

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 综合程序设计课程设计**

**专业班级： 计算机科学与技术1707班**

**学 号： U201714808**

**姓 名： 陈绿然**

**指导教师： 祝建华**

**报告日期： 2019年4月6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：**将数独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

**参考文献**

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

[2]Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

[4]Carsten Sinz.Visualizing SAT Instances and Runs of the DPLL Algorithm.J Autom Reasoning (2007) 39:219–243

[5] 360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic forProgramming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

[7]Ins Lynce and Jol Ouaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

目录

[**1引言** 1](#_Toc5639113)

[1.1课题背景与意义 1](#_Toc5639114)

[1.2国内外研究现状 1](#_Toc5639115)

[1.3课程设计的主要研究工作 2](#_Toc5639116)

[**2系统需求分析与总体设计** 3](#_Toc5639120)

[2.1系统需求分析 3](#_Toc5639121)

[2.2系统总体设计 3](#_Toc5639122)

[**3系统详细设计** 6](#_Toc5639123)

[3.1有关数据结构的定义 6](#_Toc5639124)

[3.2主要算法设计 8](#_Toc5639125)

[**4系统实现与测试** 17](#_Toc5639126)

[4.1系统实现 17](#_Toc5639127)

[4.2系统测试 17](#_Toc5639128)

[4.2.1各函数功能测试 17](#_Toc5639129)

[4.2.2 SAT算例测试 21](#_Toc5639130)

[4.2.3 数独游戏测试 25](#_Toc5639131)

[4.3系统测试结论 27](#_Toc5639132)

[**5总结与展望** 28](#_Toc5639133)

[5.1全文总结 28](#_Toc5639134)

[5.1.1工作总结 28](#_Toc5639135)

[5.1.2 自我反思和总结 28](#_Toc5639136)

[5.2工作展望 29](#_Toc5639137)

[**6体会** 30](#_Toc5639138)

[**参考文献** 31](#_Toc5639139)

[**附录** 32](#_Toc5639140)

# 1引言

## 1.1课题背景与意义

作为计算机科学与技术、信息安全与物联网专业大二学生，在前三个学期已经学习了C语言程序设计、数据结构两门面向编程知识与技术的基础理论课，以及C语言程序设计实验、数据结构实验两门编程实践课程，在此基础上，不仅具有较为系统性的C语言、常用数据结构基本知识，而且具有初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。但两门实验课仍属于对基本编程模型与技术的验证性训练，而“综合程序设计”课程设计正是使大家从简单验证到综合应用，为后续学习与进行计算机系统编程打下坚实的基础，该课题为基于SAT求解的数独游戏程序，其关键点就是如何实现SAT问题求解，要求算法都以DPLL算法为基础，设计良好的数据结构，并尽可能高效的完成求解过程，这就涉及到算法优化的问题，通过这次实践相信自己能够更好的掌握对链表、栈的数据结构的设计以及对算法的思考和理解。

## 1.2国内外研究现状

SAT问题是计算机科学领域和人工智能领域重要研究的课题，最经典求解SAT问题的完备算法是DPLL算法，其他算法均是对其的改进，自1996年以后，涌现了众多高效的SAT算法如MINISAT等，采用新的数据结构、新的变量决策策略或者新的算法实现。国内也涌现了许多高效的求解算法，如2000年金人超和黄文奇提出的并行的Solar算法，2002年张德富提出的模拟退火算法。

关于数独游戏，据著名的动游戏开发商Astraware Ltd.预计，移动数独游戏的版本多达几十种，Palm和Windows Mobile设备版本的数独游戏就各有20种左右。Sudokumo推出的移动数独游戏，能够下载到大多数手机中。这家位于英国的游戏软件公司表示，已经在全球卖出了7500套数独游戏，而且来自用户的兴趣还在增加。

## **1.3课程设计的主要研究工作**

首先要设计问题中变元、文字、子句、公式数据类型以及有效的物理存储结构，分析这些数据的逻辑关系及其各种基本运算，建立相应的抽象数据类型，设计其物理结构。

然后实现这些数据间的一些基本运算。包括打开文件(OpenFile)、增加子句(addClause)、删除子句(removeClause)、删除文字(removeLiteral)、判断是否为单子句(isUnitClause)、判断是否含有空子句(EmptyClause)、寻找单子句(findunit)、寻找最短子句(optimize)、遍历公式(display)、保存文件(savefile)等，依据已建立的链表结构，实现这些基本运算的函数，可在后面的DPLL算法求解SAT问题中灵活调用，根据DPLL算法写出判断一个公式是否能满足的函数，并把所赋的值保存在结果数组里。

按照要求，最终需要能够实现读取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，并遍历结构以验证解析的正确性。对每个算例求解出满足性、求解时间、结果赋值。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，计算出性能优化率。把数独游戏转化为SAT问题，利用SAT求解器进行问题的求解，形成有一定的用户交互性的数独游戏。

# **2系统需求分析与总体设计**

## 2.1系统需求分析

我设计的这个程序，主要就是面向SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），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，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。

基于求解SAT问题，可以设计经典的数独游戏，作为一款智力游戏，可以开发用户智力，培养用户的动脑能力。

## 2.2系统总体设计

我根据设计问题的功能要求，以一个程序处理流程图(图2-1)展现程序的各个功能模块，红色部分为程序中实现基于DPLL的SAT求解相关功能模块，蓝色部分是数独游戏生成、转化、求解等处理模块。

本系统实现了程序模块化，程序源代码进行模块化组织。主要模块包括：

1.主控、交互与显示模块（display）

2.CNF解析模块（cnfparser）

3.核心DPLL模块( solver)

4.数独模块,包括数独生成、归约、求解(Sudoku)

5.CNF公式的内部存储结构

完整的系统模块结构图，如图2.1所示。



图2-1 程序模块化流程图

该系统开始界面为一个选项，可以选择1.求解SAT或者2.数独游戏。

当输入1后，界面会显示一个菜单，其中有包括DPLL算法求解SAT问题在内的八种操作0-7，分别为:

1.OpenFile(打开、读取cnf文件，并建立公式的内部表示，并遍历物理结构，显示出变元数、子句数和每个子句)

2.DPLLsolve(核心算法，求解SAT的关键，显示是否可满足、满足时各变元取值、求解所用的时间)

3.addClause(增加子句)

4.removeclause(删除子句)

5.simplify(根据输入的文字化简公式)

6.display(遍历输出各子句)

7.removeLiteral(删除文字)

0.exit(退出菜单)

当输入为0时，进入数独游戏，界面随机显示一个数独棋盘，并调用函数transformcnf(int s[9][9])将数独格局转化为SAT问题，生成在一个cnf文件中，随即打开文件调用SAT求解器中的DPLL函数对文件进行求解，最后可按enter键查看该数独游戏的答案，同样也是以9\*9的棋盘形式给出。

# 3系统详细设计

## 3.1有关数据结构的定义

数据包括：文字中数值data，文字literal，子句clause，相同文字集same\_literal，结果数组result。

其中，数值为Elemtype类型，即为整型；文字为结构类型，包括数据data、指向下一文字的指针next;子句为结构类型，包括该子句所含文字个数num、指向下一子句的指针next、指向该子句中第一个文字的指针；相同文字集为结构类型，包括数据data、所含相同文字的个数num。

有关数据类型及变量的定义的源代码：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

typedef int status;

typedef int ElemType;

int bool\_num,clause\_num;//定义变元数量和子句数量

typedef struct literal{

ElemType data;

struct literal \*next;

}literal;

typedef struct clause{

struct clause \*next;

int num;//指示这条子句文字数，若为1则说明是单子句，头结点的num指示子句的个数

struct literal \*HeadNode;

}clause,\*Root;

typedef struct same\_literal{

ElemType data;

int num;

}same\_literal;

int result[500];

所有数据类型可用表3-1所表示。

表3-1 数据类型表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据 | 数值data | 文字literal | 子句clause | 相同文字集same\_literal | 结果数组Result |
| 所包含的数据项及其数据类型 | Elemtype类型（即int型） | data;  Elemtype型 | num;  int型 | data;  Elemtype型 | int型 |
|  | next；  literal型指针 | next;  clause型指针 | num;  int型 |  |
|  |  | HeadNode;  literal型指针 |  |  |

子句表示为由文字构成的链表，而整个公式又是由子句构成的链表，数据在系统中的关联方式如图3-1所示。根据图3-1可以直观地看出，笔者设计的数据结构为一链表，其中每个结点都为指向一个新的单链表的头指针，指向一个子句的第一个文字，并且标记了该子句所含的文字个数。

NULL

data

next

num

next

HeadNode

num(字句个数)

next

NULL

data

NULL

num

next

HeadNode

data

next

num

next

HeadNode

data

NULL

data

NULL

NULL

data

next

num

next

HeadNode

num(字句个数)

next

NULL

data

NULL

num

next

HeadNode

data

next

num

next

HeadNode

data

NULL

data

NULL

图3-1 数据间的关联图示

## **3.2主要算法设计**

1.OpenFile (Root &L)功能及算法

初始条件：线性表未建立。

操作结果：建立单链表，将所需要读取的文件打开并将公式解析到所建立的线性表中，最后遍历内部结构，输出每个子句。

算法设计：用不同的文件读取函数读相应的cnf文件内容，用全局变量bool\_num和clause\_num分别保存变元数和子句数，用两个while循环分别控制字句个数和每个子句中的文字个数。每读取一个文字内容，就动态分配一个存储空间，每个子句链表以0结束，公式以clause\_num结束。流程图如图3-2所示。

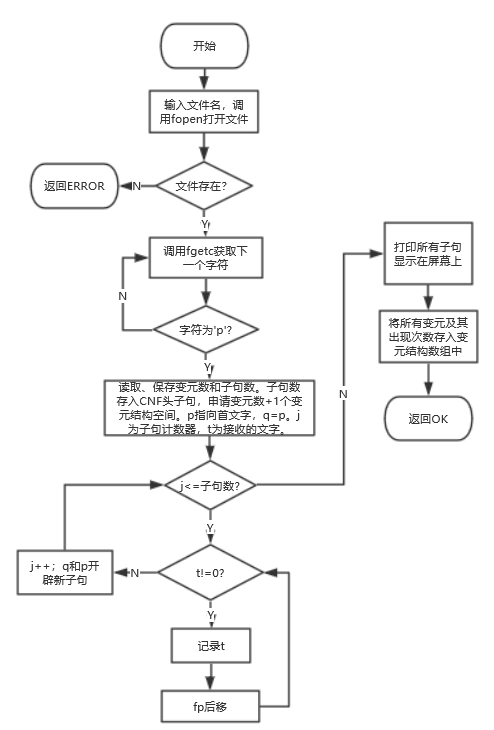


图3-2打开文件建立内部结构算法流程图

2.addClause (Root &L)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：在链表头部插入一个新的子句。

算法设计：先根据所输入的文字个数动态分配存储空间，建立文字的链表，再利用首插法，将链表插入子句集链表中，最后将子句集链表的L->num加一。

3.removeClause (Root &L,ElemType e)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：删除包含文字e的子句

算法设计：使用一前一后两个指针将公式的物理结构进行遍历，遇到文字为e的结点，释放该子句每个节点的存储空间，并使上一子句的指针next指向下一子句。最后将子句集链表的L->num减一。流程图如图3-3所示。

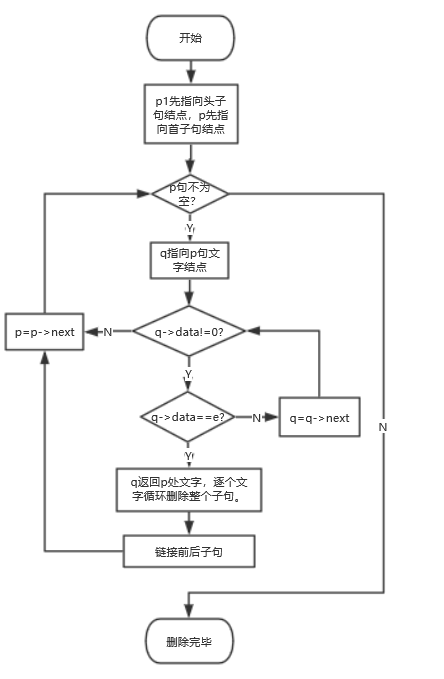


图3-3 removeClause删除包含文字e的子句流程图

4.removeLiteral(Root &L,ElemType e)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：删除每个子句中的文字e

算法设计：使用一前一后两个指针将公式的物理结构进行遍历，遇到文字为e的结点，释放该结点空间，并使上一文字的next指针指向下一文字，每删除一个文字，使该子句的num减一。删除指定文字e的算法流程图如图3-4所示。

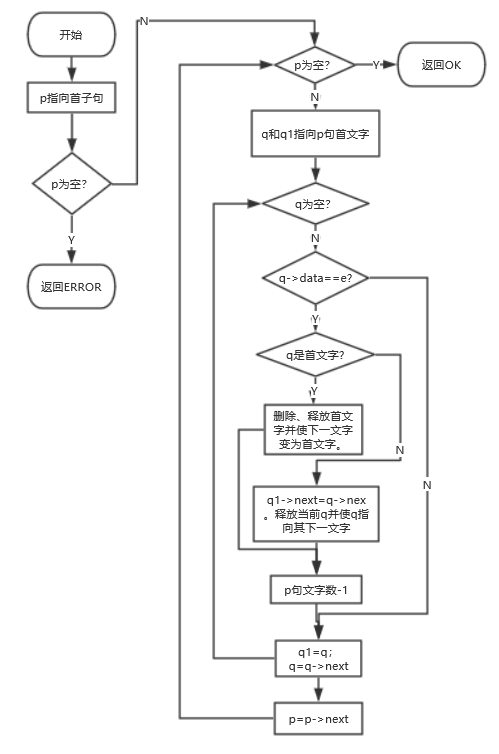


图3-4 removeLitera删除所有子句中的文字e算法流程图

5. absoluteValue(ElemType e)功能及算法

初始条件：存在任意一个ElemType类型的数据e。

操作结果：返回它的绝对值。

算法设计：判断参数e的符号，若为正数，返回e；若为负数，返回-e。

6. simplify(Root L,ElemType e)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：复制一个相同的公式，删除新公式中所有含e的子句和含有-e的文字，最后返回新公式的头指针。

算法分析：首先建立一个新链表的头结点，依次读取现有链表的文字结点，遍历整个公式，动态分配新链表的存储空间，使新链表每个文字的data与对应的现有公式文字结点的data相等，直到NULL为止。然后调用removeClause(newL,e);

removeLiteral(newL,-e); 即删除了新公式中所有含e的子句和含有-e的文字，最后把e赋给结果数组中以e的绝对值为下标的值。最后返回新公式的头指针。

7. isUnitClause(Root &p)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。并有一个指针p指向某一子句。

操作结果：判断该子句是否为单子句。

算法设计：若p->num为一，返回OK。

8. findunit(Root L)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：判断该公式是否含有单子句，若有，返回指向该子句的指针，若无，返回NULL。

算法设计：用while循环遍历公式，若遇到某一子句的num为一，返回该指针，若遍历到子句集末尾，则说明无单子句，返回NULL。

9. optimize(Root L)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：找到最短子句（优化算法中遇到）

算法设计：先判断L的num值，若为零，说明是空子句集，返回ERROR；若为1，返回唯一子句的第一个文字；若其他，用两个指针遍历公式，其中p指向最短子句，若tp->num<p->num，则p=tp，最后返回p。优化算法寻找最短子句的流程图如图3-5所示。

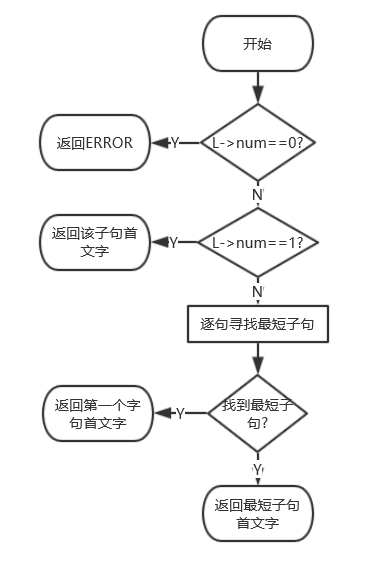


图3-5 optimize寻找最短子句算法流程图

10. EmptyClause(Root L)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：判断子句集中是否含有空子句，若有，说明公式不可满足，返回ERROR，若无，返回OK。

算法设计：遍历每个子句，若遇到p->num为零，返回ERROR，若遍历结束到NULL，返回OK。

11.DPLL(Root L)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：求解公式（可满足/不可满足），并将结果保存在结果数组中。

算法设计：算法基于单子句传播与分裂策略的，运用了递归思想。首先判断头指针指向的是否为NULL，若是，说明公式已经为一个空子句集，返回TRUE，调用函数EmptyClause(L)，判断公式内是否含有空子句，若返回值为ERROR，返回FALSE，表示不可求解。在循环中调用函数findunit(L)，寻找公式中所有单子句，用w获取单子句的文字的值，并依据单子句规则，对公式进行简化，具体做法是removeClause(L,w);removeLiteral(L,-w);然后把w赋给结果数组中以w的绝对值为下标的值，处理后，再重复上面的判断，即若公式为空子句集，返回TRUE，若公式含有空子句，返回FALSE，将所有的单子句处理后，选择第一个子句的第一个文字，用v获取其data的值（优化算法则是调用optimize(L)找到最短子句并用v获取其第一个文字的data的值），递归调用DPLL算法本身，参数为简化公式函数simplify(L,v)的返回值，即复制后进行简化的新公式，若调用后的返回值为TRUE，返回TRUE，若该值为FALSE，说明v的取值错误，递归调用DPLL算法本身，参数为简化公式函数simplify(L,-v)的返回值。

12. savefile(Root L,int t,int r)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构，求解了内部的SAT问题后得到了结果r以及运行时间t。

操作结果：将结果保存在了一个以res为后缀名的文件中。

13. display(Root &L)功能及算法

初始条件：已打开一个cnf文件并建立了与之对应的物理存储结构。

操作结果：将公式内部解析，遍历子句集并显示。

算法设计：用while循环遍历整个链表，输出每个文字的值。

14. transformcnf(int s[9][9])功能及算法

初始条件：已存在一个9行9列的二维数组。

操作结果：按照标准cnf文件格式，将二维数组转化为SAT求解的模型，存在一个cnf文件中。

算法设计：将数独每个位置的赋值以一个三位数表示，百位表示第几行，十位表示第几列，个位表示填入的数字（1-9），符号表示该填入的数字是否为真。根据数独游戏的条件：每一行、每一列填入的九个数字彼此不相等，每个3\*3九宫格中的数字也彼此不相等，建立对应的不等式，由此创建11745个子句，其中包含：81个长度为9的子句，表示每个方格必须填入1-9九个数字中的一个；81\*36个长度为2的子句，表示每个方格只能填入一个数字；27\*324个长度为2的子句，表示每一行、每一列以及每一9\*9九宫格中的九个数字都彼此不相等。依次将’p’,’cnf’’,变元数（1000）和子句数写入创建的cnf文件中。

# 4系统实现与测试

## **4.1系统实现**

**编程环境**：

处理器Inter(R) Core(TM) i5-7300HQ CPU，64位操作系统 8G

WIN10下使用codeblocks16.01进行编译调试，语言选择为c++。

**各函数调用关系**：在SAT求解选项中主函数main通过switch调用各功能函数，各函数之间相互独立。在数独游戏选项中，主函数直接生成数独棋盘，并保存在一个名为sudo的二维数组中，系统会在用户选择数独游戏选项后显示一个数独棋盘，然后将二维数组作为实参送入transformcnf函数中，即转化为SAT问题，并保存在创建的cnf文件中，然后调用函数OpenFile，打开文件，建立公式内部结构，会显示一长串子句，即将数独问题转化为SAT问题后的子句集，打开文件后，调用时间函数，记录开始时间start,调用函数DPLL开始求解SAT，然后记录结束时间finish，用thetime表示时间差（finish-start）,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。最后用户可以选择查看答案，屏幕将会显示数独游戏的正确答案。

**程序清单：**见附录

## **4.2系统测试**

### 4.2.1各函数功能测试

**1.增加子句**（以基准算例为例，前提是已经打开了一个"D:\SAT测试备选算例\基准算例\功能测试\sat-20.cnf"的文件，其变元数为20，子句数为91，下同）

（1）选择菜单中的功能3，增加一个长度为6的子句：“1 -2 3 4 5 -6”在链表头部，结果如图4-1所示。

（2）选择菜单中的功能6，遍历公式内部结构，由图4-2可以明显看出，输入的新子句已经被插入在子句集首部，可验证函数addClause 功能正确。

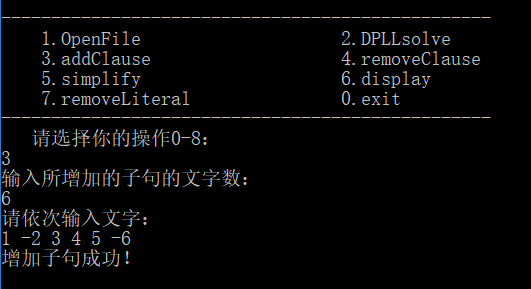


图4-1 增加子句

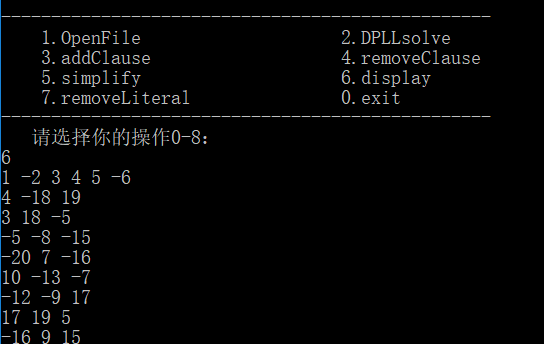


图4-2 增加子句后遍历整个子句集

**2.删除子句**

（1）选择菜单中的功能4，删除所有包含文字4的子句，界面显示如图4-3所示。

（2）选择菜单中的功能6，遍历整个子句集，可以直观地看出所有包含4的子句都被删除，最后显示“字句个数：82”。如图4-4所示。可验证函数removeClause的正确性。

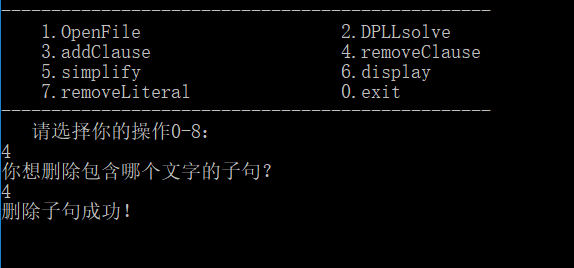


图4-3 删除子句

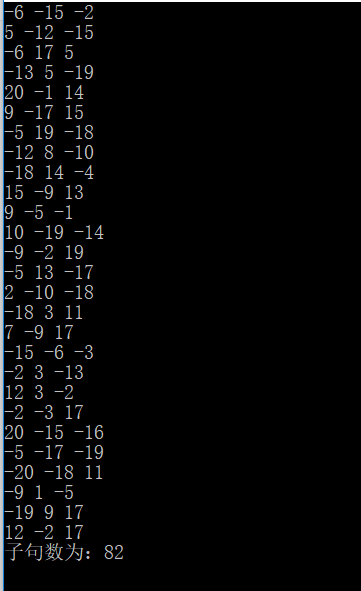
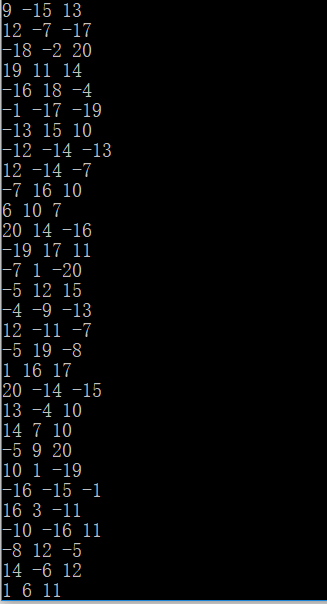
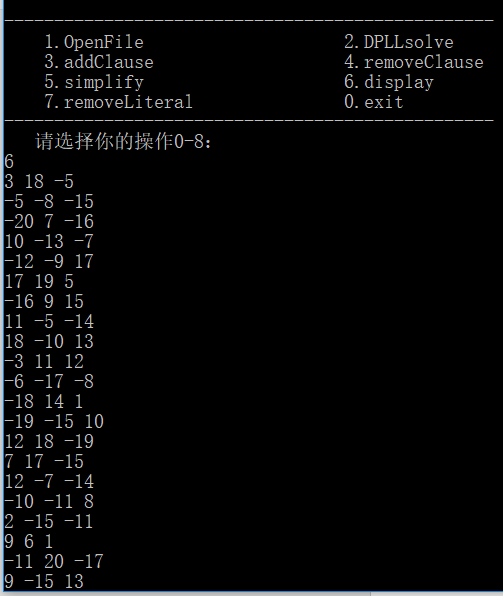


图4-4 删除成功后遍历整个子句集

**3.删除文字**

（1）选择菜单中功能7，删除指定的文字-5，按照屏幕指示进行操作后，显示输出如图4-5所示。

（2）选择菜单中功能6，对整个子句集进行遍历，可以直观地看出文字-5被删除，但最终子句数不变，仍然为91个，如图4-6所示。可验证函数removeLiteral的正确性。

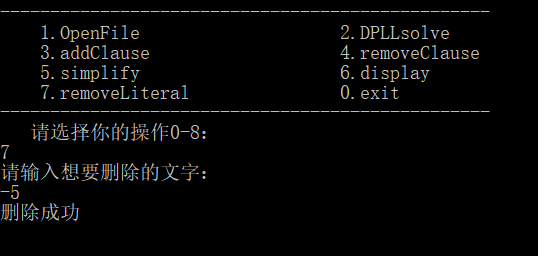


图4-5 删除指定文字

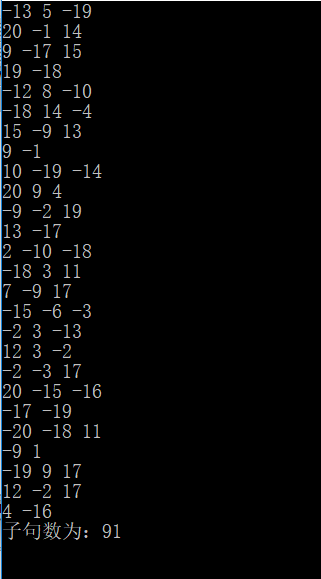
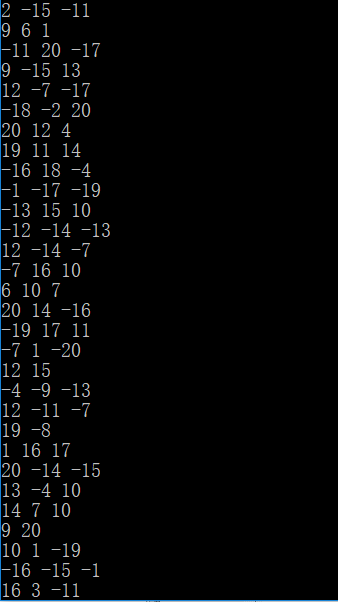
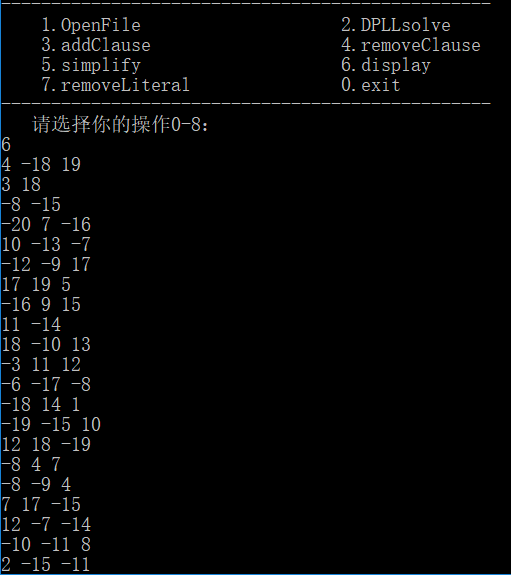


图4-6 删除指定文字后遍历整个子句集

### 4.2.2 SAT算例测试

用老师提供的SAT求解测试算例文件（包括基准算例、小型中型满足算例、不满足算例）对程序进行测试，其中算例文件“满足算例S-1”表示老师提供的SAT备选算例中满足算例的S文件夹中第一个cnf文件，算例文件“满足算例M-6”表示老师提供的SAT备选算例中满足算例的M文件夹中第六个cnf文件，算例文件“其他\_23”表示老师提供的SAT备选算例中“其他可供选择使用的算例”文件夹中第23个cnf文件，其他算例文件的表示以此类推。测试结果如表4-1所示

表4-1 SAT求解测试算例

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算例文件 | 变元数 | 子句数 | 可满足性 | 运行时间/ms(优化前） | 运行时间/ms(优化后) | 优化率 |
| Sat-20.cnf | 20 | 91 | 可满足 | 0 | 0 | 0 |
| Unsat-5cnf-30 | 30 | 420 | 不可满足 | 161 | 77 | 52.2% |
| 满足算例S-1 | 20 | 1532 | 可满足 | 123 | 89 | 27.2% |
| 满足算例S-2 | 20 | 91 | 可满足 | 1 | 0 | 100% |
| 满足算例S-3 | 50 | 80 | 可满足 | 1 | 0 | 100% |
| 满足算例S-4 | 100 | 340 | 可满足 | 21 | 30 | -42.8%% |
| 满足算例S-5 | 50 | 100 | 可满足 | 1 | 5 | -400% |
| 满足算例S-6 | 50 | 300 | 可满足 | 3 | 4 | -33.3% |
| 满足算例S-7 | 100 | 200 | 可满足 | 17 | 6 | 64.7% |
| 满足算例S-8 | 100 | 600 | 可满足 | 9 | 7 | 22.2% |
| 满足算例S-9 | 25 | 100 | 可满足 | 1 | 0 | 100% |
| 满足算例M-1 | 231 | 1166 | 可满足 | 15 | 23 | -53.3% |
| 满足算例M-6 | 200 | 320 | 可满足 | 32 | 19 | 9.3% |
| 满足算例M-7 | 200 | 1200 | 可满足 | 22 | 60 | -172.7% |
| 满足算例M-10 | 232 | 1901 | 可满足 | 25 | 23 | 8% |
| 满足算例M-11 | 303 | 2851 | 可满足 | 117 | 102 | 12.8% |
| 满足算例M-13 | 224 | 1762 | 可满足 | 9 | 8 | 11.1% |
| 满足算例M-14 | 297 | 2721 | 可满足 | 31 | 27 | 12.9% |
| 不满足算例\_4 | 80 | 370 | 不可满足 | 5957 | 6665 | -2.6% |
| 不满足算例\_6 | 10 | 100 | 不可满足 | 0 | 0 | 0% |
| 不满足算例\_14 | 50 | 100 | 不可满足 | 6 | 42 | -600% |
| 不满足算例\_15 | 100 | 200 | 不可满足 | 65 | 48 | 26.2% |
| 不满足算例\_16 | 99 | 264 | 不可满足 | 6869 | 5367 | 21.9% |
| ais10.cnf | 181 | 3151 | 可满足 | 1926 | 1273 | 33.9% |
| sud00009.cnf | 303 | 2851 | 可满足 | 115 | 107 | 7.0% |
| 其他\_23 | 20 | 91 | 可满足 | 1 | 1 | 0% |

部分算例测试截图如下：

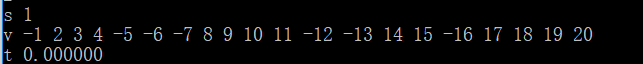
Sat-20.cnf的输出结果如图4-7所示（优化后）。

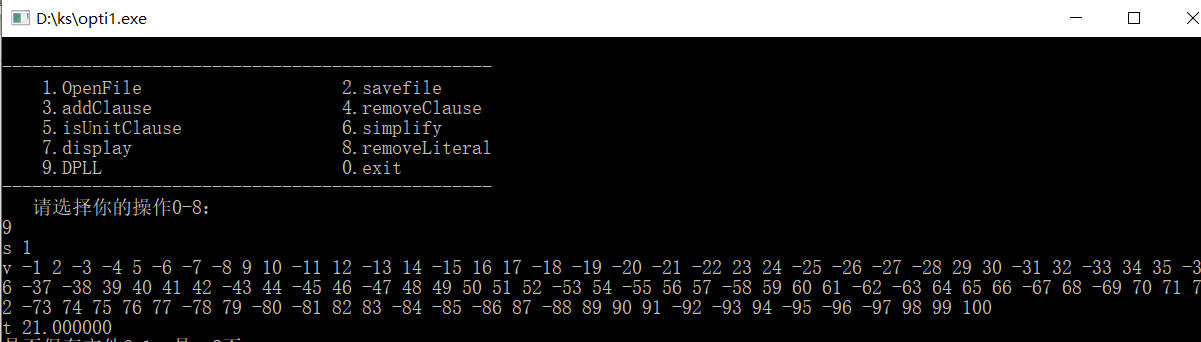
图4-7 基准算例的可满足算例

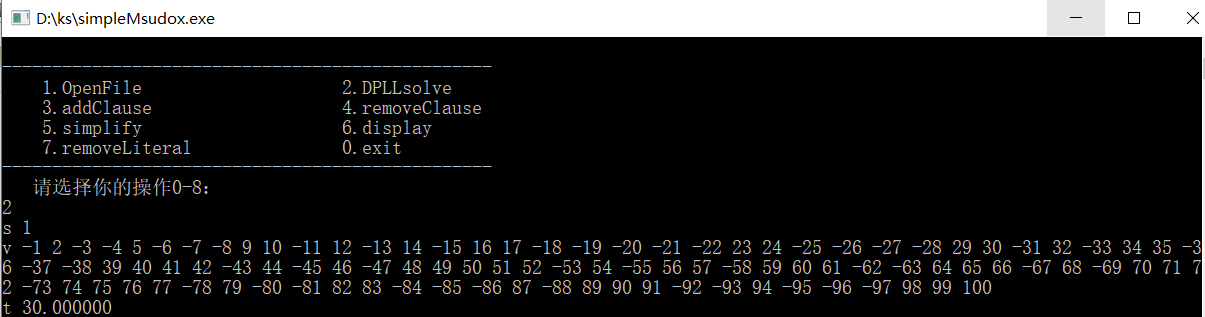
Unsat-5cnf-30.cnf的输出结果如图4-8所示（优化后）。



图4-8 基准算例的不满足算例

满足算例S-4（即“SAT测试备选算例\满足算例\S\problem3-100.cnf"）优化前和优化后输出结果如图4-9和图4-10所示。

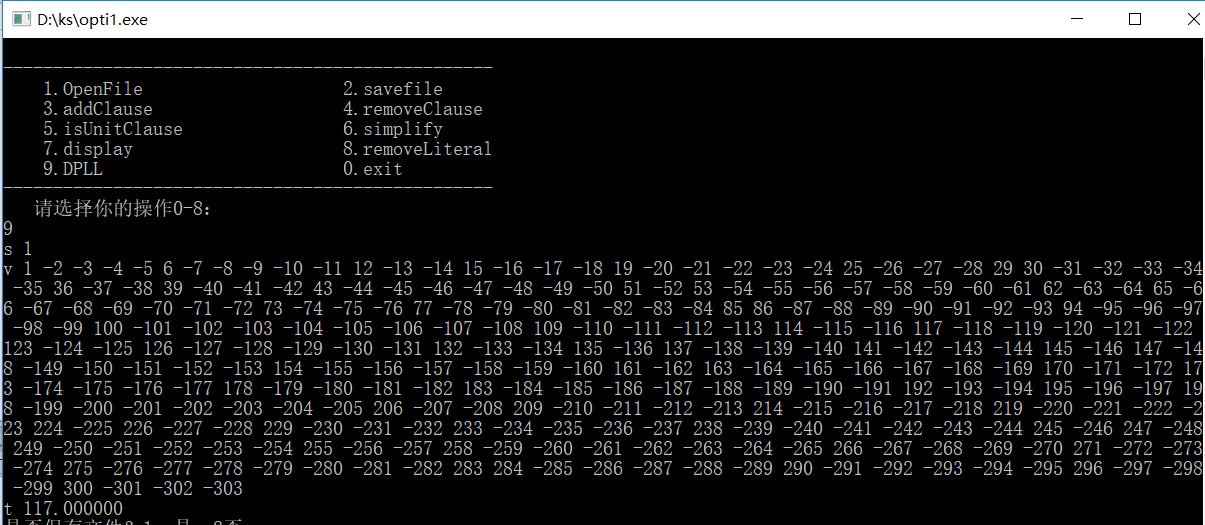
图4-9 满足算例S-4输出结果（优化前）



。

图4-10满足算例S-4的输出结果（优化后）

根据图4-9和图4-10，可看出对于该算例，优化后用时反而比优化前长。

满足算例M-10（即"D:\SAT测试备选算例\满足算例\M\sud00009.cnf"）优化前和优化后输出结果如图4-11和图4-12所示。

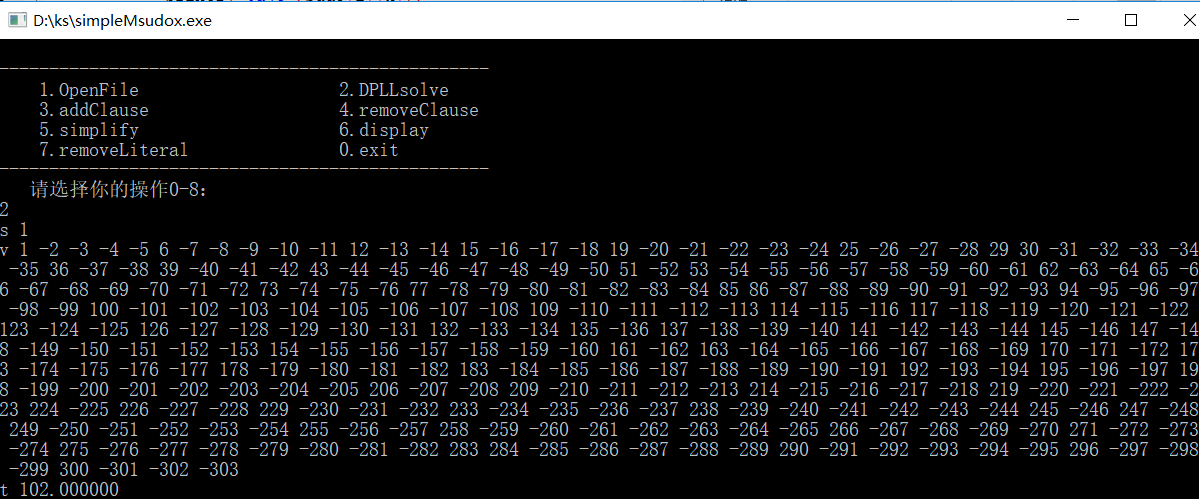
图4-11 满足算例M-10的输出结果（优化前）

图4-12 满足算例M-10的输出结果（优化后）

由图4-11和图4-12可以看出，对于该算例，优化后执行速率高于优化前，因此优化算法起到一定的提高执行速率的效果。

不满足算例\_4（即"D:\SAT测试备选算例\不满足算例\php-010-008.shuffled-as.sat05-1171.cnf"）优化前和优化后的输出结果如图4-13和图4-14所示。

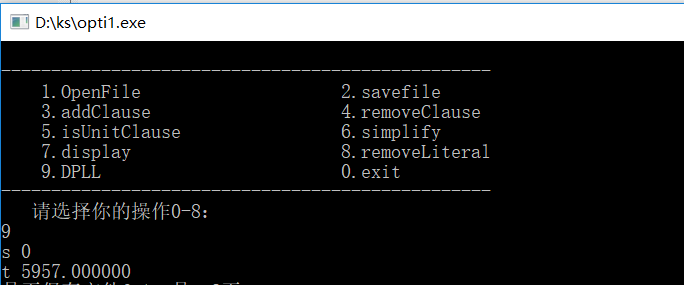
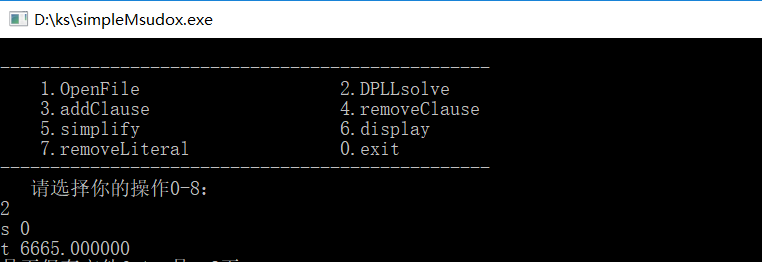
图4-13 不满足算例\_4输出结果（优化前）

图4-14 不满足算例\_4输出结果（优化后）

根据图4-13和图4-14，可以看出，对于该算例，优化后的执行时间长于优化前，但相差不是很大。

### 4.2.3 数独游戏测试

根据开始系统给出的选项，选择“0 数独游戏”，则按照预想显示出来一个初始数独格局，如图4-15所示。

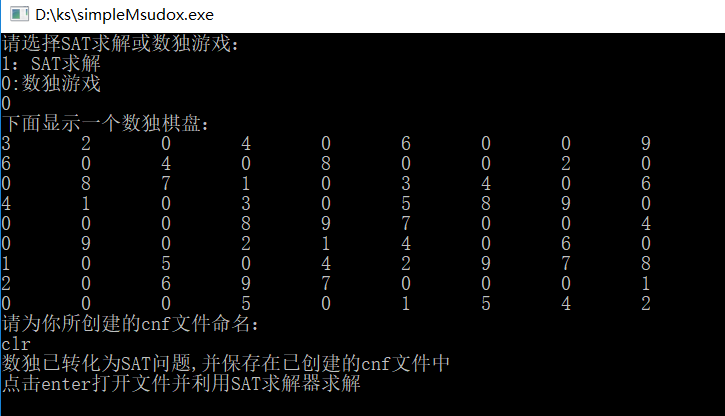


图4-15 选择数独游戏后显示的界面

此时，生成的数独格局已经转化为SAT问题，并保存在一个cnf文件中，用户可以自主为其命名。按照提示点击enter键可以打开文件，集成到上面的求解器进行问题求解，最后点击enter键，用户便可以查看数独答案，如图4-16所示。

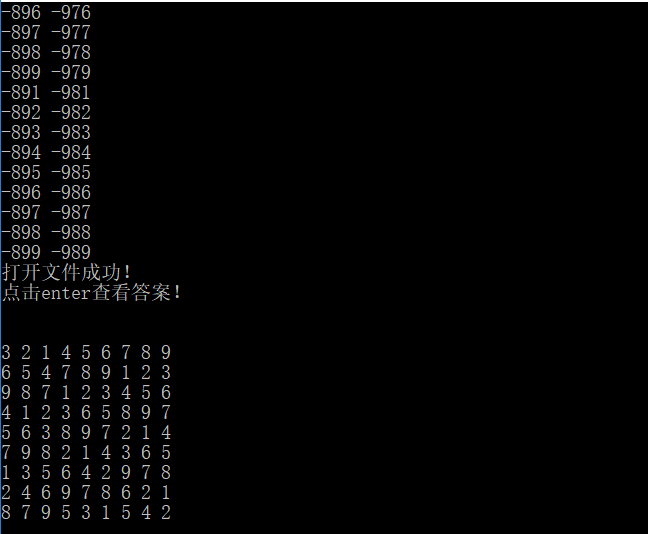


图4-16 打开文件，求解数独并显示答案

至此，系统功能便全部测试完毕。

## 4.3系统测试结论

根据图4-1至图4-6，菜单中所显示的函数功能完备，通过每次操作后遍历整个子句集便可人工验证其正确性。

根据表4-1以及图4-7至图4-14可知，该SAT求解器能实现所有小型算例的求解，部分中型和不满足算例的求解，且运行时间都较快，可以说是功能比较完备，但是可以看出，对于有些算例（如满足算例S-4、满足算例S-5、满足算例S-6），优化后的执行时间比优化前执行时间还要长，但对其他算例，优化过的程序执行效率比优化前快，说明程序功能实现的比较理想。另外，在求解不满足算例时，可以发现，用时普遍比满足算例长，说明我设计的DPLL递归算法在解决满足算例时执行效率更快。

根据图4-15和图4-16可知，该系统可提供一个数独棋盘，可供用户玩，具有一定的交互性，表明功能完备。

# 5总结与展望

5.1全文总结

### **5.1.1工作总结**

对自己的工作做个总结，主要工作如下：

（1）利用链表实现了对cnf文件的SAT问题数据结构的构建

（2）利用DPLL算法实现了SAT问题的求解

（3）将数独游戏转化为求解SAT问题

（4）游戏可玩，具有一定的/简单的交互性。

### 5.1.2 自我反思和总结

本次课程设计虽然花费很长的时间，但是收获很大，是我在应用数据结构上一个很大的突破。主要小结如下：

1. 首先在运用数据结构知识上，采取了合适的存储结构—链表来表示cnf文件的内部公式，这让我更深的理解了数据存储方式和调用方法。
2. 其次在课程设计的过程中，我也遇到了不少麻烦，例如在删除子句的函数中，一直没有发现程序的一个潜在的问题，就是跳出循环出口设置的与继续循环的条件一样，无法区分，这使我在修改bug的过程中花了很长的时间，在一步步的耐心的单步调试下，才顺利排除了错误，但这也使我进步了很多，明白了在编程时必须要考虑周全，不容有错。
3. 最后优化程序上，我也是想了许多种办法，例如，先对所有子句中出现次数最多的文字进行操作，或者是先对最短的子句进行操作，在不断的实验中，我发现对于不同的算例，不同的算法的执行时间也是不同的，这就与算例中的子句分布、长短有关了。这使我在程序优化上有了更全面的理解和认识。

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）学习中多思考、多实践，灵活运用数据结构中所学的知识，架构物理结构模型和逻辑结构，开拓思维。

（2）多研究一些算法，提高自己的编程效率。

（3）积极动脑，可在实践中多与老师、同学们交流探讨。

# 6体会

本次课程设计对我来说意义很大，花费两周时间，学到了很多东西，是我在应用数据结构上一个很大的突破。主要小结如下：

（1）首先在运用数据结构知识上，我采取了一种合适的存储结构—链表来表示cnf文件的内部公式，优化上还考虑到了使用十字链表的结构来实现，这让我更深的理解了数据存储方式和调用方法。我认为要想数量掌握数据结构这门课程的知识，就是要多实践，多思考。

（2）其次在课程设计的过程中，我也遇到了不少麻烦，例如在删除子句的函数中，一直没有发现程序的一个潜在的问题，就是跳出循环出口设置的与继续循环的条件一样，无法区分，这使我在修改bug的过程中花了很长的时间，在一步步的耐心的单步调试下，才顺利排除了错误，但这也使我进步了很多，明白了在编程时必须要考虑周全，才能为后续的操作打好基础。

（3）最后优化程序上，我也是想了许多种办法，例如，先对所有子句中出现次数最多的文字进行操作，或者是先对最短的子句进行操作，在不断的实验中，我发现对于不同的算例，不同的算法的执行时间也是不同的，这就与算例中的子句分布、长短有关了。这使我在程序优化上有了更全面的理解和认识。

（4）另外我认为老师在此次课程设计实践中给了我很大帮助，比如一开始我并不能很好的建立线性表与子句集的关系，在老师的指点下，我才顺利完成，在编程过程中，遇到了一些难以解决的问题，比如算法效率的比对、选取子句的方法等，通过与老师的交流，我才豁然开朗。所以我非常感谢老师给予我的帮助和指导。

# 参考文献

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

[2]Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

[4]Carsten Sinz.Visualizing SAT Instances and Runs of the DPLL Algorithm.J Autom Reasoning (2007) 39:219–243

[5] 360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic forProgramming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

[7]Ins Lynce and Jol Ouaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

# 附录

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define POSITIVE 1

#define NEGATIVE 0

typedef int status;

typedef int ElemType;

int bool\_num,clause\_num;

int copy\_num=0;

typedef struct literal{

ElemType data;

struct literal \*next;

}literal;

typedef struct clause{

struct clause \*next;

int num;//指示这条子句文字数，若为1则说明是单子句，头结点的num指示子句的个数

struct literal \*HeadNode;

}clause,\*Root;

typedef struct same\_literal{

ElemType data;

int num;

}same\_literal;

int result[500];

int top=0;

same\_literal \*sl;

clause \*\*copy\_formula;

int absoluteValue(ElemType e){

if(e>=0) return e;

else return (-e);

}

status OpenFile(Root &L){

L=(Root)malloc(sizeof(clause));

L->num=0;

L->next=NULL;

L->HeadNode=NULL;

FILE \*fp;

int t;//依次读入的每个文字

Root p=L;

Root tp;

literal \*tq,\*q;

char filename[100];

printf("请输入文件名：\n");

scanf("%s",filename);

fp=fopen(filename,"r+");

if(!fp){

printf("打开文件失败！\n");

getchar();

return ERROR;

}

char ts[300];

char c;

/\* char \*sp=s[0];\*/

int i,j;

i=j=0;

while((c=fgetc(fp))!='p')

fgets(ts,80,fp);//读取一行字符串到一个字符数组里

char us[10];

fscanf(fp,"%s",us);

fscanf(fp,"%d",&bool\_num);

fscanf(fp,"%d",&clause\_num);

/\* sl\_p=(same\_literal\*\*)malloc((bool\_num+1)\*(sizeof(same\_literal\*)));//一次性分配空间

sl\_n=(same\_literal\*\*)malloc((bool\_num+1)\*(sizeof(same\_literal\*)));

for(i=0;i<=bool\_num;i++){

sl\_p[i]=(same\_literal\*)malloc(sizeof(same\_literal));

sl\_n[i]=(same\_literal\*)malloc(sizeof(same\_literal));

}\*/

p->num=clause\_num;

sl=(same\_literal\*)malloc((bool\_num+1)\*(sizeof(same\_literal)));

i=0;

j=1;

p->next=(Root)malloc(sizeof(clause));

p=p->next;

q=(literal\*)malloc(sizeof(literal));

p->HeadNode=q;

printf("变元数：%d\n子句数：%d\n",bool\_num,clause\_num);

while(j<=clause\_num)

{ fscanf(fp,"%d",&t);

i=0;

while(t){

q->data=t;

i++;

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

if(t==0) break;

q->next=(literal\*)malloc(sizeof(literal));

q=q->next;

}

q->next=NULL;

p->num=i;//该子句中文字的个数

if(j==clause\_num) break;

p->next=(Root)malloc(sizeof(clause));

p=p->next;

q=(literal\*)malloc(sizeof(literal));

p->HeadNode=q;

j++;

}

p->next=NULL;

L->num=j;

p=L->next;

printf("\n下面逐行显示每个子句：\n");

while(p!=NULL){

q=p->HeadNode;

while(q!=NULL){

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

q=q->next;

}

printf("\n");

p=p->next;

}

for(i=0;i<=bool\_num;i++){

sl[i].num=0;

}

tp=L->next;//保存每个正文字的个数

while(tp!=NULL){

tq=tp->HeadNode;

while(tq!=NULL){

if(tq->data>0){

sl[tq->data].data=tq->data;

sl[tq->data].num++;}

tq=tq->next;

}

tp=tp->next;

}

/\* for(i=0;i<=bool\_num;i++){

sl\_p[i]->num\_sl=0;

sl\_n[i]->first=NULL;

sl\_n[i]->num\_sl=0;

sl\_n[i]->first=NULL;

}

tp=L->next;

i=j=0;

while(tp!=NULL){//创建十字链表

tq=tp->HeadNode;//遍历完所有文字后结束。

while(tq!=NULL){

if(tq->data>0){

tq->same=sl\_p[tq->data]->first;

sl\_p[tq->data]->first=tq;

sl\_p[tq->data]->num\_sl++;

}

if(tq->data<0){

tq->same=sl\_n[-(tq->data)]->first;

sl\_n[-(tq->data)]->first=tq;

sl\_n[-(tq->data)]->num\_sl++;

}

tq=tq->next;

}

tp=tp->next;

}

\*/

printf("打开文件成功！\n");

fclose(fp);

getchar();

return OK;

}

status addClause(Root &L){

int number;

int i;

int sc;

printf("输入所增加的子句的文字数：\n");

scanf("%d",&number);

Root new\_clause;

literal \*n;

new\_clause=(Root)malloc(sizeof(clause));

new\_clause->num=number;

new\_clause->HeadNode=(literal\*)malloc(sizeof(literal));

n=new\_clause->HeadNode;

new\_clause->next=L->next;

L->next=new\_clause;

printf("请依次输入文字：\n");

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

scanf("%d",&sc);

n->data=sc;

if(i==number-1) break;

n->next=(literal\*)malloc(sizeof(literal));

n=n->next;

}

n->next=NULL;

literal \*q=new\_clause->HeadNode;

/\* while(q!=NULL){

if(q->data>0){

q->same=sl\_p[q->data]->first;

sl\_p[q->data]->first=q;

sl\_p[q->data]->num\_sl++;

}

if(q->data<0){

q->same=sl\_n[-(q->data)]->first;

sl\_n[-(q->data)]->first=q;

sl\_n[-(q->data)]->num\_sl++;

}

q=q->next;

}\*/

L->num++;//字句个数加一

clause\_num++;

printf("增加子句成功！\n");

return OK;

}

status removeClause(Root &L,ElemType e){//删除包含文字e的子句

Root p=L->next;

Root p1=L;

Root p2;

if(p==NULL) return ERROR;

/\* if(e>0) sl\_p[e]->first=NULL;

if(e<0) sl\_n[-e]->first=NULL;\*/

literal \*q,\*rq;

while(p!=NULL){

q=p->HeadNode;

while(q!=NULL){

if(q->data==e){

q=p->HeadNode;

while(q!=NULL){

rq=q; q=q->next; free(rq);

}

p1->next=p->next;

p->HeadNode=NULL;

p2=p;

p=p->next;

L->num--;//子句数减一

clause\_num--;

free(p2);

if(p==NULL) return OK;

q=p->HeadNode;

break;

}

q=q->next;

}

if(q==NULL)

{p1=p;

p=p->next;}

}

return OK;

}

status removeLiteral(Root &L,ElemType e){//删除每个子句中的文字e

Root p=L->next;

if(p==NULL) return ERROR;

/\* if(e>0) sl\_p[e]->first=NULL;

if(e<0) sl\_n[-e]->first=NULL;\*/

literal \*q,\*q1,\*q2;

while(p!=NULL){

q=p->HeadNode;

q1=p->HeadNode;

while(q!=NULL){

if(q->data==e){

if(q==p->HeadNode){

p->HeadNode=q->next;q2=q;

q=q->next;free(q2);}

else {q1->next=q->next;q2=q;q=q->next;free(q2);}

p->num--;//文字数减一

}

else {q1=q;q=q->next;}

}

p=p->next;

}

return OK;

}

Root simplify(Root L,ElemType e){//假设e为真，化简L，返回指向新子句集的指针

/\*copy\_formula[copy\_num]=L;\*/

Root p=L;

literal \*q;

Root newL;

newL=(Root)malloc(sizeof(clause));//新链表的头结点

newL->num=L->num;

newL->HeadNode=NULL;

newL->next=NULL;

Root newp=newL;

literal \*newq;

p=p->next;

while(p!=NULL){

newp->next=(Root)malloc(sizeof(clause));

newp=newp->next;

newp->HeadNode=NULL;newp->next=NULL;

newp->num=p->num;

q=p->HeadNode;

newp->HeadNode=(literal\*)malloc(sizeof(literal));

newq=newp->HeadNode;

while(q!=NULL){

newq->data=q->data;

if(q->next==NULL) {newq->next=NULL;break;}

newq->next=(literal\*)malloc(sizeof(literal));

newq=newq->next;

q=q->next;

}

p=p->next;

}//复制出一个新的公式

newp->next=NULL;

removeClause(newL,e);

removeLiteral(newL,-e);

result[absoluteValue(e)]=e;

/\* copy\_num++;

copy\_formula[copy\_num]=newL;\*/

return newL;

}

status isUnitClause(Root &p){//判断是否为单子句

if(p==NULL) return ERROR;

if(p->num==1) return OK;

}

status EmptyClause(Root L){//是否含有空子句

Root p=L->next;

while(p!=NULL){

if(p->num==0) return ERROR;//含有空子句，不可满足

p=p->next;

}

return OK;

}

Root findunit(Root L){

Root p=L->next;

while(p!=NULL){

if(p->num==1)

return p;

p=p->next;

}

return NULL;//无单子句

}

ElemType optimize(Root L){

if(L->num==0) return ERROR;

if(L->num==1) return L->next->HeadNode->data;

Root p=L->next;

Root tp=p->next;

while(tp!=NULL){

if(tp->num<p->num){

p=tp;

}

tp=tp->next;

}

return p->HeadNode->data;

}

status DPLL(Root L){

Root tp=L->next;

if(tp==NULL) return TRUE;

if(!EmptyClause(L)) return FALSE;

while((tp=findunit(L))!=NULL){

ElemType w=tp->HeadNode->data;

removeClause(L,w);

removeLiteral(L,-w);

result[absoluteValue(w)]=w;

if(L->num==0) return TRUE;

else if(!EmptyClause(L)) {return FALSE;}

}

ElemType v=optimize(L);

if(DPLL(simplify(L,v))) return TRUE;//假设v为真

else return (DPLL(simplify(L,-v)));

}

status savefile(Root L,int t,int r){

int i;

FILE \*fp;

char filename2[100];

printf("为保存的文件命名(扩展名为.res)：\n");

scanf("%s",filename2);

fp=fopen(filename2,"w");

if(!fp) return ERROR;

if(r==0) {fprintf(fp,"%c ",'s');fprintf(fp,"%d\n",0);fprintf(fp,"t %d",t);printf("保存文件成功！\n");fclose(fp);return OK;}

fprintf(fp,"%c ",'s');fprintf(fp,"%d\n",1);

fprintf(fp,"%c ",'v');

for(i=1;i<=bool\_num;i++){

if(result[i]!=0)

fprintf(fp,"%d ",result[i]);

}

fprintf(fp,"\nt %d",t);

fclose(fp);

printf("保存文件成功！\n");

return OK;

}

status display(Root &L){

Root p=L->next;

literal \*q;

if(p==NULL) return ERROR;

while(p!=NULL){

q=p->HeadNode;

while(q!=NULL){

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

q=q->next;

}

printf("\n");

p=p->next;

}

printf("子句数为：%d\n",L->num);

return OK;

}

status transformcnf(int s[9][9]){

int i,j,d,k,l;

int zero=0;

int bonum=1000;

int clanum=0;

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

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

if(s[i][j]!=0) clanum++;

}

}

clanum=clanum+10287;

FILE \*fp;

char filename[100];

printf("请为你所创建的cnf文件命名：\n");

scanf("%s",filename);

if(!(fp=fopen(filename,"w"))) return ERROR;

fprintf(fp,"%c ",'p');

fprintf(fp,"%s ","cnf");

fprintf(fp,"%d %d\n",bonum,clanum);

//fprintf//需写入变元数和子句数

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

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

if(s[i][j]!=0)//已给的数字,存为单子句

{fprintf(fp,"%d %d\n",100\*(i+1)+10\*(j+1)+s[i][j],zero);}

}

}

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

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

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

fprintf(fp,"%d ",100\*(i+1)+10\*(j+1)+d);

}

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

}

}

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

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

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

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+1)+k));

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+1)+l));

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

}

}

}

}

//每行可满足：

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

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

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

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(k+1)+d));

fprintf(fp,"%d ",-(100\*(i+1)+10\*(l+1)+d));

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

}

}

}

}

//每列可满足

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

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

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

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

fprintf(fp,"%d ",-(100\*(k+1)+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(l+1)+10\*(j+1)+d));

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

}

}

}

}

//每个小方格需可满足

for(i=1;i<=7;i=i+3){

for(j=1;j<=7;j=j+3){

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

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+1)+d));

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

}

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

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+2)+d));

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

}

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

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

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+1)+d));

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

}

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

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

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+2)+d));

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

}//8个

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

fprintf(fp,"%d ",-(100\*i+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(i+1)+10\*j+d));

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

}

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

fprintf(fp,"%d ",-(100\*i+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+2)+d));

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

}

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

fprintf(fp,"%d ",-(100\*i+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*j+d));

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

}

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

fprintf(fp,"%d ",-(100\*i+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+2)+d));

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

}//7

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

fprintf(fp,"%d ",-(100\*i+10\*(j+2)+d));

fprintf(fp,"%d ",-(100\*(i+1)+10\*j+d));

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

}

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

fprintf(fp,"%d ",-(100\*i+10\*(j+2)+d));

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+1)+d));

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

}

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

fprintf(fp,"%d ",-(100\*i+10\*(j+2)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*j+d));

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

}

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

fprintf(fp,"%d ",-(100\*i+10\*(j+2)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+1)+d));

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

}

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*j+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+1)+d));

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

}

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*j+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+2)+d));

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

}//5

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*j+d));

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

}

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+1)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+2)+d));

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

}//4

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+2)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*j+d));

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

}

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

fprintf(fp,"%d ",-(100\*(i+1)+10\*(j+2)+d));

fprintf(fp,"%d ",-(100\*(i+2)+10\*(j+1)+d));

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

}

}}

fclose(fp);

return OK;

}

int main()

{

Root formula;

copy\_formula=(Root\*)malloc(100\*sizeof(clause\*));

int op\_1=1;

int op\_2=1;

int t;

while(op\_1==1)

{

printf("请选择SAT求解或数独游戏：\n1：SAT求解\n0:数独游戏\n");

scanf("%d",&op\_2);

if(op\_2==0) break;

getchar();

while(op\_2){

system("cls");

printf("\n");

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

printf(" 1.OpenFile 2.DPLLsolve\n");

printf(" 3.addClause 4.removeClause\n");

printf(" 5.simplify 6.display \n");

printf(" 7.removeLiteral 0.exit\n");

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

printf(" 请选择你的操作0-8：\n");

scanf("%d",&op\_2);

switch(op\_2){

case 1://创建子句

OpenFile(formula);

getchar();

break;

case 2:

int i,re;

clock\_t start,finish;

double thetime;

start=clock();

re=DPLL(formula);

finish=clock();

thetime=(double)(finish-start);

if(re==0)

printf("s 0\n");

else {

printf("s 1\n");

printf("v ");

for(i=1;i<=bool\_num;i++){

printf("%d ",result[i]);

}

printf("\n");

}

printf("t %f\n",thetime);

int o;

printf("是否保存文件？1：是 2否\n");

scanf("%d",&o);

if(o=1) savefile(formula,thetime,re);

getchar();

getchar();

break;

case 3:

addClause(formula);

getchar();

getchar();

break;

case 4:

printf("你想删除包含哪个文字的子句？\n");

scanf("%d",&t);

if(removeClause(formula,t))

printf("删除子句成功！\n");

else printf("该子句集为空，删除失败！\n");

getchar();

getchar();

break;

case 5:

int k;

Root newL;

printf("请输入想要化简的文字：\n");

scanf("%d",&k);

newL=simplify(formula,k);

printf("化简成功\n");

display(newL);

getchar();

getchar();

break;

case 6:

if(!display(formula))

printf("该子句集为空！\n");

getchar();

getchar();

break;

case 8:

int e;

printf("请输入想要删除的文字：\n");

scanf("%d",&e);

if(removeLiteral(formula,e)) printf("删除成功\n");

else printf("删除失败\n");

getchar();

getchar();

break;

case 0:

getchar();

break;

}//end of switch

}//end of while(op\_2),之后应该为数独游戏

}//end of while(op\_1)

int sudo[9][9];

sudo[0][0]=3;sudo[0][1]=2;sudo[0][2]=1;sudo[0][3]=4;sudo[0][4]=5;sudo[0][5]=6;sudo[0][6]=7;sudo[0][7]=8;sudo[0][8]=9;

sudo[1][0]=6;sudo[1][1]=5;sudo[1][2]=4;sudo[1][3]=7;sudo[1][4]=8;sudo[1][5]=9;sudo[1][6]=1;sudo[1][7]=2;sudo[1][8]=3;

sudo[2][0]=9;sudo[2][1]=8;sudo[2][2]=7;sudo[2][3]=1;sudo[2][4]=2;sudo[2][5]=3;sudo[2][6]=4;sudo[2][7]=5;sudo[2][8]=6;

sudo[3][0]=4;sudo[3][1]=1;sudo[3][2]=2;sudo[3][3]=3;sudo[3][4]=6;sudo[3][5]=5;sudo[3][6]=8;sudo[3][7]=9;sudo[3][8]=7;

sudo[4][0]=5;sudo[4][1]=6;sudo[4][2]=3;sudo[4][3]=8;sudo[4][4]=9;sudo[4][5]=7;sudo[4][6]=2;sudo[4][7]=1;sudo[4][8]=4;

sudo[5][0]=7;sudo[5][1]=9;sudo[5][2]=8;sudo[5][3]=2;sudo[5][4]=1;sudo[5][5]=4;sudo[5][6]=3;sudo[5][7]=6;sudo[5][8]=5;

sudo[6][0]=1;sudo[6][1]=3;sudo[6][2]=5;sudo[6][3]=6;sudo[6][4]=4;sudo[6][5]=2;sudo[6][6]=9;sudo[6][7]=7;sudo[6][8]=8;

sudo[7][0]=2;sudo[7][1]=4;sudo[7][2]=6;sudo[7][3]=9;sudo[7][4]=7;sudo[7][5]=8;sudo[7][6]=6;sudo[7][7]=2;sudo[7][8]=1;

sudo[8][0]=8;sudo[8][1]=7;sudo[8][2]=9;sudo[8][3]=5;sudo[8][4]=3;sudo[8][5]=1;sudo[8][6]=5;sudo[8][7]=4;sudo[8][8]=2;

int c,i,j,k;

int sudoresult[9][9];

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

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

sudoresult[i][j]=sudo[i][j];

}

}

srand(time(NULL));

int limit=5;

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

for(k=0;k<limit;k++)

{

c=rand()%9;

sudo[i][c]=0;

}

}

printf("下面显示一个数独棋盘：\n");

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

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

printf("%d\t",sudo[i][k]);

}

printf("\n");

}

if(!transformcnf(sudo))

{printf("保存文件失败！\n");return 0;}

printf("数独已转化为SAT问题,并保存在已创建的cnf文件中\n点击enter打开文件并利用SAT求解器求解\n");

getchar();

getchar();

Root formula2;

OpenFile(formula2);//打开这个cnf文件

int temp;

int r2;

long thetime2;

clock\_t start2,finish2;

double thetime;

start2=clock();

r2=DPLL(formula2);

finish2=clock();

thetime2=(double)(finish2-start2);

/\*printf("是否要保存cnf文件的输出结果？\n1：是 2：否");

scanf("%d",&temp);

if(temp==1) {savefile(formula2,thetime2,r2);}\*/

printf("点击enter查看答案！\n");

getchar();

getchar();

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

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

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

}

printf("\n");

}

return 0;

}//end of main