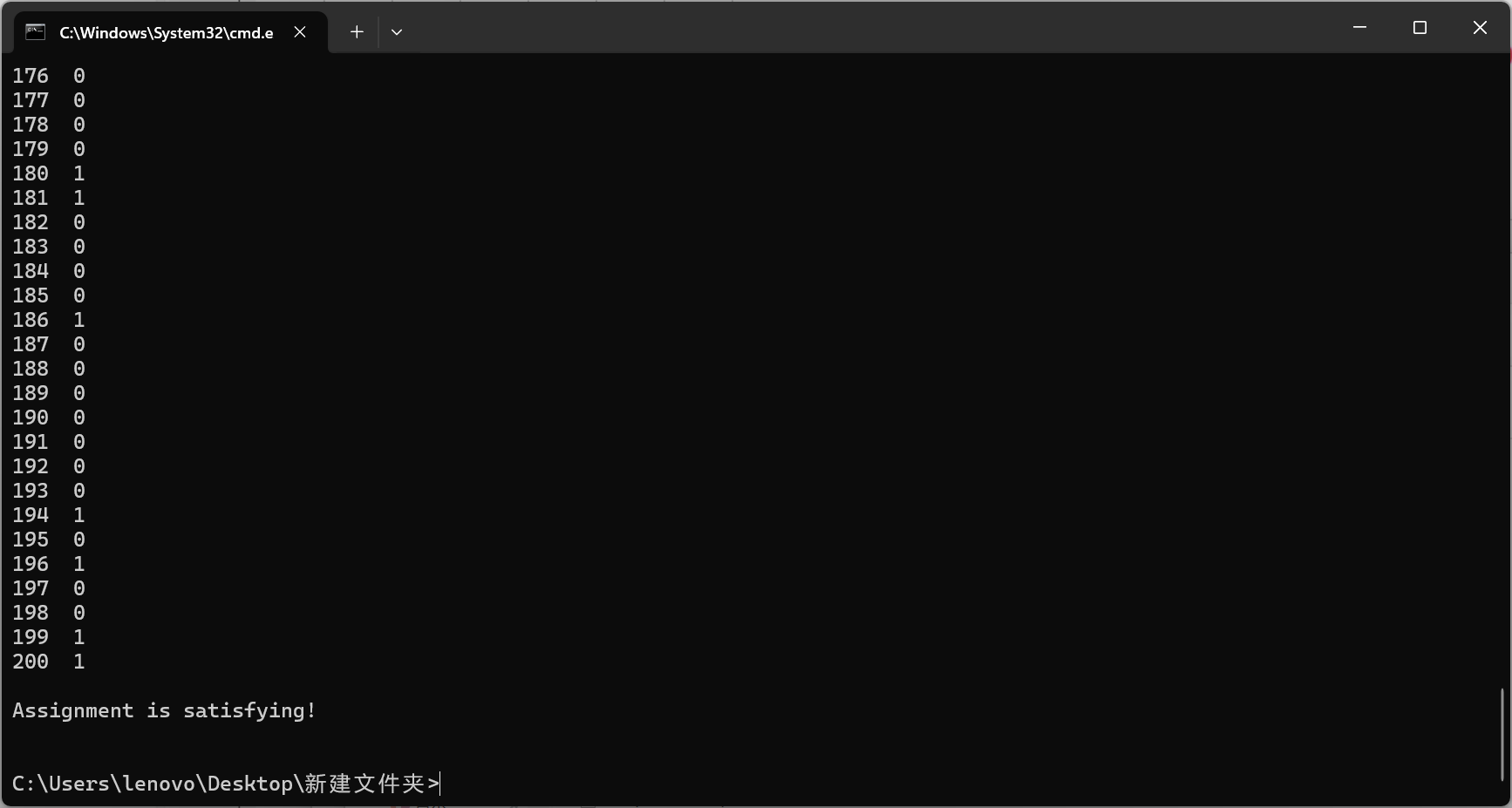
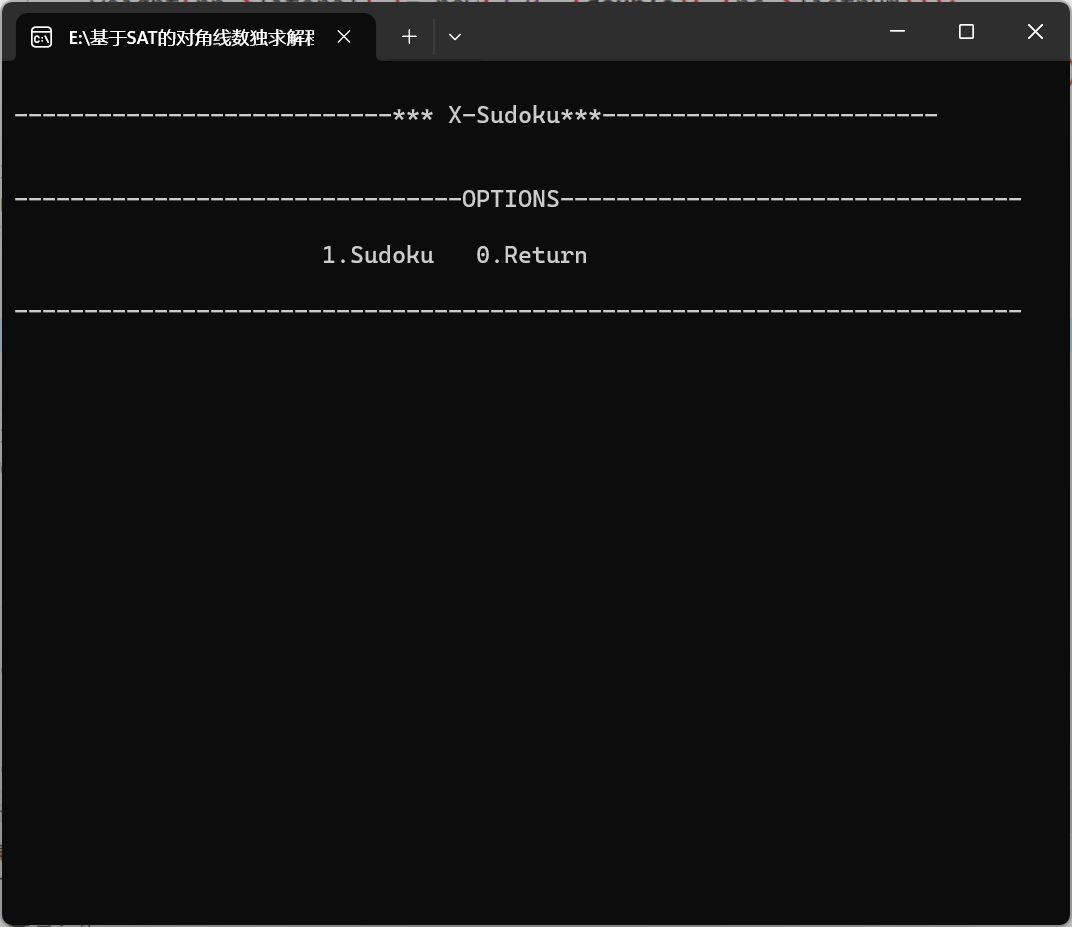


图4-7 cnf文件求解使用verify.exe验证是否正确



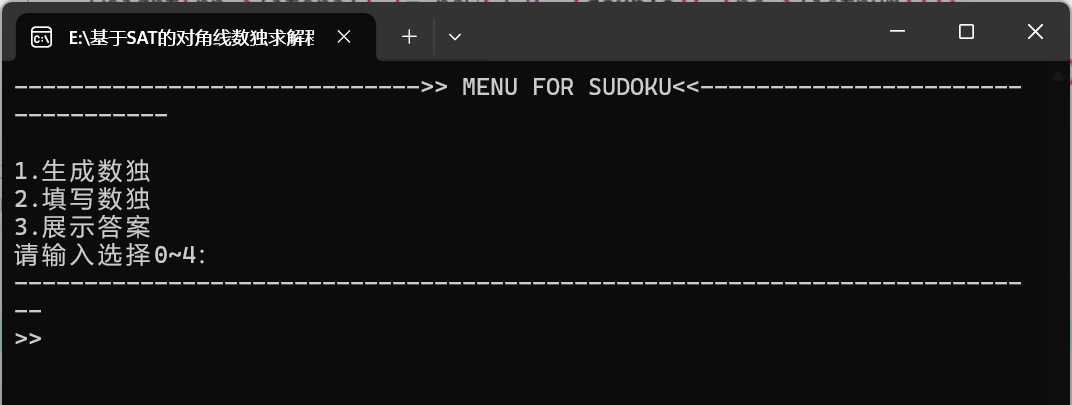


图4-8 数独游戏界面

输入初始数字个数生成数独

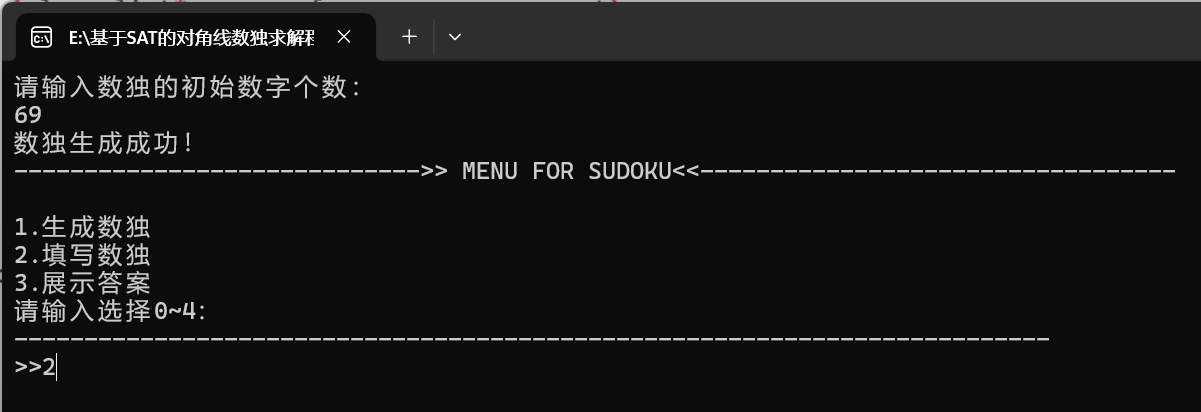


图4-9 数独游戏生成数独



图4-10 数独游戏填写数独

如果输入结果正确，则将输入数字填入数独。



图4-11 数独游戏填写数独输入正确

如果输入数字错误，则提示错误，继续输入。

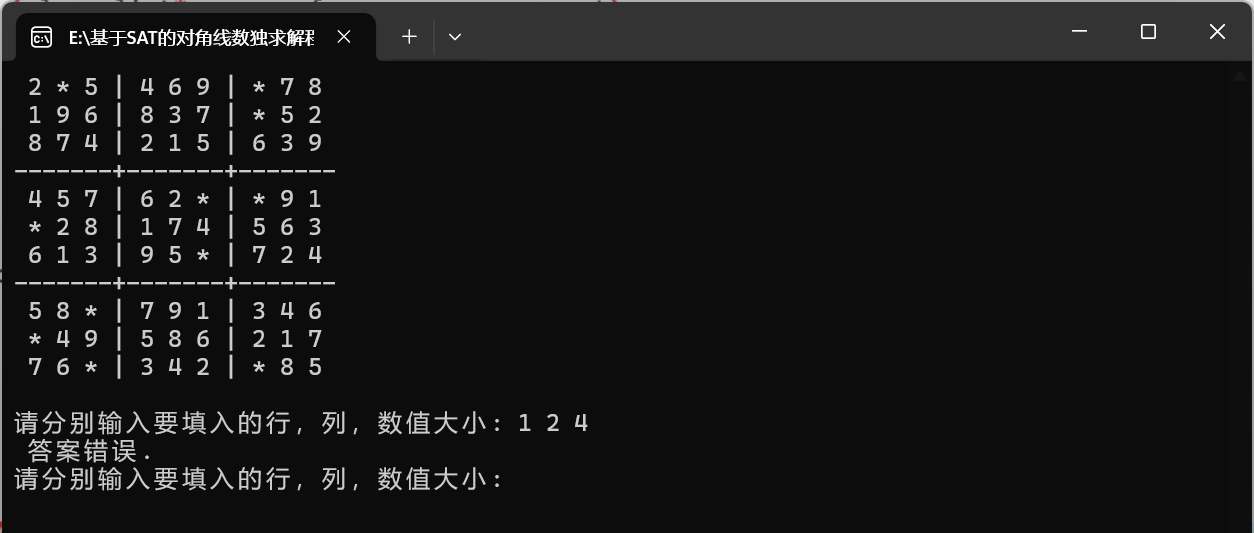


图4-12 数独游戏填写数独输入错误

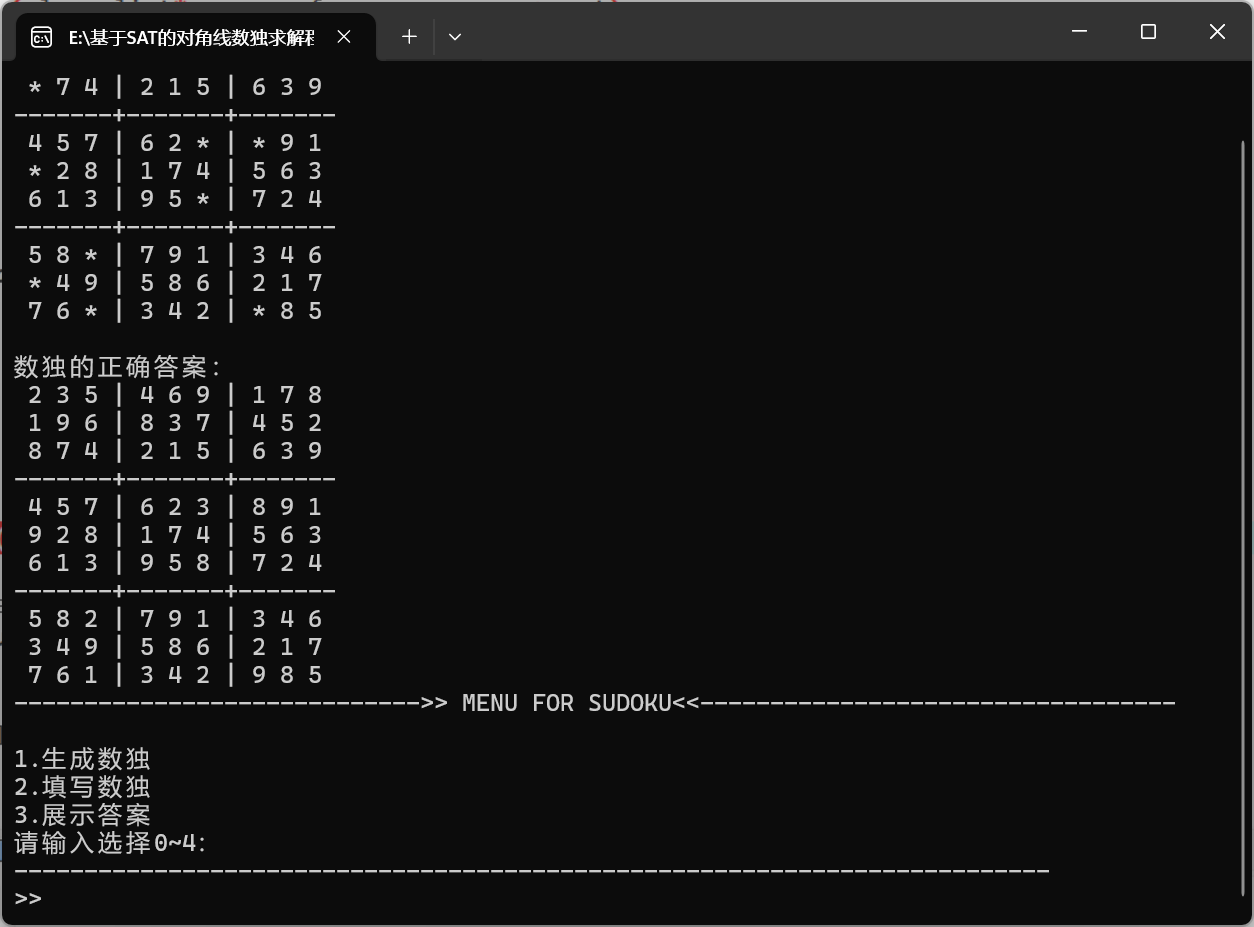


图4-13 数独游戏展示数独答案

接下来展示不同的算例使用不同的变元选择策略所需要的时间（仅举部分算例为例）。

表4-1 不同的变元选择策略所需要的时间（单位：ms）

|  |  |  |  |
| --- | --- | --- | --- |
| 算例 | 策略1 | 策略2 | 策略3 |
| 1.cnf | 11 | 183604 | 832 |
| 2.cnf | 25861 | 7683 | 695 |
| 3.cnf | 62 | 54 | 106601 |
| 4（unsatisfied）.cnf | 9832 | 95819 | 86987 |
| 5.cnf | 176 | 34 | 24 |
| 6.cnf | 90184 | 254 | 399768 |

**5总结与展望**

**5.1全文总结**

本次程序设计我所做的主要工作如下：

1. 阅读了sat问题和求解数独的多篇论文，了解 SAT 的发展历程和数独的背景；
2. 所写程序实现了简易的人机交互功能
3. 实现了一个基于 DPLL 算法的 SAT 求解器，并能够将对角线数独问题归约为sat问题进行求解
4. 为程序选择了合适的数据结构，并在变量决策阶段进行优化，共给出了三种变量决策方法；
5. 设计了测试方案并完成了多个算例的测试；
6. 完成了程序的模块化，实现了各部分的衔接，构建一个简易系统；

（7）对算法优化有了更深刻的了解。

**5.1工作展望**

在今后的研究中，我将围绕着如下几个方面开展工作：

1. 进一步优化 DPLL 算法，提高求解效率，探索更多的变量决策策略和学习方法，以更好地解决复杂的 SAT 问题。
2. 继续完善人机交互功能，提高程序的用户友好性和实用性。
3. 持续学习和探索新的技术和方法，不断提升自身的编程能力和算法理解能力，为程序的持续改进和发展打下坚实的基础。

# 6 体会

在本次课程设计的过程中我阅读了很多篇文献，吸取了前人的经验，自己也在实现算法的过程中提高了编程能力，同时我也认识到了自己的很多不足。

首先就是代码的模块化。此前我并没有养成模块化代码的良好习惯，在拆分对头文件做引用时也遇到了一些问题。在未来的工作中，我将更加注重代码的模块化设计，将程序分解成多个独立的模块，提高代码的可维护性和可扩展性。

其次是dpll算法的实现。通过学习我加深了对DPLL算法的理解，探索更多优化方法，提高算法的效率和求解能力。Dpll中二叉树回溯的应用也让我更加熟练的运用数据结构中学习的知识。

然后是数独归约成sat问题，通过此过程我掌握了如何将一个复杂问题转换成一个我们所熟知问题的能力

最后，挖洞法生成数独的算法也令我对算法学习能力有了较大的提升。

从无到有的完成一个项目令我受益匪浅，成就感满满。

* **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**

cnfparser.h

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <math.h>

#include <malloc.h>

#define TRUE 1

#define FALSE 0

typedef struct clausenode{

int literal;

int score;

struct clausenode\* next;

}clausenode;

typedef struct clauselist{

int listnum;//记录子句中的文字数

clausenode\* nodehead;

struct clauselist\* next;

}clauselist;

int varnum,cnum;//变元数量和子句数量

int \*assignment;

int op = 1, op1 = 1, op2 = 1, op3=1,flag\_cnf, difficulty;

int d;

clauselist \*head=(clauselist\*)malloc(sizeof(clauselist));

clock\_t start, finish;

double time1, time2, rate;

bool res;

char filename[500],wfilename[500];

int readcnf(char \*filename, clauselist \*&head){

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

if(fp){

clauselist \*p = head;

p->nodehead = (clausenode\*)malloc(sizeof(clausenode));

clausenode \*pn = p->nodehead;

p->listnum = 0;

char c;

while((c = getc(fp)) == 'c'){

while ((c = getc(fp)) != '\n') //行首为c时读完此行

continue;

}//跳出循环时行首应为p

if(c != 'p') return FALSE;

char ch[5];

while(fscanf(fp, "%s", ch) != EOF){

if(strcmp(ch, "cnf") == 0){

fscanf(fp, "%d", &varnum); // 变元数

fscanf(fp, "%d", &cnum); // 子句数

break;

}

else return FALSE;

}

int i;

for(i = 0; i < cnum; i++){

int dd;

fscanf(fp, "%d", &dd);

if(dd > varnum||dd < -1\*varnum) return FALSE;

while(dd){//读到0结束

p->listnum++;

pn->literal = dd;

pn->score=0;

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

fscanf(fp, "%d", &dd);

if(dd > varnum||dd < -varnum) return FALSE;

if(dd == 0) pn->next = NULL;

pn = pn->next;

}

if(i == cnum - 1){

p->next = NULL;

break;

}

p->next = (clauselist\*)malloc(sizeof(clauselist));

p->next->listnum = 0;

p->next->nodehead = (clausenode\*)malloc(sizeof(clausenode));

p = p->next;

pn = p->nodehead;

}

fclose(fp);

return TRUE;

}

else return FALSE;

}

int printcnf(int s, char \*wfilename, int \*assignment, double t1, double t2)

{

FILE\* fp;

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

{

printf("文件创建失败!\n");

return FALSE;

}

fprintf(fp, "s %d\nv ", s);

if (s)//如果可满足

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

{

if (assignment[i])

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

else

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

}

fprintf(fp, "\nt %f ", t1);

fclose(fp);

return TRUE;

}

solver.h

#include "cnfparser.h"

void destroyClause(clauselist\*& cnf)

{

clauselist \*ppc, \*pc2;

clausenode \*pn1, \*pn2;

ppc = cnf;

while (ppc)

{

pn1 = ppc->nodehead; //循环删除子句

while (pn1){

pn2 = pn1->next;

free(pn1);

pn1 = pn2;

}

pc2 = ppc->next;

free(ppc);

ppc = pc2;

}

cnf = NULL;

}

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

{

if (cl == cnf) //如果是头子句

cnf = cnf->next;

else

{

clauselist\* ppc = cnf;

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

ppc = ppc->next;

ppc->next = ppc->next->next;

}

clausenode\* pn1, \* pn2;

for (pn1 = cl->nodehead; pn1;){

pn2 = pn1->next;

free(pn1);

pn1 = pn2;

}

free(cl);

cl = NULL;

}

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

{

clausenode\* ppn = head;

if (ppn == nd) //如果要删除头节点

head = head->next;

else

{

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

ppn = ppn->next; //定位节点

ppn->next = ppn->next->next;

}

free(nd);

nd = NULL;

}

int addClause(clauselist \*cl, clauselist \*&cnf)

{

if (cl)//首插法插入单子句

{

cl->next = cnf;

cnf = cl;

return TRUE;

}

return FALSE;

}

int isUnitClause(clauselist \*cl)

{

if (cl->nodehead != NULL) {

if (cl->nodehead->next == NULL)

return TRUE;

}

return FALSE;

}

void simplify(clauselist \*s, clauselist \*&cnf)

{

clauselist\* tmp;

int n = s->nodehead->literal;//要删除的变元的值

for (clauselist\* ppc = cnf; ppc; ppc = tmp)

{

tmp = ppc->next;

for (clausenode\* ppn = ppc->nodehead; ppn; ppn = ppn->next)

{

if (ppn->literal == n)

{

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

break;

}

if (ppn->literal == -n)

{

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

ppc->listnum--;

break;

}

}

}

}

int emptyClause(clauselist \*cnf)

{

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

if (!p->nodehead)

return TRUE;

return FALSE;

}

void getcopy(clauselist \*&forgery, clauselist\* cnf)

{

clauselist\* ppc, \* pc;

clausenode\* ppn, \* pn;

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

forgery->nodehead = (clausenode\*)malloc(sizeof(clausenode));

forgery->next = NULL;

forgery->nodehead->next = NULL;

forgery->listnum = 0;

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

{

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

{

ppc->listnum++;

ppn->literal = pn->literal;

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

ppn->next->next = NULL;

if (pn->next == NULL)

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

}

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

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

ppc->next->next = NULL;

ppc->next->nodehead->next = NULL;

ppc->next->listnum = 0;

if (pc->next == NULL){

free(ppc->next->nodehead);

free(ppc->next);

ppc->next = NULL;

}

}

}

int varFirst(clauselist \*cnf)

{

clauselist \*pc = cnf;

return abs(pc->nodehead->literal);

}

int varMax(clauselist \*cnf) //选取总出现次数最多的变元

{

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

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

cnt[i] = 0;

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

for (clausenode\* pn = pc->nodehead; pn; pn = pn->next)

{

if (pn->literal > 0)

cnt[pn->literal]++;

else

cnt[varnum - pn->literal]++;

}

int maxBool, maxTimes = 0;

for (int i = 1; i <= varnum; i++) //选择变元正

if (cnt[i] > maxTimes){

maxTimes = cnt[i];

maxBool = i;

}

if (maxTimes == 1) //变元的反

{

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

if (cnt[i] > maxTimes){

maxTimes = cnt[i];

maxBool = varnum - i;

}

}

free(cnt);

return maxBool;

}

int varMax2(clauselist\* cnf) //选取在短子句出现次数最多的变元

{

double \*weight = (double\*)malloc(sizeof(double) \* (varnum \* 2 + 1));

for (int i = 0; i <= varnum \* 2; i++)

weight[i] = 0.0;

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

for (clausenode\* pn = pc->nodehead; pn; pn = pn->next)

{

if (pn->literal > 0)

weight[pn->literal] += pow(2.0, (double)(-(pc->listnum)));

else

weight[varnum - pn->literal] += pow(2.0, (double)(-(pc->listnum)));

}

double maxWeight = 0.0;

int maxBool=0;

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

if (weight[i] + weight[i + varnum] > maxWeight)

{

maxWeight = weight[i] + weight[i + varnum];

maxBool = i;

}

if (weight[maxBool] < weight[maxBool + varnum])

maxBool = -maxBool;

free(weight);

return maxBool;

}

//DPLL算法

int dpll(clauselist \*&list, int \*assignment, int op)

{

int flag = 1;

clauselist \*p;

while (flag) {

flag = 0;

p = list;//子句集

while (p && !isUnitClause(p)) //寻找单位子句，找到退出循环

{

p = p->next;

}

if (p != NULL)

{

if (p->nodehead->literal>0)

{

assignment[p->nodehead->literal] = 1;//文字为正

}

else

{

assignment[-p->nodehead->literal] = 0;//文字为负

}

simplify(p, list);//p指向单位元子句，删除单位子句

if (list == NULL) return TRUE;

else if (emptyClause(list)) return FALSE;//出现空子句，次分支不通，回退至上一节点，走另一分支

flag = 1;

}

}

int v;

if (op == 1) v = varFirst(list);

else if (op == 2) v = varMax(list);

else if (op == 3) v = varMax2(list);

clauselist \*SingleClause = (clauselist\*)malloc(sizeof(clauselist)), \*forgery;

SingleClause->nodehead = (clausenode\*)malloc(sizeof(clausenode));

SingleClause->next = NULL;

SingleClause->nodehead->literal = v;

SingleClause->nodehead->next = NULL;

SingleClause->listnum = 1;//被选中的变元组成的单位子句

getcopy(forgery, list);//将list复制到forgery

addClause(SingleClause, forgery);//将单位子句插入子句集

int ans = dpll(forgery, assignment, op);//继续搜素

if (ans == TRUE) return TRUE;

destroyClause(forgery);//删除复制子句集

//进行另一分支v取假值

SingleClause = (clauselist\*)malloc(sizeof(clauselist));

SingleClause->nodehead = (clausenode\*)malloc(sizeof(clausenode));

SingleClause->next = NULL;

SingleClause->nodehead->literal = -v;

SingleClause->nodehead->next = NULL;

SingleClause->listnum = 1;

addClause(SingleClause, list);

getcopy(forgery, list);

ans = dpll(forgery, assignment, op);

destroyClause(forgery);

return ans;

}

X-Sudoku.h

#include "solver.h"

void printmenu()

{

printf("----------------------------->> MENU FOR SUDOKU<<----------------------------------\n");

printf("\n1.生成数独\n2.填写数独\n3.展示答案\n");

printf("请输入选择0~4：\n");

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

}

void Shuffle(int arr[], int n){//使用随机数来交换数组中的元素，从而实现数组的随机打乱

srand(time(NULL)); // 随机选一个数与前面的数交换

for (int i = n - 1; i > 0; i--){

int j = rand() % (i + 1);

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int Fill\_Box(int origin[10][10], int player[10][10], int answerecord[10][10], int rowStart, int colStart){

int n[9 - 3]; // 除了对角线以外的6个格子

int flag1[9] = {0}; // 标记1-9是否已经填入

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

{

flag1[origin[rowStart + i][colStart + i] - 1] = 1; // 对角线的3个格子上的数字标记为已填入

}

int index = 0;

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

{

if (!flag1[i])

n[index++] = i + 1;

}

Shuffle(n, 6); // 打乱数组

index = 0;

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

{

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

{

if (origin[rowStart + i][colStart + j] == 0) // 没有填入

{

origin[rowStart + i][colStart + j] = n[index];

player[rowStart + i][colStart + j] = n[index];

answerecord[rowStart + i][colStart + j] = n[index];

index++;

}

}

}

return TRUE;

}

int Writecnf(int origin[10][10], int num, char name[])

{

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

if (!fp)

{

printf("ERROR!\n");

return FALSE;

}

fprintf(fp, "c %s\n", name);

fprintf(fp, "p cnf 729 %d\n", num + 12654); // 约束条件

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

{

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

{

if (origin[i][j] != 0)

fprintf(fp, "%d 0\n", (i - 1) \* 81 + (j - 1) \* 9 + origin[i][j]);

}

}

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

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int i = 1; i <= 9; i++){//每个格子不能填入两个数字

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

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

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

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

}

}

}

}

for (int i = 1; i <= 9; i++){//行约束

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int i = 1; i <= 9; i++){// 每一行不能填入两个相同的数字

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

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

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

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

}

}

}

}

for (int i = 1; i <= 9; i++){//列约束

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int i = 1; i <= 9; i++){// 每一列不能填入两个相同的数字

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

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

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

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

}

}

}

}

for (int i = 1; i <= 9; i += 3){//3x3宫格约束

for (int j = 1; j <= 9; j += 3){

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

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

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

fprintf(fp, "%d ", ((i + l - 1) \* 9 \* 9 + (j + m - 1) \* 9 + k));

}

}

fprintf(fp, "0\n");

}

}

}

for (int i = 1; i <= 9; i += 3){// 每个3x3宫格不能填入两个相同的数字

for (int j = 1; j <= 9; j += 3){

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

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

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

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

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

}

}

}

}

}

}

for (int i = 1; i <= 9; i++){// 主对角线约束

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

fprintf(fp, "%d ", (j - 1) \* 9 \* 9 + (j - 1) \* 9 + i);

}

fprintf(fp, "0\n");

}

for (int i = 1; i <= 9; i++){// 主对角线不能填入两个相同的数字

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

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

fprintf(fp, "%d %d 0\n", 0 - ((j - 1) \* 9 \* 9 + (j - 1) \* 9 + i), 0 - ((k - 1) \* 9 \* 9 + (k - 1) \* 9 + i));

}

}

}

for (int i = 1; i <= 9; i++){// 副对角线必须填入1-9

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

fprintf(fp, "%d ", (j - 1) \* 9 \* 9 + (9 - j + 1) \* 9 + i);

}

fprintf(fp, "0\n");

}

for (int i = 1; i <= 9; i++){// 副对角线不能填入两个相同的数字

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

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

fprintf(fp, "%d %d 0\n", 0 - ((j - 1) \* 9 \* 9 + (9 - j + 1) \* 9 + i), 0 - ((k - 1) \* 9 \* 9 + (9 - k + 1) \* 9 + i));

}

}

}

fclose(fp);

return TRUE;

}

int Generate(int origin[10][10], int player[10][10], int answerecord[10][10], int prompt[10][10], int num, int value[10\*10\*10 + 1])

{

char name[100] = "Sudoku.cnf";

START:

srand(time(NULL));

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

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

{

origin[i][j] = 0;

player[i][j] = 0;

answerecord[i][j] = 0;

prompt[i][j] = TRUE; // 默认全为提示数字

}

int n[9];

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

n[i] = i + 1;

Shuffle(n, 9); // 打乱数组

int index = 0;

for (int i = 1; i <= 9; i++) // 对角线填入1-9

{

origin[i][i] = n[index];

player[i][i] = n[index];

answerecord[i][i] = n[index];

index++;

}

for (int i = 1; i <= 9; i += 3)

{

Fill\_Box(origin, player, answerecord, i, i); // 主对角线3个填充3x3的宫格

}

Writecnf(origin, 27, name); // 将数独约束条件写入文件

clauselist \*p=(clauselist\*)malloc(sizeof(clauselist));

readcnf(name,p);

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

value[i] = FALSE;

if (dpll(p, value, 3) == FALSE) // 求解数独

goto START;

// 将DPLL的结果填入数独

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

{

if (value[i] == TRUE)

{

int row = (i - 1) / (9 \* 9) + 1;

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

int v = (i - 1) % 9 + 1;

origin[row][col] = v;

player[row][col] = v;

answerecord[row][col] = v;

}

}

int remove = 81 - num;// 挖洞

int single = remove / 9;

int res = remove - 9 \* single;

// int c[SIZE]={9-single};

for (int row = 1; row <= 9; row++) // 每行挖single个

{

int s = single;

while (s)

{

int col = rand() % 9 + 1;

if (origin[row][col] != 0) // 没有被挖

{

origin[row][col] = 0;

player[row][col] = 0;

answerecord[row][col] = 0;

prompt[row][col] = FALSE;

s--;

}

}

}

while (res) // 挖剩下的

{

int row = rand() % 9 + 1;

int col = rand() % 9 + 1;

if (origin[row][col] != 0)

{

origin[row][col] = 0;

player[row][col] = 0;

answerecord[row][col] = 0;

prompt[row][col] = FALSE;

res--;

// c[row-1]--;

}

}

return TRUE;

}

void PrintSudoku(int origin[9 + 1][9 + 1]){

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

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

if (origin[i][j] == 0) // 未填入

printf(" \*");

else // 已填入

printf("%2d", origin[i][j]);

if (j % 3 == 0 && j != 9) // 每3列打印一个竖线

printf(" |");

}

printf("\n");

if (i % 3 == 0 && i != 9) // 每3行打印一个横线

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

}

}

int isvalid(int origin[9 + 1][9 + 1], int row, int col, int v){

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

{

if (origin[row][i] == v || origin[i][col] == v) // 行或列有重复

return FALSE;

}

int startRow = (row - 1) / 3 \* 3 + 1; // 宫格的起始行

int startCol = (col - 1) / 3 \* 3 + 1; // 宫格的起始列

for (int i = startRow; i < startRow + 3; i++)

{

for (int j = startCol; j < startCol + 3; j++)

{

if (origin[i][j] == v) // 宫格内有重复

return FALSE;

}

}

// 检查主对角线

if (row == col)

{

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

{

if (origin[i][i] == v)

return FALSE;

}

}

// 检查副对角线

if (row + col == 9 + 1)

{

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

{

if (origin[i][9 - i + 1] == v)

return FALSE;

}

}

return TRUE;

}

void Play\_Sudoku(int origin[9 + 1][9 + 1], int prompt[9 + 1][9 + 1])

{

system("cls"); // 清屏

PrintSudoku(origin); // 打印初始数独

printf("\n");

while (1)

{

int row, col, v;

printf("请分别输入要填入的行，列，数值大小: ");

scanf("%d", &row);

if (row == 0) // 退出

{

system("cls");

return;

}

scanf("%d%d", &col, &v);

if (row < 1 || row > 9 || col < 1 || col > 9 || v < 1 || v > 9) // 输入不合法

{

printf(" 输入无效\n");

continue;

}

if (prompt[row][col]) // 是提示数

{

printf("输入的是提示数\n");

continue;

}

if (!isvalid(origin, row, col, v)) // 不符合数独规则

{

printf(" 答案错误.\n");

continue;

}

else // 符合数独规则

{

origin[row][col] = v;

system("cls");

PrintSudoku(origin); // 打印新数独

printf("\n");

}

}

}

int Slove(int origin[9 + 1][9 + 1], int value[9 \* 9 \* 9 + 1])

{

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

{

if (value[i] == TRUE)

{

int row = (i - 1) / (9 \* 9) + 1;

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

int v = (i - 1) % 9 + 1;

origin[row][col] = v;

}

}

return TRUE;

}

void Sudoku()

{

system("cls");

int num;//提示数的个数

int prompt[10][10]; //记录是否为提示数字

int origin[10][10];//生成的初始数独

int player[10][10]; //用来玩的数独

int answerecord[10][10]; // 保存答案的数独

int value[10 \* 10 \* 10 + 1]; // 记录DPLL的结果

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

value[i] = FALSE;

int opx = 1;

int flag = 0; // 是否生成数独

while (opx)

{

printmenu();

scanf("%d", &opx);

system("cls");

switch (opx){

case 1:{

printf("请输入数独的初始数字个数：\n");

scanf("%d", &num);

while (num < 18 || num > 81){

printf("错误！请重新输入：");

scanf("%d", &num);

}

if (Generate(origin, player, player, prompt, num, value))

{

printf("数独生成成功！\n");

flag = 1; // 生成成功

}

else

printf("数独生成失败！\n");

break;

}

case 2:

{

if (flag)

{

Play\_Sudoku(player, prompt);

printmenu(); // 每次玩完跳转回来重新打印菜单

}

else

printf("请先生成数独\n");

break;

}

case 3:

{

if (flag)

{

PrintSudoku(origin); // 打印原始数独

printf("\n");

if (Slove(answerecord, value)) // 求解数独

{

printf("数独的正确答案：\n");

PrintSudoku(answerecord); // 打印答案

}

else

printf("数独无解\n"); // 无解

}

else

printf("请先生成数独\n");

break;

}

case 0:

{

system("cls"); // 退出时清屏

break;

}

}

}

}

display.cpp

#include "X-Sudoku.h"

int main()

{

int op = 1, op1 = 1, op2 = 1, op3=1,flag\_cnf, difficulty;

int d;

clauselist \*head=(clauselist\*)malloc(sizeof(clauselist));

clock\_t start, finish;

double time1, time2, rate;

bool res;

char filename[500],wfilename[500];

while (op)

{

system("cls");

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

printf("\n1.cnf文件求解\n2.数独游戏\n0.退出 \n\n");

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

printf("请输入选择0~2：\n");

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

scanf("%d", &op);

switch (op)

{

case 1:

flag\_cnf = 1, op1 = 1;

while (op1)

{

system("cls");

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

printf("\n1.读取文件\n2.遍历文件\n3.dpll求解\n4.打印结果并保存\n0.退出\n\n");

printf("请输入选择0~4：\n");

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

scanf("%d", &op1);

switch (op1)

{

case 1: {

printf("请输入测试文件地址:\n");

scanf("%s", filename);

d=readcnf(filename,head);

if (d == TRUE){

printf("读取成功！\n");

printf("变元数量为：%d 子句数量为：%d",varnum,cnum);

}

else

printf("读取失败!\n");

getchar();

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

getchar();

break;}

case 2:{

if (d){

printf("文件中有如下子句：\n");

for(clauselist \*lp=head;lp;lp=lp->next){

for(clausenode \*np=lp->nodehead;np;np=np->next){

printf("%d ",np->literal);

}

printf("\n");

}

}

else

printf("尚未读取！\n");

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

getchar(), getchar();

break;}

case 3:{

if (d==FALSE)

printf("尚未读取!\n ");

else

{

printf("----请选择变元选取策略：\n");

printf("----0:退出\n");

printf("----1:选取第一个变元\n");

printf("----2:选取总体变元中出现次数最多的变元\n");

printf("----3:选取短子句中出现次数最多的变元\n");

scanf("%d", &op3);

assignment = (int\*)malloc(sizeof(int) \* (varnum + 1));

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

assignment[i] = 1;

start = clock();

res = dpll(head, assignment, op3);

finish = clock();

if (!res) printf("不可满足！\n");

else printf("可满足！\n");

time1 = (double)(finish - start);

printf("运算完成，时间t = %lfms\n", time1);

time2=time1;

while(op3)

{

time1 = time2;

printf("\n----请再次选择变元选取策略：\n");

scanf("%d", &op3);

if(op3 == 0) break;

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

assignment[i] = 1;

start = clock();

res = dpll(head, assignment, op3);

if (!res) printf("不可满足！\n");

else printf("可满足！\n");

finish = clock();

time2 = (double)(finish - start);

printf("运算完成，时间t = %lfms\n", time2);

if(time1 == 0 || time1 == time2 || time2 == 0) printf("时间优化率为 0 %%\n");

else if(time1 < time2) printf("时间优化率为-%lf %%", (100 \* 1.0 \*(time2 - time1)) / time1);

else printf("时间优化率为%lf %%", (100 \* 1.0 \*(time1 - time2)) / time1);

}

}

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

getchar(), getchar();

break;}

case 4:{

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

{

if (assignment[i])

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

else

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

}

if (!d)

printf("尚未读取!\n ");

else{

printf("请输入保存文件地址:\n");

scanf("%s", wfilename);

if (printcnf(res, wfilename, assignment, time1, time2))

printf("结果已保存\n");

else

printf("结果保存失败!\n");

}

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

getchar(), getchar();

break;}

case 0:

break;

}

}

break;

case 2:

op2 = 1;

while (op2)

{

system("cls");

printf("\n---------------------------\*\*\* X-Sudoku\*\*\*------------------------\n\n");

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

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

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

scanf("%d", &op2);

switch (op2)

{

case 1:

Sudoku();

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

getchar(), getchar();

break;

default:

break;

}

}

break;

case 0:

break;

}

}

return 0;

}

**附录二**

程序使用说明：

在初始界面中可输入1选择cnf文件求解，在cnf求解菜单中可对cnf进行操作求解：输入1读取文件，输入2遍历输出cnf文件中的子句，输入3求解，输入4输出结果并保存到同名文件中，输入0返回到初始界面。

在初始界面中可输入2选择数独游戏，输入1可游玩，在suduko菜单下输入1并输入18到81之间的一个整数作为初始数字个数，输入2可填写生成的数独，按照行列数字（ijk）格式填写，输入3可展示答案，输入0返回到初始界面.

输入0退出程序。