

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 程序设计综合课程设计**

**专业班级： 计卓1701**

**学 号： U201714481**

**姓 名： 张宇平**

**指导教师： 许贵平**

**报告日期： 2019.3.16**

**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**

[任 务 书 I](#_Toc4275701)

[1 引言 1](#_Toc4275702)

[1.1课题背景与意义 1](#_Toc4275703)

[1.1.1课题问题概述 1](#_Toc4275704)

[1.1.2课题研究背景及出发点 3](#_Toc4275705)

[1.1.3 SAT问题研究意义 3](#_Toc4275706)

[1.2国内外研究现状 4](#_Toc4275707)

[1.3课程设计的主要研究工作 5](#_Toc4275708)

[2系统需求分析与总体设计 6](#_Toc4275709)

[2.1系统需求分析 6](#_Toc4275710)

[2.2系统总体设计 6](#_Toc4275711)

[3系统详细设计 9](#_Toc4275712)

[3.1有关数据结构的定义 9](#_Toc4275713)

[3.2主要算法设计 10](#_Toc4275714)

[4系统实现与测试 19](#_Toc4275715)

[4.1系统实现 19](#_Toc4275716)

[4.1.1系统概况： 19](#_Toc4275717)

[4.1.2系统使用的函数支持： 19](#_Toc4275718)

[4.1.3程序所包含的数据类型（C语言描述）： 19](#_Toc4275719)

[4.1.4系统中包含的函数： 20](#_Toc4275720)

[4.1.5系统中的函数间的组织和调用关系 21](#_Toc4275721)

[4.2系统测试 23](#_Toc4275722)

[5总结与展望 36](#_Toc4275723)

[5.1全文总结 36](#_Toc4275724)

[5.2工作展望 36](#_Toc4275725)

[6.体会 37](#_Toc4275726)

[参考文献 38](#_Toc4275727)

[附录 39](#_Toc4275728)

# 1 引言

## 1.1课题背景与意义

命题逻辑公式的可满足性（即SAT）是数理逻辑，计算机科学，集成电路设计与验证和人工智能等领域中的核心问题，并且是第一个被证明出来的NP问题。SAT问题在计算复杂性问题具有非常重要的意义。若设计出高效算法，对人工智能等计算科学领域的发展具有重要意义，因此我们努力研究新的算法，谋求更高效的算法！

## 1.1.1课题问题概述

可满足性问题，（Satisfiability Problem）,是对一个以合取范式（常简称cnf公式）的形式给出命题逻辑公式进行判断，以找出是否存在一个真值指派，该命题公式的逻辑值为真。

对于任一布尔变元x，x与其非“¬x”称为文字(literal)。对于多个布尔变元，若干个文字的或运算l1∨l2∨…∨lk称为子句(clause)。只含一个文字的子句称为单子句。不含任何文字的子句称为空子句，常用符号□表示。子句所含文字越多，越易满足，空子句不可满足。

一个CNF公式也可以表示成子句集合的形式：S={c1,c2,...,cm}.例如，由三个布尔变元a,b,c所形成的一个CNF公式（¬a∨b）∧（¬b∨c）,可用集合表示为{¬a∨b,¬b∨c}，该公式是满足的，a=0, b=0,c=1是其一组解。

一个CNF SAT公式或算例的具体信息通常存储在一个.cnf文件中，下面是算例problem1.cnf文件前若干行的截图（如图1-1所示）。



图1-1 cnf文件格式

SAT问题一般可描述为：给定布尔变元集合{x1, x2, ..., xn}以及相应的子句集合{c1, c2, ..., cm}，对于合取范式（CNF范式）：F = c1∧c2∧...∧cm，判定是否存在对每个布尔变元的一组真值赋值使F为真，当为真时（问题是可满足的，SAT），输出对应的变元赋值（一组解）结果。SAT问题是许多领域的中心问题，是典型的NP问题。

DPLL算法是一种经典的基于树的回溯算法，主要使用两种基本处理策略：

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

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

根据上述规则可不断对公式化简，并最终达到终止状态，其执行过程可表示为一棵二叉搜索树（如图1-2所示）。



图2.2 DPLL算法搜索树

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

}

## 1.1.2课题研究背景及出发点

从1960年至今，SAT问题一直备受人们关注，因为SAT问题在许多领域有重大意义。最开始人们想出来的解SAT算法时间复杂度或空间复杂度较高，无法解一些数据量较大的问题。但是，人们并没有放弃对求解SAT问题更高效的算法的研究，以及求解SAT问题的重大意义，在此背景下，人们已经想出了许多高效的完备性算法，能够在有限时间内求解大数据。截止目前，可满足性问题的研究仍是以完备算法为主要研究方向，许多高效算法是在一种称为DPLL算法的基础上衍生出来的，现已被称为SAT问题的标准求解方案。我对该课题的研究也是在DPLL算法的基础上，对SAT问题求解。

## 1.1.3 SAT问题研究意义

可满足性问题是逻辑学的一个基本问题，也是当今计算机科学和人工智能研究的核心问题。是许多领域的重要研究对象，但是一般求解算法的时间和空间开销却异常惊人。对于一个含有n个命题逻辑变量的合取范式来说，如果使用穷举法来罗列所有真值指派进行求解，虽然在理论上可行的，但是从算法的时间效率和空间效率来看不可行的，时间复杂度达到了指数级为O(2^n),计算机如果采用这种方式进行求解将负担不起如此大的开销，并且所花费的时间是人们不能容忍的。在工程技术，军事，工商管理，交通运输与自然科学研究中的许多重要问题，如程控电话的自动交换，大型数据库的维护， 集成电路的自动布线，软件自动开发等，都可转化成SAT问题。因此致力于寻找求解SAT问题的快速而高效的算法，不仅在理论研究上，而且在许多应用领域都具有极其重要的意义。

## 1.2国内外研究现状

因SAT问题的在计算理论研究领域和人工智能领域的重大作用，许多学者都在SAT求解领域做了大量的研究，可满足性问题进而也成为国内外研究的热点问题，并在算法研究和技术上取得了较大突破，这也推动了许多领域的发展。SAT问题现在国内外研究出来的求解算法大致分为完备性算法和局部搜素算法，局部搜素算法时间相对于完备算法时间效率较高，但是有点不可靠，因为可能解不出来解。DPLL是最经典的完备算法，其他的完备算法大多都是在DPLL算法的基础上衍生出来的。由于SAT问题本身的特性使得其在最坏的情况下的时间复杂度是指数级别的，使得很多研究者望而却步，但是我们研究者并没有放弃对SAT问题的研究，到目前为止，SAT问题的解法发展较快。很多国家都有举办了一些SAT竞赛和研讨会，加速了SAT问题的发展，目前仍然在研究的是找到一种更快的DPLL算法。过去十几年中SAT算法的发展所取得的惊人变化速度（如图1-3所示）。

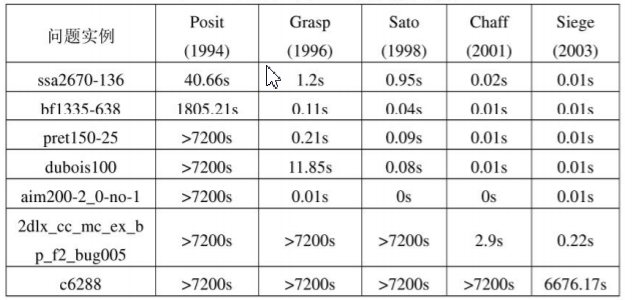


图1-3 过去十几年中SAT求解速度的进展

## 1.3课程设计的主要研究工作

本文主要学习了命题逻辑可满足性问题的相关理论知识，对DPLL问题做了较为深入的学习与研究，对基于DPLL的求解器所采用的关键技术及其算法进行了研究。

对于课程设计的相关要求，本文学习了一些数据结构，以便求得更适用于DPLL算法的高效的数据结构，对线性表与邻接表的删除插入等操作进行了学习，对文件的读写，建立数据结构物理存储结构等操作做了演示，应用于DPLL算法。DPLL算法中的决策变量阶段较为重要，对算法整体的时间复杂度影响较大，因此，本文对DPLL算法决策变量阶段做了讨论。对SAT问题的研究背景，意义及现状进行了简要总结，学习了命题逻辑可满足性问题的基本理论知识。阐述了对DPLL算法研究价值与意义。最终优化改进了DPLL算法。

综上所述，本文研究了以下内容：

1. 对SAT问题的研究背景，意义及现状进行了简要阐述；
2. 学习了命题逻辑可满足性问题的基本理论知识；
3. 学习了一些经典的数据结构和文件读写操作与物理存储的建构
4. 对该课题中所需的数据结构与一些需要的算法进行了设计
5. 对DPLL算法变量决策进行了较深入的学习与分析，思考更高效的变量决策
6. 对DPLL算法进行了优化
7. 对整个程序进行模块化处理，对系统进行设计，将数独与SAT求解器更好的结合一起；
8. 对数独游戏进行了思考，设计算法自动生成数独格局并转化为cnf文件，设计人机交互。

# 2系统需求分析与总体设计

## 2.1系统需求分析

本系统主要功能是基于DPLL过程实现一个高效的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，能够解数独游戏，具有人机交互。具体有以下功能：

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。
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]。
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[1-3]等某一方面进行优化设计与实现，提供明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。
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。数独格局的生成由算法自动生成。

本系统将上述功能实现，先设计问题中的变元，文字，子句，公式等有效的李存储结构，在读取cnf文件，建构物理存储结构，验证公式保证读取的有效性，接下来进行DPLL过程，输出SAT问题的解，也可以进行数独游戏。

## 2.2系统总体设计

根据课程设计问题的相关要求，本文设计和规划了一个系统，程序模块化组织，主要模块包括如下：

主控、交互与显示模块（display）

CNF解析模块（cnfparser）

核心DPLL模块( solver)

数独模块,包括数独生成、归约、求解(Sudoku)

具体来说，系统主要具有以下功能模块

1. 主函数模块：初始化界面，用户选择功能，参数的输入以及结果的输出；
2. 读取解析：读取cnf文件解析并建立cnf公式的物理存储结构；
3. 验证：输出cnf公式的内部表示以验证cnf公式解析的正确性；
4. DPLL：根据DPLL算法求解SAT问题的解，保存解，同时输出DPLL的运行时间，求出优化率；
5. 保存文件：保存SAT解到一个与cnf公式同名的res文件；
6. 数独格局：依据算法以及挖洞法自动生成初始化数独格局；
7. 输出数独公式文件：根据数独格局转换成相应的cnf公式，并保存为 sudoku.cnf;
8. 解数独：将数独转化为SAT求解后，在将解转换为对应的数独格局；
9. 人机交互模块：SAT过程中的求解数独过程中的人机交互，游戏具有可玩性；

依据上面阐述，下图提供了一个程序处理流程图（如图2-1所示），红色部分为程序中实现基于DPLL的SAT求解相关功能模块，蓝色部分是数独生成，转换，求解等处理模块。

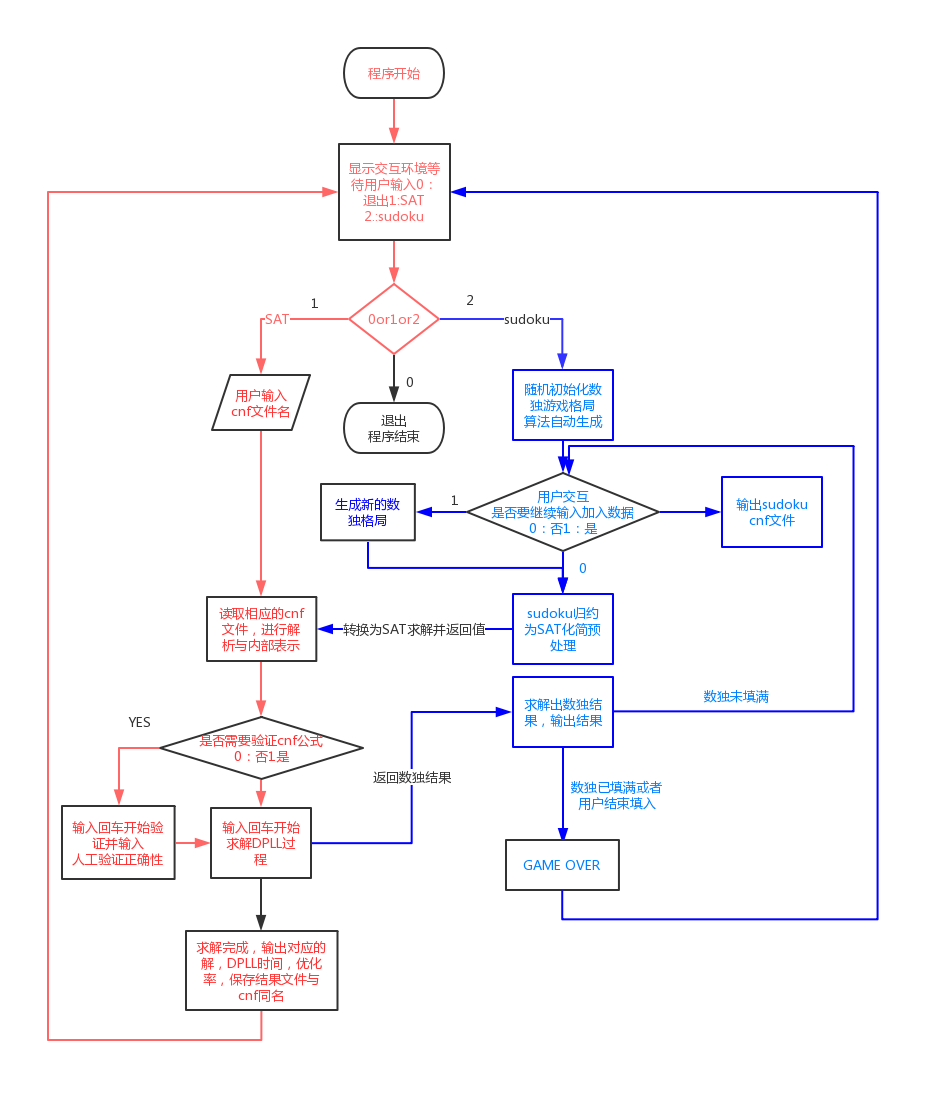


图2-1 程序模块结构图

# 3系统详细设计

## 3.1有关数据结构的定义

1. 系统中需要处理的和考虑的数据有
2. cnf公式中的文字数与子句数；
3. cnf公式中的每一个子句，子句类型包括子句句头，子句中包含的现有的文字数
4. 子句中的文字类型，主要是变元；
5. 以及求解过程中每一个变元的解（0或1）需要记录下来；
6. 为了直观的表示每一个文字以及子句在求解过程中是否被删除，每一个子句类型，增加了判定子句删除的标记clamark,在每一个文字类型增加了判定文字是否删除的标记litmark;
7. 为将整个数据连接起来，设置了一些指针；

（如表3-1所示）

表3-1 系统中要处理的数据

|  |  |  |  |
| --- | --- | --- | --- |
| 数据元素 | 数据项 | | |
| 根类型 | 子句个数 | 文字个数 | 指向第一个子句的指针 |
| 子句类型 | 子句包含的文字数 | 子句是否被删除的标志 | 指向下一个子句的指针以及  指向该子句第一个文字的指针 |
| 文字类型 | 文字中的变元以及变元的解 | 文字是否被删除的标志 | 指向下一个文字的指针 |

1. 系统在读取解析cnf公式后，需要将数据在程序中进行内部表示，因此就需要设计数据结构，本系统采用邻接表的形式，将数据进行关联；即根结点具有指向第一个子句的指针，每一个子句需要与下一个子句连接起来，所以每个子句需要有指向下一个子句的指针；而子句与第一个变元连接，所以每个子句类型又具有指向文字的指针；文字后需要连接下一个文字，所以文字类型中需要指向下一个文字的指针；
2. 为了优化原始DPLL，提高时间效率，本系统另外设计了一个邻接表，成为变元表：该表表头为公式中出现的不同变元，即将出现的每个相同的变元采用线性表连接起来，数据项包括，指向该变元所在子句的子句头指针，指向该变元出现的文字的指针；将所有线性表表头连接起来；

物理存储结构如下（如图3-2所示所示）：



图3-2 公式数据存储的物理结构

## 3.2主要算法设计

1. status Loadcnf(char \*filename, Root\* root, int flag)读取解析文件;

算法输入：cnf文件名

算法输出：如果成功读取解析文件并实现物理结构的构造，返回OK;

否则返回ERROR;

算法描述: （1）先打开相应的文件，逐行读入字符，直到读到p cnf结束该步骤，失败返回ERROR;

（2）为根结点开辟空间，读取文字数与字节数，并赋值给(\*root)->varnum 和root-> calnum；

（3）读取数据，为子句头开辟结点，为文字开辟结点，将读取到的变元赋给liter,不断后插，将真个子句连起来，到0结束；

（4）循环处理第三步，直到读到的子句数满足为止；

（5）结束算法，返回OK；

时间复杂度T(n)=O(m\*n),m为子句数量，n为变元数

空间复杂度S(n)=O(1);

具体的算法流程图（如图3-3所示）

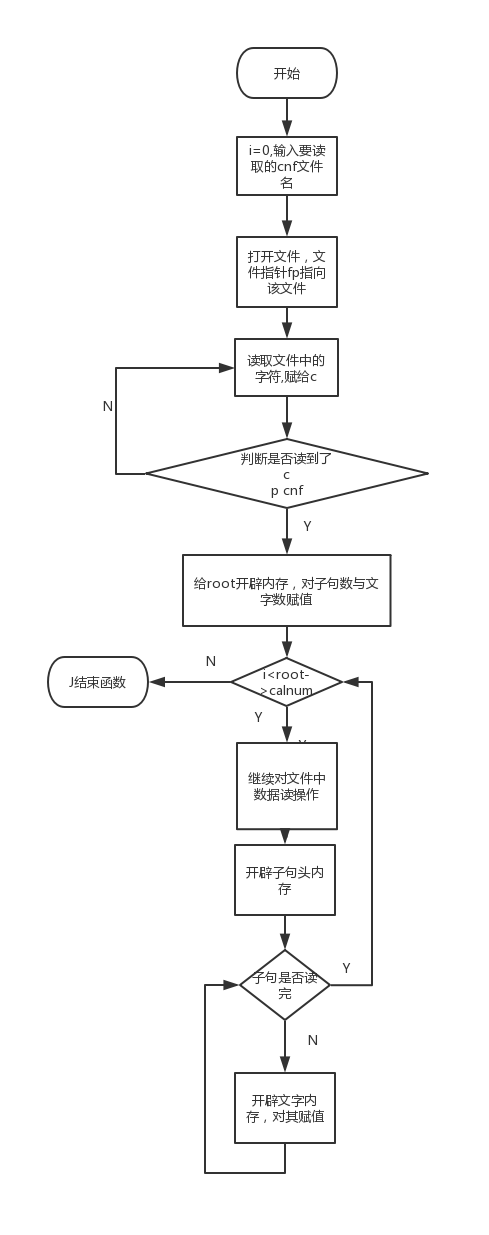


图3-3 Loadcnf函数算法流程图

1. status Testcnf(Root root)；验证读取的公式解析是否正确

算法输入：根结点指针root;

算法输出：成功验证返回OK;

否则返回ERROR;

算法描述：即遍历root指向的邻接表，输出内部表示；

时间复杂度T(n)=O(m\*n),m为子句数量，n为变元数

空间复杂度S(n)=O(1);

1. status savefile(char \*filename, int \*valuation, Root root, int flag, double t)保存结果文件；

算法输入：求解结果，cnf公式文件名，根结点，是否可解的标志flag，运行时间t；

算法输出：成功保存与cnf公式同名的res文件则返回OK;

否则返回ERROR；

算法描述： （1）根据cnf文件名，找到文件名后的.cnf,把文件名改成.res;

（2）文件指针创建该文件，若失败返回ERROR，将s 0/1 v存入；

（3）遍历解向量，若变元L解为1，则存入L，否则存入-L；

（4）最后存入时间t，关闭文件指针；

（5）函数结束，返回OK；

时间复杂度为O(n)，n为变元数量

空间复杂度S（n）=O(1);

1. void liternum(Root root, Word \*lb),构建变元表；

算法输入：根结点指针root,变元表lb;

算法输出：无；

算法描述： （1）遍历根结点，访问子句，将每个子句所含的变元数赋给数组litnum，litnum[i]表示第i-1个子句所含的变元数；

（2）为变元表开辟空间，读取子句变元，若变元为+L,则在变元表数组lb第L+1位置为其创建线性表，若变元为-L,则在变元表lb第1+L+root->varnum位置为其创建线性表；直到读完该子句所有变元；

（3）重复第一和第二步，直到读完所有子，文字表构建完成；

（4）函数结束；

时间复杂度T(n)=O(n)n为子句数量

空间复杂度S(n)=O(1);

1. int variable\_decide(Root root, int \*valuation)决策变量；

算法输入：根结点root,解向量valuation;

算法输出：选出的变量L,或-1，-2；

算法描述：（1）L赋值为-2，遍历子句头，直到找到p->clamark（表示该子句未被删除），定位到该子句；

（2）继续从步骤2定位到的子句遍历子句头，如果存在未被删除的但是为空的子句，返回-2；

（3）遍历过程中找到未被删除的单子句，则依据文字值给其赋值为1，valuation[0]--,返回该变元；

（4）遍历过程中如果没有未被删除的子句，即子句集为空，则返回-1；

（5）如果没有上述条件的子句，则找到最短子句的第一个变元，赋值为1，valuation[0]--，返回该变元；

（6）结束算法；

时间复杂度为O（n），空间复杂度为O（1）；

算法流程图如下图（如图3-4）

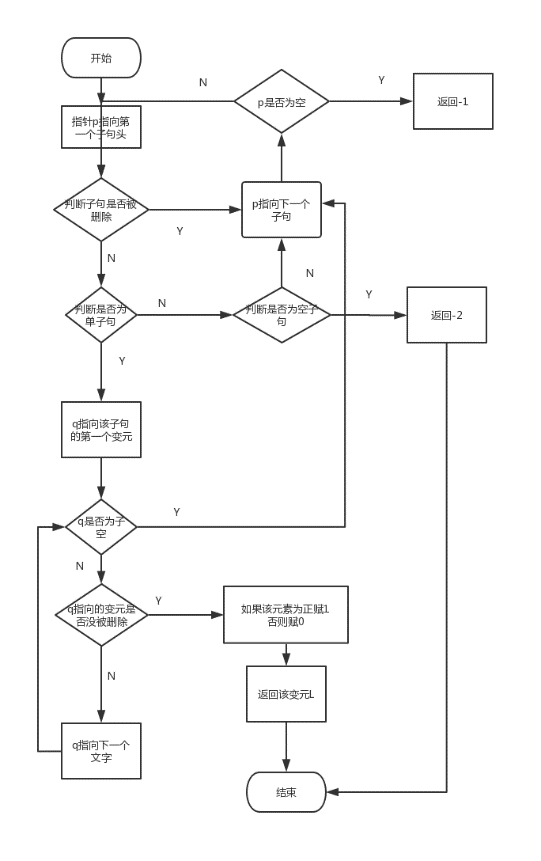


图3-4 决策变量部分算法流程图

1. void cnfrestore(Root \*root, int L, Word \*lb)；恢复子句

算法输入：根结点指针\*root，变元L,变元表指针lb;

算法输出：空；

算法描述： （1）指针p,q分别指向变元表第L个和第L + (\*root)->varnum元素的next,即p,q指向了子句中第一个出现L和-L的变元；

（2）p向后遍历，r指向L出现的子句的子句头，t指向L那个变元，

如果r指向的子句因为L被删除，则将该标志改为0；如果t指向的变元mark=L，则将mark改为0，同时将子句头中的litnum+1，直到p为空；

（3）对q同样执行遍历操作，与步骤（2）操作相同；

（4）结束算法；

时间复杂度为O（n）n为子句数量

空间复杂度为O(1)

1. void cnfdeduce(Root \*root, int L, int flag, Word \*lb)；子句化简

算法输入：根结点指针\*root,变元L，flag表示L的值0或1，变元表指针\*lb;

算法输出：空；

算法描述： （1）指针p指向L为真的文字表，q指向L为假的文字表；

（2）遍历p，r指向p->clahead;,如果该L出现的子句没有被删除，则执行r->clamark = L;p后移，重复上述操作，直至p为空；

（3）遍历q，r指向q->clahead，t指向该-L出现的那个文字，如果子句未被删除，则将该文字的mark改为L，同时将子句头的litnum-1；q后移，重复上述操作，直到q为空；

（4）结束算法；

时间复杂度为O（n）n为子句数量

空间复杂度为O（1）

1. status DPLL(Root root, int \*valuation)；解sat的关键算法DPLL；

算法输入：根结点root，解向量valuation；

算法输出：若该公式可解输出OK；

若不可解输出ERROR；

算法描述： （1）先对栈和文字表进行初始化操作；

（2）调用函数litnum构建变元表；决策第一个变量并入栈；

（3）判断公式是否可解，若可解执行步骤（4），若不可解执行步骤（6）；

（4）化简公式，判断公式是否为空，若为空则返回OK；否则执行步骤（5）；

（5）判断公式是否可解，若可解则将决策变量并入栈，若不可解则执行步骤（6）；

（6）从栈顶开始向下，找到首个未翻转变量，根据已翻转的变量恢复子句集0同时出栈，接着执行步骤（3），若栈已为空则返回ERROR；

时间复杂度为O（2^n）n为变元数量

空间复杂度为O(1)

算法流程图如下（如图3-5）

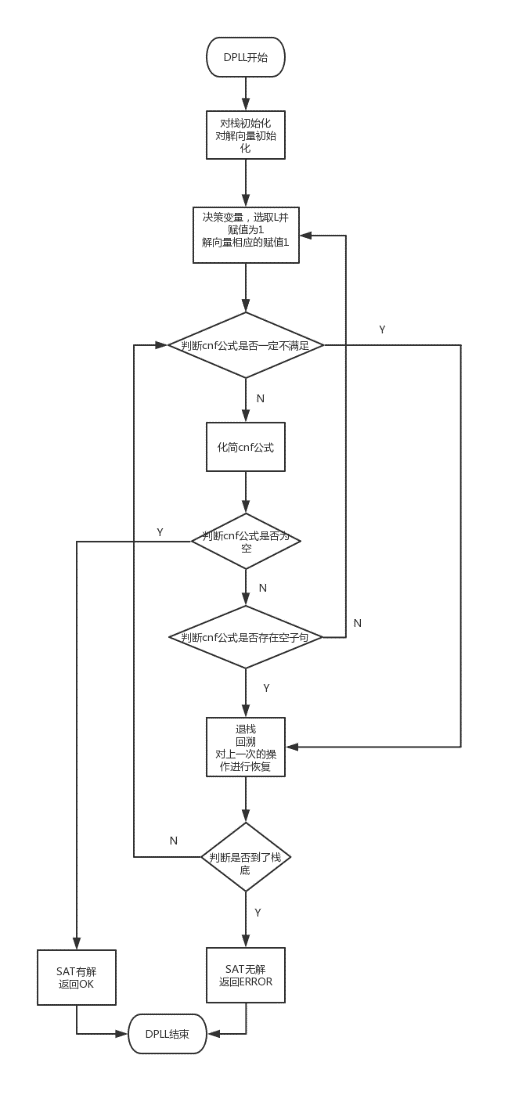


图3-5 DPLL算法流程图

1. status evaluateClause(Root root, int L, int \*value)；判断是否一定存在冲突，单子句冲突或含有空子句；

算法输入：根结点root，解向量valuation,上一步决策的变元；

算法输出：可满足输出OK；不满足输出ERROR；

算法描述： （1）从第一个子句开始向后遍历，若该子句为一个没有被删除的而且变元数为1 的子句，如果该子句的变元t为正但值为0，或者t为负但值为1，返回ERROR；

（2）如果找到一个子句未被删除但子句变元数为0，则返回ERROR；

（3）一直向后下一个子句遍历，重复步骤（2）（3）；

（4）若遍历完所有子句，则返回OK；

时间复杂度为O(n)

空间复杂度为O(1)

算法流程图如下（如图3-6 所示）

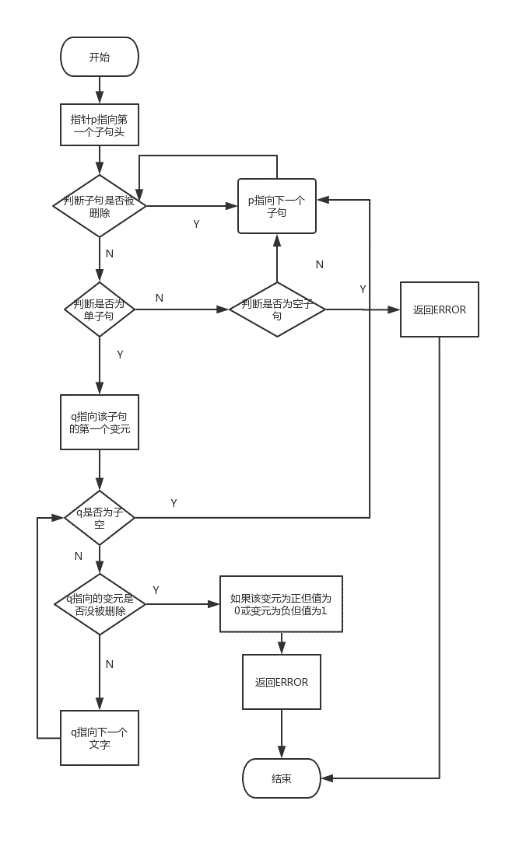


图3-6 判断是否存在冲突函数算法流程图

1. status createsudoku()；数独有关的操作；

算法输入：无

算法输出：如果成功完成 操作返回OK；否则返回ERROR；

算法描述： （1）首先根据随机初始化中间那个3\*3方格，根据矩阵行列变换生成其余3\*3格局，完成数独格局的初始化操作；

（2）随机挖洞生成数独初始格局，输出当前格局；

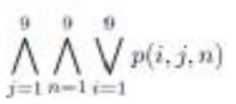
（3）用户判断是否需要继续加入数据，如果需要则加入数据；不需要则进行第（6）步；

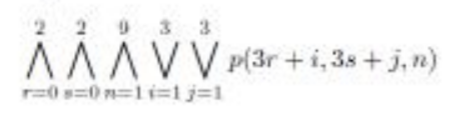
（4）依据算法将数独格局转化为cnf公式保存起来；

（5）调用DPLL解出公式的解，若可解，则继续进行第三步，不可解，则将第三步加入的数据恢复，进行第三步；

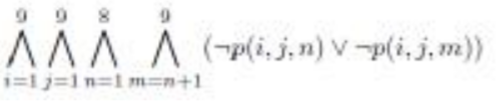
（6）先依据算法将数独格局转化为cnf公式，然后DPLL解出解，反向将解转化为数独格局，输出格局，结束算法；

其中数独格局转化为cnf公式的算法如下：

1. 例如变元161表示第一行第六列含有数字1，-455表示第四行第五列不含有数字5
2. 现在将数独格局中第i行第j列的值为k转化为cnf公式中的81(i-1)+9(j-1)+k号变元处理；
3. 第i行第j列值为k,则有单子句81(i-1)+9(j-1)+k
4. Every row contains every number，即；
5. Every colnum contains every number，即；
6. Every 3\*3block contains every number,即

；

1. NO cell contains more than one number:即



时间复杂度为O（2^n）n为变元数量

空间复杂度为O（1）

算法流程图如下（如图3-7所示）

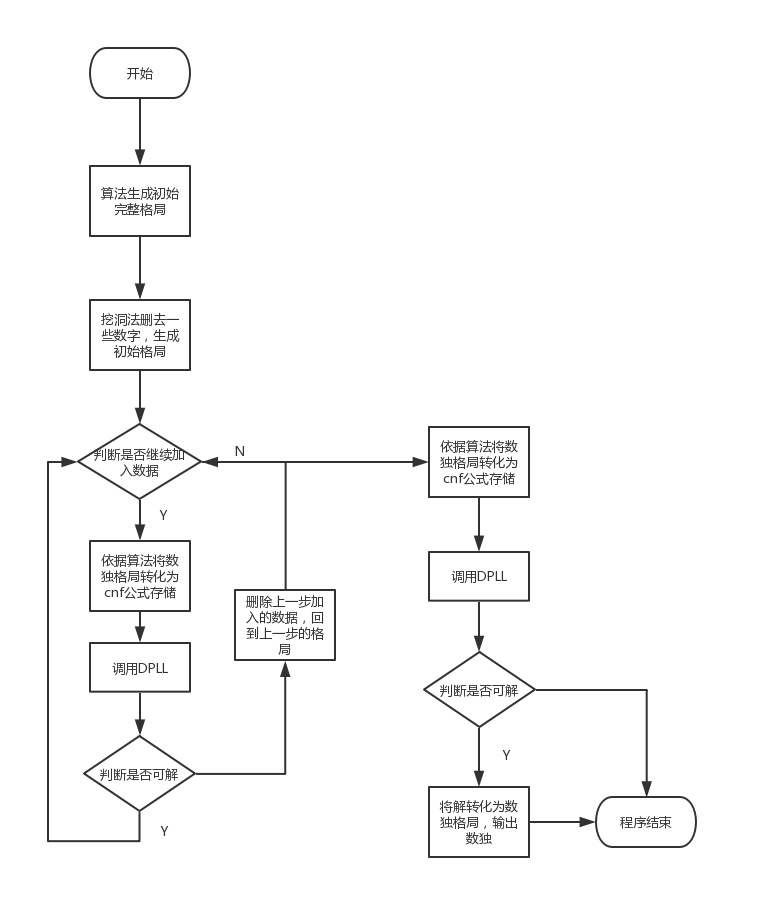


图3-7 数独模块函数算法流程图

# 4系统实现与测试

## 4.1系统实现

## 4.1.1系统概况：

编程环境为DEV C++；

本机环境：Windows 10;

程序语言：C语言；

程序源码详见附录；

## 4.1.2系统使用的函数支持：

<stdio.h>

<stdlib.h>

<malloc.h>

<math.h>

<time.h>

## 4.1.3程序所包含的数据类型（C语言描述）：

根据3.1的数据结构设计，数据类型定义如下：

typedef struct root {//根

int varnum;//变元数

int calnum;//子句数

struct clause\* fircal;//指向第一个子句

}\*Root;//根类型

typedef struct clause {//子句头

int litnum;//该子句变元个数

struct clause\* nextcal;//指向下一个子句

struct literal\* first;//指向第一个文字

int clamark;//标记子句是否删除

}clause;//子句类型

typedef struct literal {//文字

int liter;//变元

struct literal\* next;//指向下一个文字

int litmark;//标记是否删除

}literal;//文字类型

typedef struct word {//变元表

clause \*clahead;//指向该变元所在的子句头

literal\* liter;//指向该变元所在的文字

int L;//该变元所在的子句序号

word \*next;//指向下一个相同变元

}word, \*Word;//变元表类型

## 4.1.4系统中包含的函数：

1. status Loadcnf(char\*filename, Root\* root, int flag),该函数功能为读取解析cnf文件，基于设计的物理结构进行内部表示，参数传址；
2. status Testcnf(Root root);该函数为测试解析的正确性，输出内部表示人工验证；
3. status DPLL(Root root, int \*valuation);该函数为DPLL算法的主函数，内部进行函数调用，解SAT问题，参数为公式内部表示的起始地址，将变元的解存储在valuation中；
4. status evaluateClause(Root root, int L, int \*value);该函数是为了判断公式是否存在冲突 ，即是否存在单子句冲突或者含有空子句；
5. status createsudoku();该函数为数独游戏函数，函数内部用算法自动生成数独格局，同时将数独格局转化为cnf公式，函数内部也解数独，将解再次转化为数独格局，具有人机交互功能；
6. status savefile(char \*filename, int \*valuation, Root root, int flag, double t)；该函数是为了将输出结果按任务书要求保存在与cnf公式同名的res文件中
7. int variable\_decide(Root root, int \*valuation);该函数是为了DPLL算法中的决策变量，决策机制为选取单子句或者出现最短子句第一个变元 ；
8. void liternum(Root root, Word \*lb);该函数是为了优化DPLL算法，遍历cnf公式，建立变元表，方便公式化简和恢复的时候不用遍历整个公式；同时统计每个子句变元个数，为决策变量做准备；
9. void cnfdeduce(Root \*root, int L, int flag, Word \*lb);该函数根据已经赋值的变量和赋的值flag化简公式，及删除子句或文字；
10. void cnfrestore(Root \*root, int L, Word \*lb);/该函数是为了DPLL算法中回溯操作，根据翻转的变量L，恢复cnf公式，即增加子句或文字；

## 4.1.5系统中的函数间的组织和调用关系

根据系统总体设计，系统中各模块的功能用用这些函数完成，cnf文件读取模块直接调用函数Loadcnf直接完成，验证cnf公式解析正确性调用函数Testcnf即可，求解SAT的主要模块DPLL算法可调用函数DPLL完成，保存文件调用函数savefile完成，生成数独格局，输出数独公式文件和解数独等与数独有关的模块均封装在函数createsudoku中，调用即可完成数独模块；功能与函数的调用关系如下（如图4-1）

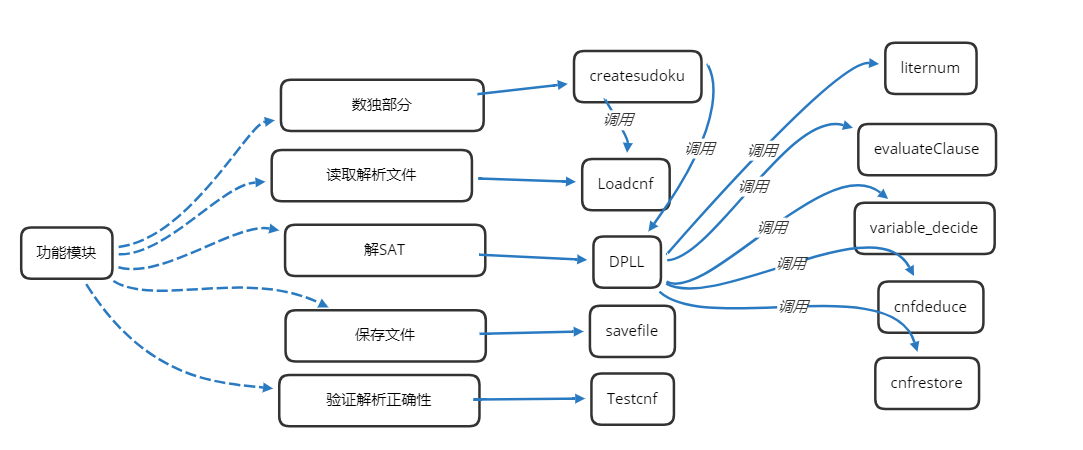


图4-1 功能模块与函数的组织关系图

整个系统需要包含两个大模块，SAT和数独部分，而数独部分又依赖于SAT部分，因此createsudoku函数中调用了SAT相关函数；求解SAT模块主要是先调用函数Loadcnf读取解析cnf公式，而后用户判断是否需要验证解析正确性，需要验证则调用函数Testcnf,接着调用函数DPLL解cnf公式，结果保存调用函数savefile；DPLL中需要调用函数evaluateClause判断公式是否存在冲突，调用函数variable\_decide决策变量，调用函数liternum建文字表以及统计每个子句变元个数，函数cnfdeduce和函数cnfrestore化简和恢复cnf公式，整个模块依据一定的流程来运行；数独部分则是先生成初始数独格局，经过人机交互用户输入相关信息后，调用函数DPLL求解数独后再次转化为数独格局；

函数间的调用关系如下（如图4-2）

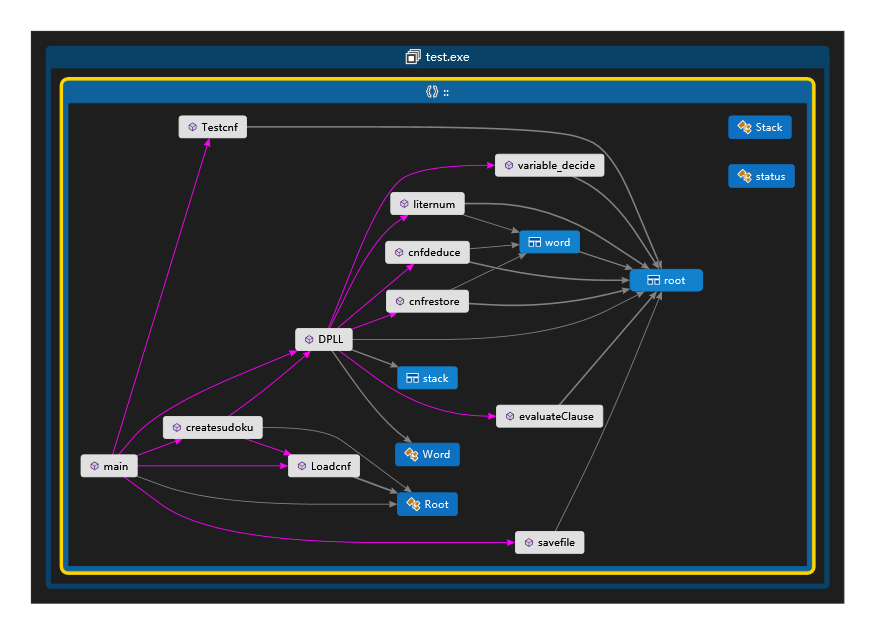


图4-2 函数间的调用关系

## 4.2系统测试

1. 文件读取解析与验证测试

该模块功能为读取cnf公式并解析，基于物理结构内部表示，验证过程即遍历内部表示输出；对于大的cnf算例，验证过程需要人工检查，较为麻烦，因此选用小算例，检查过程简单；

共设计三个测试样例进行测试，测试步骤及结果如表4-3所示

表4-3 文件读取解析与验证用例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例 | 文件名 | 变元数 | 子句数 | 文件是否符合格式要求 | 是否验证 |
| 用例1 | 1.cnf | 1 | 2 | 是 | 是 |
| 理论结果 | | | | 运行结果 | |
|  | | | |  | |
| 用例 | 文件名 | 变元数 | 子句数 | 文件是否符合格式要求 | 功能测试 |
| 用例2 | 2.cnf | 1 | 2 | 是 | 否 |
| 理论结果 | | | | 运行结果 | |
|  | | | |  | |
| 用例 | 文件名 | 变元数 | 子句数 | 文件是否符合格式要求 | 是否验证 |
| 用例3 | 3.cnf | 1 | 2 | 否 | 无需验证 |
| 理论结果 | | | | 运行结果 | |
|  | | | |  | |

经测试，该模块运行正常，理论结果与运行结果相同。

1. SAT测试

该部分为解SAT公式模块，具有判断cnf满足或者不满足的功能，满足则输出解，不满足输出UNSAT，同时求得优化前时间和优化后时间，求出优化率；之后保存解到文件，因为做测试用例，运行时间不宜过长，但尽量选择大用例；同时测试保存文件的功能；不满足算例测试时间较长，因此选择小算例，但尽量选择大算例；

此部分共设计31个算例，期中满足算例26个，不满足算例5个

测试步骤及运行结果如表4-4所示

表4-4 SAT部分测试用例表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 用例 | 文件名(.cnf) | 变元数 | 子句数 | 满足or不满足 | t(ms)  优化前 | T(ms)  优化后 | 优化率 |
| 用例1 | sat-20 | 20 | 91 | 满足 | 1 | 0 | 100% |
|  | | | | | | | |
| 用例2 | unsat-5cnf-30 | 30 | 420 | 不满足 | 170 | 30 | 82.353% |
|  | | | | | | | |
| 用例3 | ais10 | 181 | 3151 | 满足 | 23171.0 | 4114 | 82.245% |
|  | | | | | | | |
| 用例4. | sud00009 | 303 | 2851 | 满足 | 29817 | 0 | 100% |
|  | | | | | | | |
| 用例5. | 7cnf20\_90000 | 20 | 1532 | 满足 | 6 | 73 | -1116.7% |
|  | | | | | | | |
| 用例6. | problem1-20 | 20 | 91 | 满足 | 1 | 0 | 100% |
|  | | | | | | | |
| 用例7 | problem2-50 | 50 | 80 | 满足 | 49 | 0 | 100% |
|  | | | | | | | |
| 用例8. | problem3-100 | 100 | 340 | 满足 | 2931 | 30 | 98.976% |
|  | | | | | | | |
| 用例9 | problem8-50 | 50 | 300 | 满足 | 10 | 0 | 100% |
|  | | | | | | | |
| 用例10. | problem9-100 | 100 | 200 | 满足 | 15800 | 10 | 99.937% |
|  | | | | | | | |
| 用例11. | problem11-100 | 100 | 600 | 满足 | 69 | 10 | 85.5072% |
|  | | | | | | | |
| 用例12. | tst\_v25\_c100 | 25 | 100 | 满足 | 0 | 0 | 100% |
|  | | | | | | | |
| 用例13. | sud00861 | 297 | 2721 | 满足 | 26098 | 60 | 99.770% |
|  | | | | | | | |
| 用例14 | sud00082 | 224 | 1762 | 满足 | 5695 | 93 | 98.367% |
|  | | | | | | | |
| 用例15 | sud00079 | 301 | 2810 | 满足 | 54014 | 16 | 99.970% |
|  | | | | | | | |
| 用例16 | sud00001 | 301 | 2780 | 满足 | 233676 | 26 | 99.988% |
|  | | | | | | | |
| 用例17 | problem12-200 | 200 | 1200 | 满足 | 211622 | 36 | 99.983% |
|  | | | | | | | |
| 用例18 | sud00021 | 308 | 2911 | 满足 | 153155 | 150 | 99.902% |
|  | | | | | | | |
| 用例19 | sud00012 | 232 | 1901 | 满足 | 4508 | 114 | 97.471% |
|  | | | | | | | |
| 用例20 | sud00009 | 303 | 2851 | 满足 | 30319 | 10 | 99.967% |
|  | | | | | | | |
| 用例21 | tst\_v200\_c210 | 200 | 210 | 满足 | 0 | 0 | 100% |
|  | | | | | | | |
| 用例22 | ais12 | 265 | 5666 | 满足 | 超时 | 19456 | 100% |
|  | | | | | | | |
| 用例23 | anomaly | 48 | 261 | 满足 | 0 | 0 | 100% |
|  | | | | | | | |
| 用例24 | flat30-1 | 90 | 300 | 满足 | 0 | 0 | 100% |
|  | | | | | | | |
| 用例25 | bw\_large.a | 459 | 4657 | 满足 | 1840 | 3421 | -85.923% |
|  | | | | | | | |
| 用例26 | bmc-ibm-2.cnf | 2810 | 11683 | 满足 | 超时 | 1230 | 100% |
|  | | | | | | | |
| 用例27 | php-010-008 | 80 | 370 | 不满足 | 11360 | 6071 | 46.558% |
|  | | | | | | | |
| 用例28 | tst\_v10\_c100 | 10 | 100 | 不满足 | 0 | 0 | 100% |
|  | | | | | | | |
| 用例29 | u-5cnf\_3500\_ | 30 | 420 | 不满足 | 160 | 30 | 81.25% |
|  | | | | | | | |
| 用例30 | u-problem7-50 | 50 | 100 | 不满足 | 85 | 10 | 88.235% |
|  | | | | | | | |
| 用例31 | u-problem10-100 | 100 | 200 | 不满足 | 61656 | 3 | 99.99% |
|  | | | | | | | |

经测试，该模块运行正常，运行结果与理论结果相同。

1. 数独测试

该模块具有与数独的一些功能，例如初始化数独格局，解出数独等，能够人机交互进行游戏；

共设计4个测试样例进行测试，测试步骤及结果如表4-5所示

表4-5 数独模块测试样例表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 测试功能 | 理论结果 | 运行结果 |
| 用例1 | 初始化数独格局 | 成功初始化数独格局 |  |
| 用例2 | 人工加入数据 | 加入数据后数独错误返回加入前的状态 |  |
| 用例3 | 人工加入数据 | 加入数据后数独正确成功加入 |  |
| 用例4 | 直接看答案 | 直接输出数独的解 |  |

经测试，该模块运行正常，理论结果与运行结果相同；

# 5总结与展望

## 5.1全文总结

本次课程设计在学习了SAT问题的基础上，对DPLL算法进行了学习与优化。

综上所述，本文研究了以下内容：

1. 对SAT问题的研究背景，意义及现状进行了简要阐述；
2. 学习了命题逻辑可满足性问题的基本理论知识；
3. 学习了一些经典的数据结构和文件读写操作与物理存储的建构；
4. 对该课题中所需的数据结构与一些需要的算法进行了设计；
5. 对DPLL算法变量决策进行了较深入的学习与分析，思考更高效的变量决策
6. 对DPLL算法进行了不断优化，能够正确求解SAT问题，但仍存在许多不足，例如求解某些算例求解时间太长；
7. 将DPLL用于解数独游戏。
8. 对数独游戏进行了思考，设计算法自动生成数独格局并转化为cnf文件，设计人机交互。
9. 对自己的求解器的正确性做了测试。

## 5.2工作展望

基于DPLL的求解器在改进后性能上有了很大的改善，但求解速度还是慢，难以满足人们的需求，现如今的求解器用一些比较巧妙，快速的DPLL算法，这里面有很多值得我去学习研究。

在今后的研究中，围绕着如下几个方面开展工作：

1. 为算法加入更高级的变量决策机制；
2. 加入一些cnf预处理器，简化对cnf公式的化简；
3. 学习一些子句学习机制，提高DPLL的速度；
4. 可以设置一些更巧妙的数据结构；

# 6.体会

在本次程序设计综合课城设计中，我学习到了很多知识，例如DPLL算法，SAT问题，回溯机制等，同时也了解到了自己本身的诸多不足，例如写程序不够认真,不仔细，导致很多bug，结果也是浪费自己的时间。从最开始的不懂DPLL,不知道课程设计如何着手开始，到写出来最初版本，再到不断调试，优化改进DPLL算法，我能感觉到自己编程能力的提高，学习能力的提升，对问题研究的态度的转变。对回溯机制的不懂，再到把回溯更好的应用去DPLL，我认识到学习的重要性，也了解到回溯的重要性。综合来讲，本次程序设计综合课设，不光是对我编程能力的一次提高，更是对我综合能力的一次突破！

# 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. Masterthesis, 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. 严蔚敏等. 数据结构(C语言版). 清华大学出版社
7. [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005
8. 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社,2014:192～197
9. 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

#include<time.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

typedef int status;

typedef struct root {//根

int varnum;//变元数

int calnum;//子句数

struct clause\* fircal;//指向第一个子句

}\*Root;//根类型

typedef struct clause {//子句头

int litnum;//该子句变元个数

struct clause\* nextcal;//指向下一个子句

struct literal\* first;//指向第一个文字

int clamark;//标记子句是否删除

}clause;//子句类型

typedef struct literal {//文字

int liter;//变元

struct literal\* next;//指向下一个文字

int litmark;//标记是否删除

}literal;//文字类型

typedef struct word {//变元表

clause \*clahead;//指向该变元所在的子句头

literal\* liter;//指向该变元所在的文字

int L;//该变元所在的子句序号

word \*next;//指向下一个相同变元

}word, \*Word;//变元表类型

typedef struct stack {//栈，方便回溯时使用

int literal;//存入赋值的值

int value;//变元的值

}\*Stack;//栈指针，动态分配数组

status Loadcnf(char\*filename, Root\* root, int flag);//解析cnf文件

status Testcnf(Root root);//测试解析是否正确

status DPLL(Root root, int \*valuation);//DPLL算法

status evaluateClause(Root root, int L, int \*value);//判断公式是否存在冲突

status createsudoku();//数独游戏，算法自动生成数独格局，同时解数独，具有人机交互

status savefile(char \*filename, int \*valuation, Root root, int flag, double t);//将输出结果保存在res文件中

int variable\_decide(Root root, int \*valuation);//决策变量，选取单子句或者出现最短子句第一个变元

void liternum(Root root, Word \*lb);//遍历，建立变元表，同时统计每个子句变元个数

void cnfdeduce(Root \*root, int L, int flag, Word \*lb);//化简cnf公式，删除

void cnfrestore(Root \*root, int L, Word \*lb);//回溯cnf公式。回溯时

int sudoku[9][9];//数独格局

int litnum[10000000];//子句个数，人工设定。若子句个数大于，则增加数组大小

int main(void) {

clock\_t start, finish;//时间

double tim, tim2;

tim2=0;

int op = 1;

int i = 0;

int j, k, w; int oooo = 0;

int flag; int select = 0;

char filename[60];

FILE\*fp;

while (op) {

system("cls"); printf("\n\n");

printf(" Menu for 综合课设 \n");

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

printf(" 1. SAT 2. sudoku\n 0.EXIT\n");

scanf("%d", &op);

switch (op) {

case 1:

Root root;

if (Loadcnf(filename, &root, 1)) {

printf("该公式包含%d个变元， %d个子句！\n", root->varnum, root->calnum);

printf("是否需要验证！\n1.YES 0.NO\n");

scanf("%d", &select);

if (select == 1) {

printf("输入回车开始验证！\n");

if (Testcnf(root))

printf("遍历完成！\n");

}

getchar();

printf("输入回车开始求解！\n");

getchar();

printf("求解ing...\n");

}

else return 0;

int \*valuation;

valuation = (int\*)malloc(sizeof(int)\*(root->varnum + 1));

valuation[0] = root->varnum;

for (i = 1; i <= root->varnum; i++) {

valuation[i] = 0;

}

start = clock();

printf("文件：%s\n",filename);

if (DPLL(root, valuation)) {

finish = clock();

flag = 1;

printf("SAT!\n");

}

else {

flag = 0;

finish = clock();

printf("UNSAT！\n");

}

tim = (double)(finish - start);

printf("DPLL的执行时间是%fms.\n", tim);

printf(" %fs.\n", tim / CLK\_TCK);

printf("如需要计算优化率，请确定该sat已在优化前.cpp运行！\n");

printf("是否完成上述要求\n0:NO(不计算优化率) 1.YES(计算且已完成优化前的运行)\n");

scanf("%d", &oooo);

if (oooo) {

if(fp = fopen("t.txt", "r")){

fscanf(fp, "%lf", &tim2);

}

else {

printf("优化前未运行。\n");

}

fclose(fp);

printf("优化前的执行时间为%fms\n", tim2);

if(tim2==0){

printf("优化率为%f%%\n", (1 - tim) / 1.0 \* 100);

}

else printf("优化率为%f%%\n", (tim2 - tim) / tim2 \* 100);

getchar();

}

savefile(filename, valuation, root, flag, tim);

getchar();

break;

case 2:

if (createsudoku()) {

printf("GAME OVER！\n");

}

getchar();

break;

case 0:

break;

}

}

printf("欢迎再次体验本系统！\n");

return 0;

}

status DPLL(Root root, int \*valuation) {//DPLL解SAT

int t, flag, L;

int flag3=0;

int i = 0;

int j = 0;

flag = 1;

Stack visit;//栈

Word lb;//新建一个链表

lb = (Word)malloc(sizeof(word)\*(2 \* root->varnum + 1));//变元表

visit = (Stack)malloc(sizeof(stack)\*(root->varnum + 1));//栈，方便回溯

for (i = 0; i < root->varnum + 1; i++) {//初始化

visit[i].literal = 0;

visit[i].value = 0;

lb[i].clahead = NULL;

lb[i].liter = NULL;

lb[i].next = NULL;

lb[i].L = 0;

}

for (i; i <= 2 \* root->varnum; i++) {///初始化

lb[i].clahead = NULL;

lb[i].liter = NULL;

lb[i].next = NULL;

lb[i].L = 0;

}

liternum(root, &lb);//建立变元表

i = 0;

L = variable\_decide(root, valuation);//决策变量

visit[i].literal = L;

visit[i].value = valuation[L];

i++;

while (1) {

if (flag3!=-2&&evaluateClause(root, L, valuation)){//flag==-2表示存在空子句，判断是否存在冲突

cnfdeduce(&root, L, valuation[L], &lb);//化简

if ((L = variable\_decide(root, valuation))==-1) //如果子句集为空，则说明可满足， 如果存在空子句，L==-2,

return OK; //可满足

else if(L!=-2){//L!=-2，说明决策出变量，入栈

visit[i].literal = L;

visit[i].value = valuation[L];

i++;

}

else{//存在冲突

flag3=-2;//含有空子句，不满足

}

}

else {//回溯，

flag3=0;

for (j = i - 1; j >= 0; j--) {

t = visit[j].literal;

if (visit[j].value == 0) {//栈顶是0，则说明已翻转，

visit[j].value = 1;//恢复

valuation[t] = 1;

valuation[0]++;

if(j!=i-1)cnfrestore(&root, t,&lb);

continue;

}

else if (visit[j].value == 1) {//栈里第一个未翻转变量

visit[j].value = 0;//翻转

valuation[t] = 0;

L = t;

if(j!=i-1)cnfrestore(&root, L,&lb);//恢复

i = j + 1;

break;

}

}

if (j == -1) return ERROR;

}

}

}

status evaluateClause(Root root, int L, int \*value) {//判断是否一定存在冲突，单子句冲突或含有空子句

clause\*p;

literal\*q;

p = root->fircal;

int i = 0;

int t;

for (i = 0; i < root->calnum; i++) {

if (p->clamark == 0 && p->litnum == 1) {

q = p->first;

while (q) {

if (q->litmark == 0) {

t = q->liter;

if (abs(t) == L) {

if (t > 0 && value[L] == 0) return ERROR;

else if (t < 0 && value[L]>0) return ERROR;

break;

}

else break;

}

q = q->next;

}

}

else if(p->litnum==0) return ERROR;

p = p->nextcal;

}

return OK;

}

void cnfrestore(Root \*root, int L, Word \*lb) {//子句恢复

word \*p, \*q;

clause\*r;

literal\*t;

p = (\*lb)[L].next;

q = (\*lb)[L + (\*root)->varnum].next;

while (p) {

r = p->clahead;

t = p->liter;

if (r->clamark == L)r->clamark = 0;

if (t->litmark == L) {

t->litmark = 0;

r->litnum++;

litnum[p->L]++;

}

p = p->next;

}

while (q) {

r = q->clahead;

t = q->liter;

if (r->clamark == L)r->clamark = 0;

if (t->litmark == L) {

t->litmark = 0;

r->litnum++;

litnum[q->L]++;

}

q = q->next;

}

}

void cnfdeduce(Root \*root, int L, int flag, Word \*lb) {//子句化简

word \*p, \*q;

if (flag == 1) {

p = (\*lb)[L].next;

q = (\*lb)[L + (\*root)->varnum].next;

}

else {

p = (\*lb)[L + (\*root)->varnum].next;

q = (\*lb)[L].next;

}

clause\*r;

literal\*t;

while (p) {//向后遍历

r = p->clahead;

if (r->clamark == 0)//未删除

r->clamark = L;//标记更改表示删除改

p = p->next;

}

while (q) {//向后遍历

r = q->clahead;

t = q->liter;

if (r->clamark == 0) {//未删除

r->litnum--;

litnum[q->L]--;//该子句变元数目减1

t->litmark = L;//标记更改表示删除

}

q = q->next;

}

}

int variable\_decide(Root root, int \*valuation) {//决策变量

int ooo=valuation[0];

int L, t, max;int visit=0;//查看cnf是否为空

int i = 0;L=-2;

clause\*p, \*r;

literal \*q;

int min;

p = root->fircal;

r = p;

for (i = 0; i < root->calnum; i++) {

if (p->clamark == 0) {//子句未删除

if(p->litnum==0) return -2;//存在空子句返回-2

min = p->litnum;

r = p;

visit=1;

break;

}

p = p->nextcal;

}

for (; i < root->calnum; i++) {

if (p->clamark == 0) {//子句未删除

visit=1;

if(p->litnum==0) return -2;//存在空子句返回-2

if (min > litnum[i + 1]) {

min = litnum[i + 1];

r = p;

}

if (p->litnum == 1) {

q = r->first;

while (q) {

if (q->litmark == 0) {

L = q->liter;

t = abs(L);

if (L > 0) {

valuation[t] = 1;

valuation[0]--;

}

else {

valuation[t] = 0;

valuation[0]--;

}

return t;//返回已赋值的变元

}

q = q->next;

}

}

}

p = p->nextcal;

}

q = r->first;

while (q) {

if (q->litmark == 0){

L = abs(q->liter);//最短子句的第一个未删除变元

break;

}

q = q->next;

}

if(visit==0){

L=-1;//子句集为空

return L;

}

if(L>0){

valuation[L] = 1;//默认赋值为1

valuation[0]--;

}

return L;

}

void liternum(Root root, Word \*lb) {//构建变元表

clause\*p;

word \*t, \*w;

literal\*r;

int i, j;

int data;

int flag = 0;

p = root->fircal;

for (i = 0, j = 0; i < root->calnum; i++) {

r = p->first;

litnum[i + 1] = p->litnum;

while (r) {

t = (word\*)malloc(sizeof(word));

t->clahead = p;

t->liter = r;

t->L = i + 1;

data = abs(r->liter);

if (r->liter > 0) {

if ((\*lb)[data].next == NULL) {

(\*lb)[data].next = t;

t->next = NULL;

}

else {

t->next = (\*lb)[data].next;

(\*lb)[data].next = t;

}

}

else {

if ((\*lb)[root->varnum + data].next == NULL) {

(\*lb)[root->varnum + data].next = t;

t->next = NULL;

}

else {

t->next = (\*lb)[data + root->varnum].next;

(\*lb)[root->varnum + data].next = t;

}

}

r = r->next;

}

p = p->nextcal;

}

}

status Testcnf(Root root) {//验证解析正确性

clause\* p;

literal \*r;

int i = 0;

p = root->fircal;

for (; i < root->calnum; i++) {

r = p->first;

while (r) {

printf("%d ", r->liter);

r = r->next;

}

printf("\n");

p = p->nextcal;

}

return OK;

}

status Loadcnf(char \*filename, Root\* root, int flag) {//读取解析cnf公式

char c, d;

int flag2=0;

clause\* p, \*t;

literal\*q, \*r;

int data;

int i = 0;

if (flag) {

printf("输入cnf文件名！\n");

scanf("%s", filename);

}

FILE \*fp;

if ((fp = fopen(filename, "r")) == NULL) {

printf("File open error\n ");

return ERROR;

}

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

if (c == 'p') {

do {

flag2=1;

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

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

(\*root) = (Root)malloc(sizeof(root));

(\*root)->calnum = 0;

(\*root)->varnum = 0;

fscanf(fp, "%d %d", &((\*root)->varnum), &((\*root)->calnum));

if ((\*root)->calnum) {

break;

}

} while (fscanf(fp, "%c", &c) != EOF);

}

else {

while (fscanf(fp, "%c", &c) != EOF) {

if (c == 'p') {

flag2=1;

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

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

(\*root) = (Root)malloc(sizeof(root));

(\*root)->calnum = 0;

(\*root)->varnum = 0;

fscanf(fp, "%d %d", &((\*root)->varnum), &((\*root)->calnum));

if ((\*root)->calnum) {

break;

}

}

}

}

if ((\*root)->calnum) {

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

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

t = p;

p->litnum = 1;

p->clamark = 0;flag=1;

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

p->first = q;

r = q;

q->liter = data;

q->litmark = 0;

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

while (data) {

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

q->liter = data;

q->litmark = 0;

r->next = q;flag=1;

r = q;

p->litnum++;

p->clamark = 0;

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

}

r->next = q;

q->next = NULL;

(\*root)->fircal = p;

p->nextcal = NULL;

}

if ((\*root)->calnum > 1) {

for (i = 0; i < (\*root)->calnum; i++) {

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

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

p->litnum = 1;

p->clamark = 0;

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

p->first = q;

r = q;

q->liter = data;

q->litmark = 0;

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

while (data != 0) {

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

q->liter = data;

q->litmark = 0;

r->next = q;

r = q;

p->litnum++;

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

}

r->next = q;

q->next = NULL;

t->nextcal = p;

t = p;

p->nextcal = NULL;

}

}

fclose(fp);

if(flag2){

printf("文件加载完成！\n");

return OK;

}

else printf("文件加载失败");

return ERROR;

}

status savefile(char \*filename, int \*valuation, Root root, int flag, double t) {//保存文件

FILE \*fp;

int i = 0;

for (i = 0; i < 60; i++) {//将文件名后缀改成.res

if (filename[i] == '\0') {

filename[i -3 ] = 'r';

filename[i -2] = 'e';

filename[i -1] = 's';

break;

}

}

if ((fp = fopen(filename, "w")) == NULL) {

printf("File open error\n ");

return ERROR;

}

fprintf(fp, "s %d\nv ", flag);//以下按要求存储数据

if (flag) {

for (i = 1; i < root->varnum + 1; i++) {

if (valuation[i] == 1)fprintf(fp, "%d ", i);//为1 则存正值

else fprintf(fp, "-%d ", i);//否则存负值

}

}

fprintf(fp, "\n");

fprintf(fp, "t %f", t);

fclose(fp);

printf("结果保存成功！\n");

return OK;

}

status createsudoku() {//数独模块

int ls[3][3];

int num = 0;

int i, j, k, t;

int r, s, w;

int op = 0;

char filename[60];

FILE \*slove, \*fp;

Root root2;

int \*valuation2;

srand((unsigned)time(NULL));

k = rand();

for (i = 3; i < 6; i++) {//第五个宫格//

for (j = 3; j < 6; j++) {

ls[i - 3][j - 3] = (k % 9) + 1;

sudoku[i][j] = (k % 9) + 1;

k++;

}

}

//根据矩阵的行列变换生成数独初始完整格局

//以下顺序不可变

for (i = 3; i < 6; i++) {//第四个宫格

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

sudoku[i][j] = sudoku[(i + 2) % 3 + 3][j + 3];

}

}

for (i = 3; i < 6; i++) {//第六个宫格

for (j = 6; j < 9; j++) {

sudoku[i][j] = sudoku[(i + 1) % 3 + 3][j - 3];

}

}

for (i = 0; i < 3; i++) {//第二个宫格

for (j = 3; j < 6; j++) {

sudoku[i][j] = ls[i][(j + 2) % 3];

}

}

for (i = 0; i < 3; i++) {//第一个宫格

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

sudoku[i][j] = sudoku[i + 3][(j + 2) % 3];

}

}

for (i = 0; i < 3; i++) {//第三个宫格

for (j = 6; j < 9; j++) {

sudoku[i][j] = sudoku[i + 3][(j - 1) % 3 + 6];

}

}

for (i = 6; i < 9; i++) {//第7个宫格

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

sudoku[i][j] = sudoku[i - 3][(j + 1) % 3];

}

}

for (i = 6; i < 9; i++) {//第8个宫格

for (j = 3; j < 6; j++) {

sudoku[i][j] = sudoku[i - 3][(j + 1) % 3 + 3];

}

}

for (i = 6; i < 9; i++) {//第九个宫格

for (j = 6; j < 9; j++) {

sudoku[i][j] = sudoku[i - 3][(j + 1) % 3 + 6];

}

}

slove = fopen("slove.txt", "w");

for (i = 0; i < 9; i++) {//将原始完整数独保存

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

fprintf(slove, "%d ", sudoku[i][j]);

}

fprintf(slove, "\n");

}

for (i = 81; i > 0; i--) {

k = rand() % 81;

sudoku[k / 9][k % 9] = 0;

}

printf("目前的数独格局如下：\n");

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

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

if (sudoku[i][j])

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

else printf("\_ ");

}

printf("\n");

}

printf("是否要继续加入数据\n");

printf("0.NO直接看答案 1.YES我要做数独\n");

scanf("%d", &op);

while (op) {//加数据的话

printf("输入行列值\n");

scanf("%d %d %d", &r, &s, &w);

sudoku[r - 1][s - 1] = w;//加的数据为w

getchar();

if ((fp = fopen("sudoku.cnf", "w")) == NULL)

{

printf("File open error\n ");

}

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

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

if (sudoku[i][j]) t++;

}

}

//将数独变为cnf公式保存在sudoku.cnf文件中

fprintf(fp, "p cnf 729 %d\n", t + 10530);

for (i = 0; i < 9; i++) {//保存已经存在的数

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

if (k = sudoku[i][j]) {//处理本来存在的单子句

fprintf(fp, "%d 0\n", 81 \* i + 9 \* j + k);//按照 81 \* i + 9 \* j + k的规则将第i+1行j+1列值为k转换成变元

}

}

}

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

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

for (k = 1; k < 10; k++) {

fprintf(fp, "%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);//每个cell必为1-9

}

fprintf(fp, "0\n");

}

}

for (i = 1; i < 10; i++) {//

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

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

for (t = k + 1; t < 10; t++) {

fprintf(fp, "-%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);//每个cell不可能同时为1 2或1 3......

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

}

}

}

}

for (i = 1; i < 10; i++) {//对行处理，每一行一定存在1-9&&每一行不存在相同的数

for (k = 1; k < 10; k++) {

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

fprintf(fp, "%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

}

fprintf(fp, "0\n");

}

for (k = 1; k < 10; k++) {//每一行不存在相同的数

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

for (t = j + 1; t < 10; t++) {

fprintf(fp, "-%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (i - 1) + 9 \* (t - 1) + k);

}

}

}

}

for (j = 1; j < 10; j++) {//对列处理 方式同行

for (k = 1; k < 10; k++) {

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

fprintf(fp, "%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

}

fprintf(fp, "0\n");

}

for (k = 1; k < 10; k++) {

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

for (t = i + 1; t < 10; t++) {

fprintf(fp, "-%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (t - 1) + 9 \* (j - 1) + k);

}

}

}

}

int m, n;

//对3\*3块处理

for (m = 0; m < 3; m++) {//第m+1行的3\*3块

for (n = 0; n < 3; n++) {//第n+1列的3\*3块

for (k = 1; k < 10; k++) {

for (i = 1; i < 4; i++) {

for (j = 1; j < 4; j++) {

fprintf(fp, "%d ", 81 \* ((i - 1) + 3 \* m) + 9 \* ((j - 1) + 3 \* n) + k);//每个块必有1 2 3 4 ....9

}

}

fprintf(fp, "0\n");

}

//但每个块不可能有相同数字

for (k = 1; k < 10; k++) {//余下单独处理

i = 1;

j = 1;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j + 1) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j + 1) + k);

j = 2;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j) + k);

j = 3;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 1 - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 1 - 1) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j - 3) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j - 2) + k);

i = 2;

j = 1;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j + 1) + k);

j = 2;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

j = 3;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 3) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 2) + k);

}

}

}

fclose(fp); fclose(slove);

strcpy(filename, "sudoku.cnf");

if (Loadcnf(filename, &root2, 0)) {//调用DPLL解数独

num = root2->varnum + 1;

valuation2 = (int\*)malloc(sizeof(int)\*(num));

valuation2[0] = root2->varnum;

for (i = 1; i <= root2->varnum; i++) {

valuation2[i] = 0;

}

system("cls");

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

if (DPLL(root2, valuation2) == 0) {//如果上一步赋的值数独不成立则返回恢复上一步的格局

printf("该步骤无解！\n请重新输入！\n");

sudoku[r - 1][s - 1] = 0;

}

else {

printf("可解！\n");

}

printf("\n\n目前的数独格局如下：\n");

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

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

if (sudoku[i][j])

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

else printf("\_ ");

}

printf("\n");

}

}

free(valuation2);

valuation2 = NULL;

printf("是否要继续加入数据\n");

printf("0.结束填补直接看答案 1.继续填补\n");

scanf("%d", &op);

}

if ((fp = fopen("sudoku.cnf", "w")) == NULL)

{

printf("File open error\n ");

}

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

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

if (sudoku[i][j]) t++;

}

}

//将数独变为cnf公式保存在sudoku.cnf文件中

//以下注释同上

fprintf(fp, "p cnf 729 %d\n", t + 10530);

for (i = 0; i < 9; i++) {//保存已经存在的数

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

if (k = sudoku[i][j]) {

fprintf(fp, "%d 0\n", 81 \* i + 9 \* j + k);

}

}

}

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

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

for (k = 1; k < 10; k++) {

fprintf(fp, "%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

}

fprintf(fp, "0\n");

}

}

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

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

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

for (t = k + 1; t < 10; t++) {

fprintf(fp, "-%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

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

}

}

}

}

for (i = 1; i < 10; i++) {//对行处理，每一行一定存在1-9&&每一行不存在相同的数

for (k = 1; k < 10; k++) {

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

fprintf(fp, "%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

}

fprintf(fp, "0\n");

}

for (k = 1; k < 10; k++) {

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

for (t = j + 1; t < 10; t++) {

fprintf(fp, "-%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (i - 1) + 9 \* (t - 1) + k);

}

}

}

}

for (j = 1; j < 10; j++) {//对列处理 方式同行

for (k = 1; k < 10; k++) {

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

fprintf(fp, "%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

}

fprintf(fp, "0\n");

}

for (k = 1; k < 10; k++) {

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

for (t = i + 1; t < 10; t++) {

fprintf(fp, "-%d ", 81 \* (i - 1) + 9 \* (j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (t - 1) + 9 \* (j - 1) + k);

}

}

}

}

int m, n;

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

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

for (k = 1; k < 10; k++) {

for (i = 1; i < 4; i++) {

for (j = 1; j < 4; j++) {

fprintf(fp, "%d ", 81 \* ((i - 1) + 3 \* m) + 9 \* ((j - 1) + 3 \* n) + k);

}

}

fprintf(fp, "0\n");

}

for (k = 1; k < 10; k++) {

i = 1;

j = 1;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j + 1) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j + 1) + k);

j = 2;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j) + k);

j = 3;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 1 - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 1 - 1) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j - 3) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i + 1) + 9 \* (3 \* n + j - 2) + k);

i = 2;

j = 1;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j + 1) + k);

j = 2;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 2) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j) + k);

j = 3;

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 3) + k);

fprintf(fp, "-%d ", 81 \* (3 \* m + i - 1) + 9 \* (3 \* n + j - 1) + k);

fprintf(fp, "-%d 0\n", 81 \* (3 \* m + i) + 9 \* (3 \* n + j - 2) + k);

}

}

}

fclose(fp); fclose(slove);

strcpy(filename, "sudoku.cnf");

if (Loadcnf(filename, &root2, 0)) {

valuation2 = (int\*)malloc(sizeof(int)\*(root2->varnum + 1));

valuation2[0] = root2->varnum;

for (i = 1; i <= root2->varnum; i++) {

valuation2[i] = 0;

}

system("cls");

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

printf("\n\n目前的数独格局如下：\n");

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

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

if (sudoku[i][j])

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

else printf("\_ ");

}

printf("\n");

}

if (DPLL(root2, valuation2)) {

for (w = 1; w < 730; w++) {

if (valuation2[w]) {

k = w % 9;

if (k == 0)k = 9;

j = ((w - k) % 81) / 9 + 1;

i = (w - k - 9 \* (j - 1)) / 81 + 1;

sudoku[i - 1][j - 1] = k;

}

}

printf("数独的解为:\n");

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

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

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

}

printf("\n");

}

getchar(); getchar();

}

}

return OK;}