

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 程序设计综合课程设计**

**专业班级：计算机科学与技术201708班**

**学 号： U201714814**

**姓 名： 赵泽阳**

**指导教师： 卢 萍**

**报告日期： 2019年4月5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

[任 务 书 I](#_Toc7163)

[1 引言 1](#_Toc2251)

[1.1课题背景与意义 1](#_Toc28072)

[1.1.1 课题背景 1](#_Toc27290)

[1.1.2 课题意义 1](#_Toc10452)

[1.2 国内外研究现状 2](#_Toc26595)

[1.3 课程设计的主要研究工作 3](#_Toc12977)

[2 系统需求分析与总体设计 4](#_Toc6847)

[2.1 系统需求分析 4](#_Toc10206)

[2.1.1系统目标 4](#_Toc5273)

[2.1.2系统流程 4](#_Toc9809)

[2.2 系统总体设计 4](#_Toc27867)

[2.2.1 主控，交互与显示模块 4](#_Toc23513)

[2.2.2 CNF解析模块 5](#_Toc26109)

[2.2.3 DPLL模块 5](#_Toc9572)

[2.2.4数独模块 6](#_Toc25641)

[3 系统详细设计 7](#_Toc11683)

[3.1 有关数据结构的定义 7](#_Toc2502)

[3.1.1文字数据结构 7](#_Toc11733)

[3.1.2子句集数据结构 7](#_Toc12723)

[3.1.3数独数据结构 7](#_Toc18860)

[3.1.4 DPLL辅助数据结构 7](#_Toc31732)

[3.1.5各数据在系统中的关联 8](#_Toc1973)

[3.2 主要算法设计 9](#_Toc2328)

[3.2.1主控，交互与显示模块 9](#_Toc12714)

[3.2.2 CNF解析模块 11](#_Toc8627)

[3.2.3 DPLL解析模块 12](#_Toc1166)

[3.2.4 数独模块 15](#_Toc12941)

[4 系统实现与测试 18](#_Toc5774)

[4.1 系统实现 18](#_Toc2945)

[4.2 系统测试 18](#_Toc18676)

[4.2.1系统测试的软硬件环境 18](#_Toc32290)

[4.2.2系统功能的测试 18](#_Toc3766)

[4.2.3系统优化前后的对比 25](#_Toc8851)

[5 总结与展望 27](#_Toc21598)

[5.1 全文总结 27](#_Toc31815)

[5.2 工作展望 27](#_Toc31844)

[6体会 28](#_Toc17781)

[参考文献 29](#_Toc15156)

[附 录 30](#_Toc8078)

# **1 引言**

## **1.1课题背景与意义**

SAT问题的英文名称是Boolean Satisfiability Problem，即布尔可满足判定问题。对于一个命题公式，它由若干命题变量和逻辑符号组成，在把它化为合取范式（即CNF范式）后，我们可以利用合取范式的各种性质对合取范式进行化简，判断是否有一组真值指派使合取范式的真值为真。如果找出了一组真值指派使使合取范式的真值为真，则由于合取范式等价于原来的命题公式，原来的命题公式就获得了求解。

### **1.1.1 课题背景**

SAT问题是计算机科学的重要问题，也是第一个被证明的NP完全问题。从上个世纪60年代开始，很多从事计算机科学研究的学者开始着手研究SAT问题，在这个过程中学者们不断地研究优化求解合取范式的方法和加速程序运行的方法，数十年来已经产生出了各种各样的算法和求解器，至今已经发展出一套较为完备的SAT问题求解理论。SAT问题的可求解规模也从开始的数十个命题变元到如今的数十个命题变元。虽然在现在已经有很成熟的SAT问题求解器供研究人员使用，但作为一个计算机科学专业的学生，深入了解SAT问题的求解方法和过程并加以实现和应用，同时尝试优化SAT问题求解器使其速度尽可能地提高是很重要的。所以本课题的目的在于实现一个SAT问题求解器并尝试对其优化。

### **1.1.2 课题意义**

对于一个SAT问题，如果采用穷举法进行求解，则问题的时间复杂度为O(2n)，可见利用穷举法的时间复杂度为指数级，对时间的消耗非常大。用穷举法解决上万个变元的SAT问题时将生成一个极其庞大的搜索空间，这在现实中是不能在有限时间求解的。所以我们要寻找更加优化的方法。对于一个SAT问题，它可能是可满足的，也可能是不可满足的，尽管如今已经有了很好的求解器，但SAT问题仍然是一个NP完全问题，就算是再优秀的求解器也有解决不了的问题。

研究求解SAT问题在理论和实际中都具有重大意义。在过去几十年中，大量的计算机科学家提出了很多优秀的算法并研发了各种各样的求解器。在这个过程中，很多数学和逻辑学难题有机会得到解决，比如地图着色问题和婚礼问题和自动定理证明等，而这些数学问题的解决又可以应用到现实生活中，比如电路设计等问题，为社会的进步和科技的发展带来动力。

## **1.2 国内外研究现状**

SAT问题作为最古老最有名的NP完全问题，从上世纪六十年代，就有计算机科学家不断的尝试解SAT问题的算法，不少国内外学者在这方面做了研究。

SAT求解算法主要有两个类别，其中第一类是完备算法，另一种是不完备算法。

完备搜索算法是指这个算法采用回溯搜索算法，保证问题最终一定可以得到一个确定的结果，即可满足或不可满足。回溯搜索算法类似于遍历树，对于一个命题变元所构成的树，每次选择一个变元为它赋值，如果命题不出现冲突，则继续赋值，如果出现冲突，则选择另一相反真值赋值，如果都赋值过了，则回到上一层继续进行同样的工作。可见完备算法在最坏的情况下要遍历整个二叉树，对于命题变元很多的情况下，算法的时间复杂度为O(2n)，这显然是非常不乐观的，所以大量学者开始着手于优化算法，其中最著名的是DPLL算法。DPLL算法的名称来源于发明它的四位计算机科学家，他们分别是Davis, Putnam, Logemann和Loveland，他们在1962年提出DPLL算法，DPLL是一个非常高效的完备算法，经过近50年的发展，大量学者在其基础上进行改进，它已经愈发成熟，并被证明是至今为止最有效的求解SAT问题的完备算法。

另一种算法是不完备算法，它采取启发式搜索策略，这样搜索解空间更加快速，但它的缺点是对于某些问题，他可能求不出解，所以对于不可满足问题，它求不出问题的解，即证明不了问题的可满足性。

由于S.A.Cook在1971年证明出了SAT问题是NP完全问题，很多学者放弃了研究，所以在相当久的一段时间内，SAT问题的研究进程变得很缓慢。但在90年代后期，由于一些SAT竞赛，不少计算机科学家又开始关注SAT求解问题，在这之后，SAT求解算法得到了巨大的改进。许多学者在DPLL算法的基础上进行了优化与改进，创造出了许多优秀的SAT求解器，比如MINISAT,SATO,POSIT,CHAFF,GRASP等，这些算法的改进主要包括采用新型的数据结构，对变元选取策略的优化和更高效的优化策略。同时，国内的不少科学家也进行了研究。比如2000年金人超和黄文奇提出的并行Solar算法等[1]。

## **1.3 课程设计的主要研究工作**

本课程设计主要研究工作是了解SAT问题的背景，了解其发展过程。学习DPLL的算法框架，分析求解过程，了解命题简化的基本知识和理论，并根据DPLL算法框架设计出一个SAT求解器。由于最开始设计的SAT求解器在速度或性能上难免存在缺陷，所以本课程设计要在其基础上对于数据结构，算法等方面进行优化以使其变现出更优秀的性能并前后进行对比，分析优化算法的途径，并对优化过程进行总结。最后实际应用到数独问题中，利用程序生成一个随机数独，将其转化为SAT问题，并使用SAT求解器求解数独问题，完善数独。

# **2 系统需求分析与总体设计**

## **2.1 系统需求分析**

### 2.1.1系统目标

本系统的目标是根据DPLL算法框架，选择合理的数据结构，使用合适的快速的求解算法，研发设计出一个可用的小型SAT求解器。并尝试优化SAT求解器，使其表现出更好的性能，解决更多的问题，将优化前和优化后进行对比，分析差异。最终利用程序生成随机数独，再将数独终盘利用算法转化为SAT问题，利用之前的SAT求解器求解数独。

### 2.1.2系统流程

系统的流程主要分为以下几步：

1. 按照格式读取CNF文件，并将读取到的内容存储到合适的数据结构中。
2. 根据DPLL框架写出求解SAT的算法，对（1）中读取的SAT问题进行求解，并利用计时函数得出求解SAT问题所用的时间。
3. 使用挖空法随机生成一个9\*9的数独终盘，并提供一个界面，用户可以手动求解数独。
4. 将数独终盘转化为SAT问题，写入CNF文件中。
5. 使用SAT求解器求解数独的CNF文件，并完善数独，显示求解结果。

## **2.2 系统总体设计**

系统的总体设计主要有以下几个方面：

### 2.2.1 主控，交互与显示模块

本模块用于与用户进行交互，接受用户的输入并进行输出，是系统的主控程序。本模块主要完成的功能有：

1. 显示SAT求解菜单，这个菜单的功能包括接受用户输入的文件名，如果文件存在则加载CNF文件，显示当前加载的CNF文件的变元数和句子数目，如果文件不存在或加载不成功显示加载不成功。求解CNF公式，输出结果并将结果按照一定的格式写入，进入数独菜单和退出功能。
2. 显示数独菜单，数独菜单包括接收用户选择的数独的难度等级，生成数独并可以使人玩数独，使用SAT求解器求解数独和回退到SAT求解菜单。

### 2.2.2 CNF解析模块

该模块的主要功能有：

（1）读取cnf算例文件。对于每一个cnf文件，它含有以‘c’开头的若干注释说明行和以‘p’开头的行说明公式的总体信息，之后每行对应一个子句，0为结束标记。46表示第46号变元，且为正文字；-46则是对应的负文字，文字之间以空格分隔。

（2）将读取到的cnf文件进行解析，给予一定的数据结构，建立公式的内部表示，同时要实现对解析正确性的验证功能，即遍历内部结构逐行输出显示各子句内容以方便人工判断解析功能的正确性。

### 2.2.3 DPLL模块

该模块是整个系统的核心，本模块基于DPLL算法框架，实现对SAT算例的求解。该模块的主要组成有：

1. 创建，初始化空子句和空文字。
2. 根据CNF解析模块生成的数据结构，生成附加的数据结构以方便SAT问题的求解。
3. 在非递归求解SAT问题过程中对栈的操作函数，包括初始化栈，出栈，入栈操作。
4. 利用单子句传播规则化简子句集。
5. 根据一定的策略选择变量值进行单子句传播。
6. 判断当前子句集状态，包括不确定，可满足，不可满足。
7. 回溯过程，相当于单子句传播的逆过程，它将单子句传播过程中改变的值全部还原。
8. DPLL函数，集合上述函数，基于DPLL框架，进行非递归求解SAT问题。
9. 将生成的解按照一定的规则写入res文件。
10. 销毁CNF数据结构和附加数据结构。

### 2.2.4数独模块

该模块包括以下几个功能：

1. 生成数独。本功能的过程为随机生成一个9\*9的终盘，再根据难度随机挖空，生成标准数独。
2. 显示数独。使用不同的颜色区分需填空的和未填空的，将随机生成的数独打印出来。
3. 人玩数独。该模块接受用户的输入并进行判断，如果合适就将用户填入的值放入数独并检查当前结果是否为解，如果不合适就提醒用户重新输入。
4. 将数独归约为SAT问题。根据逻辑写命题公式，将公式写入cnf文件中。
5. 读取SAT求解器求解数独问题所生成的cnf文件，将cnf文件的解写入数独，完成求解。

# **3 系统详细设计**

## **3.1 有关数据结构的定义**

本系统主要处理的数据有：

### 3.1.1文字数据结构

每个文字（Literal）结点由一个结构组成，每个结构包括指向文字结点的指针和文字的索引。

### 3.1.2子句集数据结构

子句（Clause）由一个结构组成，每个结构包括指向文字的指针head和表示该子句真值的Cvalue，以及记录该子句文字数目的Literal\_number。子句集（ClauseSet）是一个结构数组，由子句结构指针ClauseSet指向子句构成子句集。

### 3.1.3数独数据结构

1. soduko数组：9\*9的int型数组，程序中生成数独，对数独挖空，将数独转化为SAT问题，对数独求解都是在在这个数组上进行操作。
2. knockout\_flag：9\*9的int型数组，记录挖空后数组中为0的格子位置。
3. Sodukoanswer：长度为82的int型数组，记录数独文件解。
4. Checksodukolist：9\*9的int型数组，用来标志此位置是否挖过空,1为挖过。

### 3.1.4 DPLL辅助数据结构

1. 文字检查表：该表为一个结构数组，文字检查表指针LookupValue指向这个数组。数组的每个结构元素包括记录文字值为正的数目的positive和记录文字值为负的数目的negative。
2. 文字链接表：该表为一个结构数组，LiteralSet指针指向这个结构数组。对于数组中的每个结构，它包括int型的posOcccur指针，int型negOccur指针和int型chooseflag，它们分别标记正文字出现的子句索引，负文字出现的子句索引和该层回溯次数（初始为0）。
3. 文字赋值数组：int型指针valuation指向这个数组，这个数组用于记录变量被赋的值，初值初始化为-1，表示不确定。
4. 文字栈：文子栈为一个结构，其中包括int型栈顶指针top和int型栈底指针base。
5. 变元数和子句数变量：int型变量variableNumber和clauseNumber用于记录初始时的变元数和子句数。

### 3.1.5各数据在系统中的关联

1. DPLL中各数据的关联：

在cnf解析模块解析cnf文件的同时，cnf文件同时建立起问题的数据结构。数据结构的主体是一个邻接表和一个邻接链表。邻接表是指由文字链表和子句头结点数组构成子句集结构，邻接链表结构是指每个变量的正负文字所出现的子句指针构成的链表。在DPLL过程中利用邻接表来判断句子状态从而判断子句集状态。 利用变量variableNumber和clauseNumber记录初始时子句集变元数和子句数用于DPLL过程中各函数对子句集和子句进行遍历。利用文字栈记录在DPLL过程中所选择的和舍弃的文字值，并在DPLL过程中结合检查表和文字链接表进行回溯，还用于判断程序的终止条件。文字检查表用于单子句传播和回溯。文字赋值数组用于记录每次单子句传播的赋值和回溯进行还原，如果SAT问题是可满足的，则将其记录的结果写入文件中。

1. 数独中各数据的关联：

系统先随机生成一个9\*9的终盘，将其写入soduko数组中，然后进行挖空，使用knockout\_flag数组记录挖空后数组中为0的格子，在随机挖空时如果这个格子已经挖空就可以被跳过。checksodukolist用于记录此位置是否挖过空，在用户交互界面系统根据检查表判定该位置是否可以填入数字。

## **3.2 主要算法设计**

### 3.2.1主控，交互与显示模块

本模块系统的主体，用于与用户进行交互，进行显示。

本模块的流程为：先打印CNF菜单开头的提示信息和可选择的输入选项（0~5）。系统提示用户输入要选择的操作，接受用户的输入后。

如果输入数字为1，进入系统的第一个功能：加载CNF文件。如果在加载CNF文件是发现子句集ClauseSet不为空，即之前加载过CNF文件，则先销毁原来的CNF信息防止加载和解析CNF文件时出现错误。系统提示用户输入要加载的CNF文件的文件名，然后调用CNF解析模块的readfile函数将CNF文件中的信息写入相应的数据结构中，如果成功，则提示文件打开成功，否则提示文件打开失败，在之后系统在菜单开头会将子句数量和变元数量刷新。

如果输入的数字为2，先检查子句集是否存在，如果不存在则提示CNF不存在，如果存在，则提示正在使用求解器求解，然后系统调用计时函数对DPLL过程进行计时。DPLL过程结束后，对结果进行判断，如果结果为SATISFIABLE，则输出SATISFIABLE，如果结果为UNSATISFIABLE，则输出UNSATISFIABLE。然后系统输出DPLL所用的时间，单位是毫秒。最后，系统提示用户输入结果的保存路径，接收到保存路径后，系统调用DPLL模块的writeSolution功能将结果写入目标路径中，如果写入成功，则提示文件保存成功，如果写入失败，则提示文件写入失败。

如果输入的数字为3，先检查子句集是否存在，如果不存在则提示CNF不存在，如果存在，提示正在打印CNF，并调用本模块的printCnf函数打印经过功能1所生成的子句集的结构，方便人工比较生成的数据结构是否正确。printCnf函数的流程为顺序遍历ClauseSet，逐行输出每个子句节点所连接的文字链表的每个文字的值，每个句子以0结尾。

如果输入的数字为4，则系统判断求解结果，如果结果为SATISFIABLE，则调用本模块的printRes函数打印出求解结果，如果结果为UNSATISFIABLE，则输出UNSATISFIABLE。printRes函数为遍历赋值表LiteralSet，对于赋值为0的命题变量，函数输出负的文字值，对于赋值为1的命题变量，函数输出正的文字值。

如果输入数字为5，则进入数独菜单。

如果输入数字为0，则退出程序。

CNF菜单的流程图如图3-1所示。

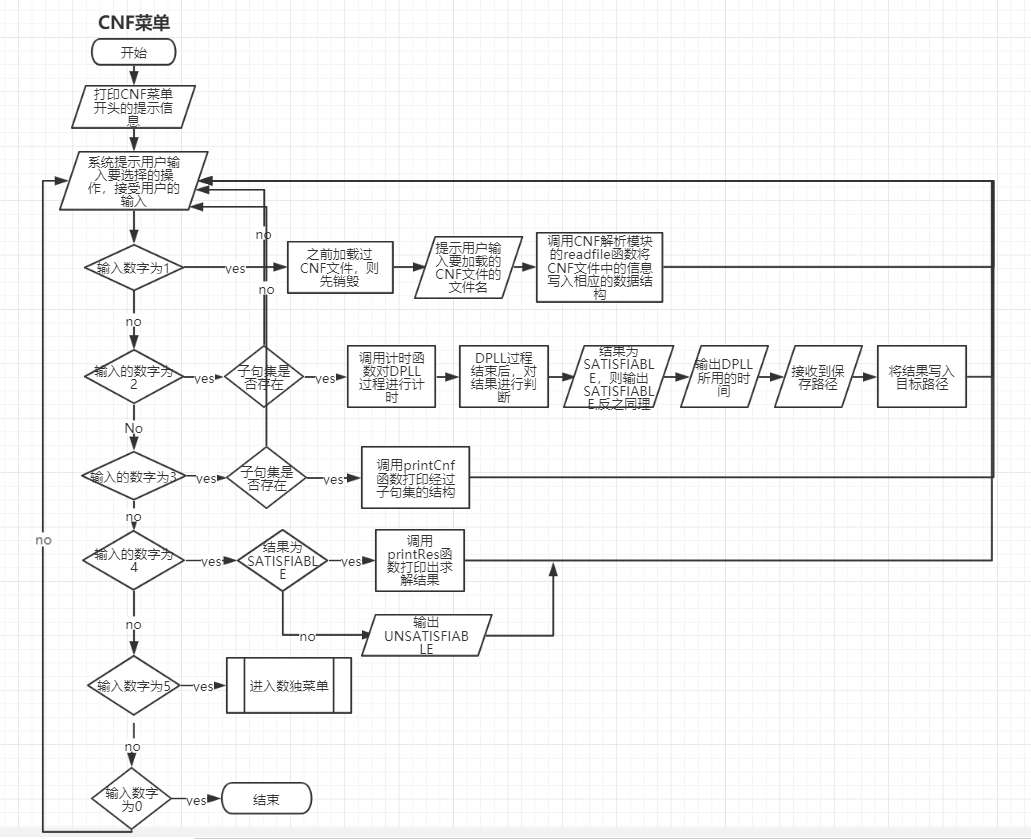


图3-1 CNF菜单的流程图

进入数独菜单后，系统提示用户输入数独的难度（1~5），数字越大难度越大。接受用户输入后，系统判断输入数字的合法性，若不合法提示重新输入，若合法，则调用数独模块的generatefinalsoduko函数生成随机挖空数独并提示用户生成数独成功。然后打印数独菜单开头的提示信息和可选择的输入选项（0~3）。系统提示用户输入要选择的操作，接受用户的输入后。

如果输入为1，则调用数独模块的playsoduko函数供人来玩数独。

如果输入为2，则如果之前的CNF的数据结构存在，系统先清空原来的数据结构，如果不存在，系统调用数独模块的sodukoToCnf函数将挖空的数独归约为SAT问题并写入sudoku.cnf文件中，然后系统调用DPLL模块的readfile函数建立CNF公式的数据结构，调用计时函数并进行DPLL过程求解SAT问题，然后后系统调用DPLL模块的writeSolution函数将解写入sudoku.res文件中，最后系统调用数独模块的playwithcnf函数和printsoduko将解写入挖空的数独中并显示。

如果输入为0，则回退到CNF菜单。

### 3.2.2 CNF解析模块

CNF解析模块只含有一个函数readfile,它对cnf文件进行读取的解析，并按照一定的结构建立公式的内部表示，同时实现对解析正确性的验证功能。

readfile函数接收文件名后，以读模式读取文件，对于文件中以字符‘c’开头的行，这些是代码行，程序将跳过，对于文件中以字符‘p’开头的行，这一行包含子句集的基本信息，其中第一个为命题变元数目，第二个为子句数目，程序将其分别写入variableNumber和clauseNumber中，然后对文字集LiteralSet，查找表LookupValue，子句集ClauseSet，文子栈分配空间并初始化为初始值。从下一行开始，程序为每一行子句创建链表，程序依次读取字符将其转化为数字，每读取一个数字就调用DPLL模块的createLiteral函数创建一个文字结点，并将文字值写入文字结点的index中，如果文字不为0，对于每个出现的文字，如果值为正，则将文字检查表LookupValue对应变量的positive自增，否则将negative自增。然后，用尾插法依次建立每个子句链表，读到0时这一行结束，即这一子句读取结束，开始写入下一个子句，直到文件读取结束。

CNF解析模块流程图如图3-2所示。

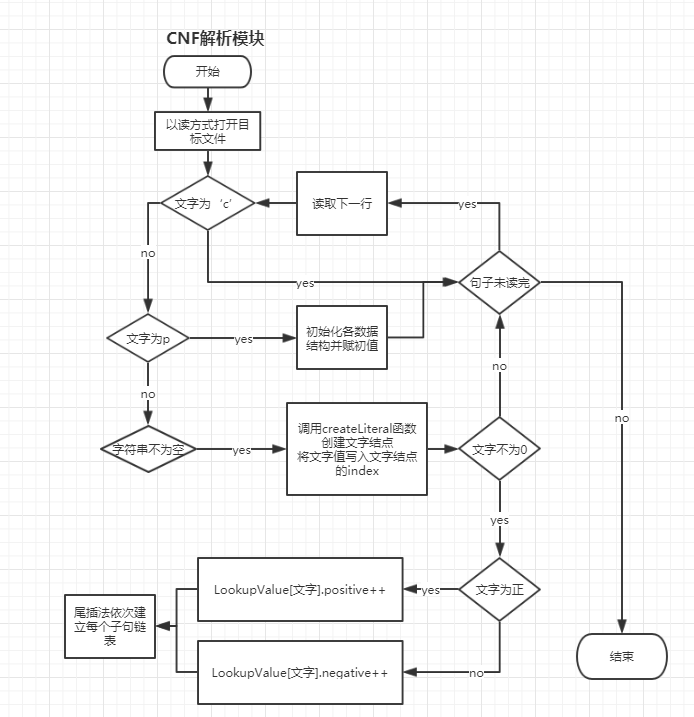


图3-2 CNF解析模块流程图

### 3.2.3 DPLL解析模块

本模块是系统的核心模块，DPLL过程采用非递归进行，设计的函数较多，现说明如下：

1. 创建，初始化空子句和空文字函数createClause和createLiteral。这两个函数分别为文字结点和句子结点分配空间并初始化，这两个函数分别返回文字结点指针和子句节点指针。
2. 初始化文字邻接表函数InitLiteralSet。该函数首先根据命题变量数目进行循环，在循环体内为每个命题变元的posOcccur和negOccur指针分配空间，空间大小为该文字正文字出现的数目和负文字出现的数目。然后将chooseflag设置为0。然后该函数遍历每一个文字创建文字邻接表，创建的过程为：另一个Literal型指针tpNode指向一个子句的头结点，然后遍历这个子句的文字链表。如果文字为正，则对应的文字邻接表的posOcccur的单元存该文字出现的子句的序号，同样的，如果文字为负，则对应的文字邻接表的negOccur的单元存该文字出现的子句的序号。遍历完该子句后再依次遍历下一子句直到遍历结束。这样就完成了文字邻接表的创建。
3. 栈操作函数InitStack，Push和Pop函数。它们完成对文字栈的初始化，出栈和入栈操作。在这里不再赘述。
4. 移除子句函数removeClause，移除子句集函数removeClauseSet。它们用于销毁CNF公式的数据结构，都是通过遍历链表的每个节点并释放空间进行销毁，在这里不再赘述。
5. 选择变元函数chooseLiteral，该函数通过一定的变元选取策略进行变元选择。首先他遍历整个子句集表头结点，如果某个句子的文字数目为1，表示该子句只剩一个文字未满足，我们取该文字的值，则该子句就得到了满足，子句集就得到了简化，利用单子句传播规则，我们可以快速缩小问题的规模，找到满足的解。本函数如果遍历完子句集表头节点没有发现单子句，则选择含有双文字的子句，从中选一个文字使句子为真再进行化简。如果搜索完没有含双文字的子句，则根据赋值表将一个还未赋值的文字选一个值进行化简。
6. 单子句传播函数Bcp。这是DPLL的核心函数之一，体现了DPLL的核心思想。Bcp函数根据变元选择函数所选择的文字值进行单子句传播。该函数的过程为：如果传入文字值为负，则赋值表中该变量值为0，否则为1。如果选择变元函数选的是正文字，则根据文字邻接表，找到该变量正文字所出现的所有子句并使这些子句的真值为真，然后找到这个变量所有负文字出现的所有子句，并令这些字句的文字数减1，如果此时句子的文字数目变为0，则说明该子句不满足，出现冲突，所以子句集也是冲突的，需要进行回溯。如果选择变元函数选的是负文字则相反。
7. 回溯函数backtrack。该函数相当于是Bcp函数的逆过程。对于产生冲突的文字值，我们要进行还原。该函数的过程为：对于传入的文字值，先将其赋值表的值改为-1，即未确定。然后将所有含该变量正文字和负文字的子句的真值和文字数目还原到赋该文字值之前的状态。具体方法是：如果传入的是正文字，则查看文字邻接表，查看该变元所有负文字出现的子句，将相应子句的文字数目加1，然后找到该变元所有正文字出现的子句，遍历每一个字句的文字，查看除该文字外其他变量的赋值情况，如果其他变元的赋值使该句子可满足，则该句子的值仍为满足，否则将句子的值修改为未满足。对于传入的是负文字的情况同理可以推出。
8. 检查子句集状态函数checkSolution，该函数判断当前赋值下，子句集的状态。该函数的过程为：遍历子句集ClauseSet的每一个头结点，获取它的句子的值，如果任何一个句子是冲突的，则子句集是冲突的，如果有任何一个句子的值为未确定，则子句集的状态为未确定，否则子句集的状态为满足。
9. DPLL框架函数dpll2，该函数是DPLL模块的主体函数，它将所有函数集合在一起，构成一个SAT求解模型。该函数的流程为：首先使用变元选择函数选择一个变元并将其入栈，同时将其在LiteralSet对应的变元的chooseflag记为1，表示选择了一次值，然后进入循环，该循环的过程为：内嵌一个循环，该层循环一直进行直到得出解。内部循环工作流程为：1.将刚才入栈的变元进行单子句传播，然后调用checkSolution函数用Set\_status记录子句集当前状态。2.如果Set\_status为CONFLICT则进入循环:将冲突变元出栈，进行回溯，如果该变元对应的LiteralSet的chooseflag为1，表示该变元选了一次，这时我们选择反变元入栈并令其chooseflag为2，跳出循环回到1处继续进行，否则该变元对应的LiteralSet的chooseflag为2，表示该变元选了两次则进入循环：这是表明该变元的上面数层决策错误，所以如果栈非空，将该变元对应的LiteralSet的chooseflag设置为0，表示没选过，回到2处继续执行。否则如果栈空，则SAT问题的解为UNSATISFIABLE，函数结束。如果Set\_status的值为SATISFIABLE，则返回SATISFIABLE，运行结束。如果Set\_status的值为UNCERTAIN，那么再调用一次选择变元函数，将选择的变元的chooseflag设置为1，并将其入栈，进入1处继续循环。
10. 将结果写入文件函数writeSolution，该函数根据DPLL的求解结果将结果写入RES文件，它的参数包括文件名，结果，运行时间。首先函数根据文件名以写方式打开文件，如果结果为SATISFIABLE，则写入s 1，并将赋值数组的文字值写入文件，如果结果为UNSATISFIABLE，则写入s 0，如果结果为UNCERTAIN，则写入s -1。最后，将运行时间写入文件。

DPLL模块流程图如图3-3所示。

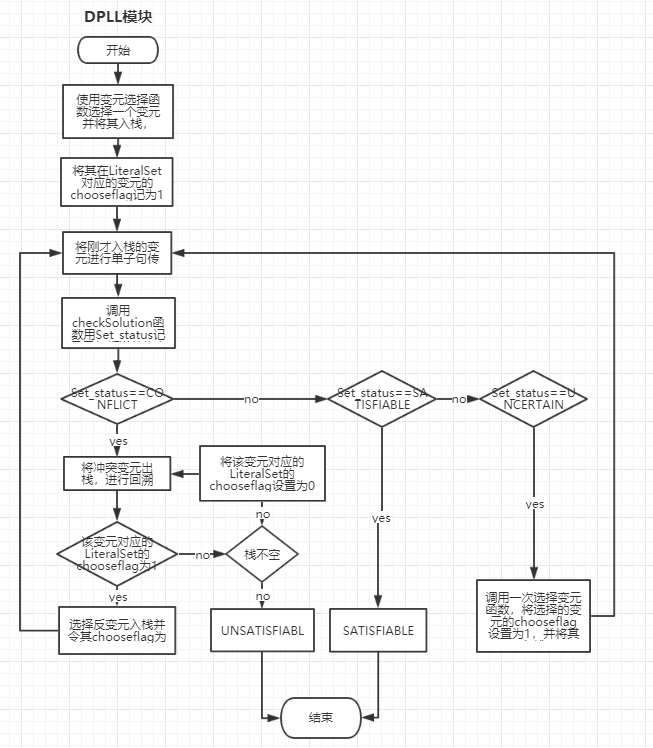


图3-3 DPLL模块流程图

### 3.2.4 数独模块

数独模块是DPLL模块的实际应用，在这个模块中随机生成数独，并转化为SAT问题，利用DPLL模块进行求解，最后再将求解结果写入数独。数独模块还可与人交互，供人玩游戏。具体过程说明如下：

1. 生成未挖空数独：该功能的实现函数包括生成随机数函数generateRandNum，生成数独第一行函数generateFirstLine，生成数独剩余8行函数generatesoduko，判断数独是否已经全部填满函数issodukoFull，检查检查表checklist函数checkChecklist和检查数独是否符合条件函数check。具体流程为：先调用generateFirstLine函数，利用generateRandNum函数随机生成第一行，同时在写第一行每个格子时每写一个数字，检查表checklist相应位置进行标记以保证第一行的9个格子包含9个不同的数字。生成第一行后，调用generatesoduko函数判断数独是否已全部填满，如果为填满，从第二行开始，每填入数字，使用check函数检查同行同列同宫是否重复，若不重复，则填下一个，若重复，则换一个数继续填，直到数独填满。
2. 对数独进行挖空函数knockout\_soduko:该函数调用随机数函数生成随机行号和随机列号，然后根据传入的难度等级决定挖空数量，以该数量为循环条件对数独进行挖空。对某个空挖空后，对应checksodukolist数组的相应位置做标记以说明位置已挖空，这样循环时如果随机挖到同样的空，则跳过重新挖，最后生成挖空数独。
3. 将数独归约为SAT问题函数sodukoToCnf：该函数将数独转化为SAT问题，并将命题公式写入cnf文件中，具体方法为：先计算数独的未挖空的格子的数量notzeronum，根据notzeronum可得命题公式变量数和子句数。经过推导，子句数=11988+notzeronum。推导过程为:每个格子对应9个命题变元，这9个命题变元对应这个格子取1-9的值，所以一共有9\*9\*9=729个变元。1.对于每一个格子，它至少包含9个数字中的一个，这样可以写9\*9=81个子句，对于每一个格子，它不能同时为两个数，这样有9\*9\*C(9,2)=2916个子句。2.对于每一行，它要包含1-9所有数字，所以有9\*9=81个子句，对于每一行，不能包含相同的数字，所以有9\*9\*C(9,2)=2916个子句。3.对于每一列，和每一行同理，有81+2916个子句。对于每个宫，它要包含1-9所有数字，所以有9\*9=81句，对于每个宫，宫里的每两个格子不能有相同数字，有9\*9\*C(9,2)=2916个子句。4.对于未挖空的格子，它的值是确定的，可以写出单子句，共有notzeronum个子句。综上所述，共有11988+notzeronum个子句，sodukoToCnf函数的流程就是把这些字句写入cnf文件。
4. 设置颜色与打印数独函数：setcolor和printsoduko函数用于实现数独的彩色输出。函数根据挖空时的标记数组对于数独中本来就存在的数，将其打印为白色，对于数独中挖空的数字，将其打印为红色，对于数独中挖空后被填入的数字，打印为绿色。
5. 玩数独：该功能包括系统用res文件玩数独函数playwithcnf和人玩数独函数playsoduko。playwithcnf函数将解写入数独并打印到屏幕，不再赘述。对于playsoduko函数，这个函数与用户进行交互，先提示用户输入行号列号和要填入的数字，然后函数对输入进行判定，使用knockout\_flag检查该空是否为挖空，若挖空则判断该空填入的数字是否符合数独规则，若符合，则写入并检查数独是否已经填满，如果已经填满则提示游戏成功，否则提示成功填入，询问下一个要填入的位置及数字。如果该位置没有挖空，则提示输入非法，请重新输入。如果用户要请求退出，则游戏结束。
6. 读取数独结果文件函数readsodukofile：本函数将DPLL生成的数独解文件结果读取，将解写入数独中，完成解数独。实现过程为，对于数独，它一共含有9\*9\*9=729个命题变元，程序以九个为一组，每九个变元值确定每个位置应该填的数，将数读取出来后，函数将结果写入sodukoanswer数组中。

数独模块流程图如图3-4所示。

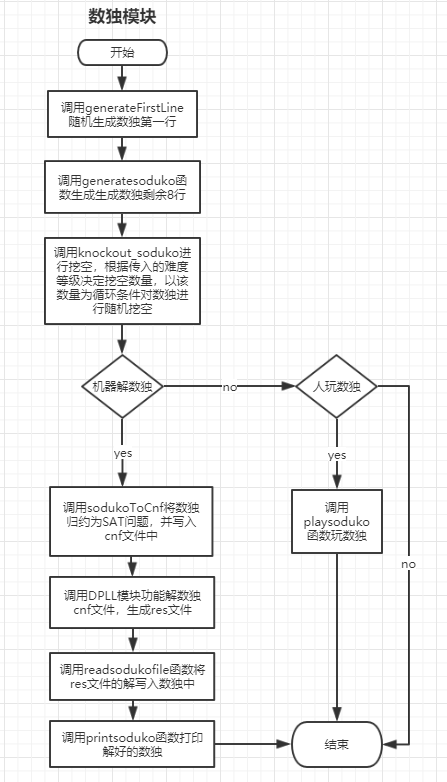


图3-4 数独模块流程图

# **4 系统实现与测试**

## **4.1 系统实现**

本系统通过四个模块来实现。这四个模块分别是主控与交互模块，CNF解析模块，DPLL模块和数独模块。主控系统负责显示信息，提示用户输入，接收用户输入和调用CNF解析模块和数独模块。CNF解析模块用于生成DPLL模块处理的数据结构，数独模块使用DPLL模块解数独。

本系统的实现目标和流程为系统能够接收用户的输入，调用系统功能解决SAT问题，并且读取文件后可显示CNF公式的结构方便用户进行对比。在DPLL过程解SAT问题时，使用计时函数对DPLL过程进行计时。在数独模块中可以根据用户的需求生成难度分级的数独并且用户可以对数独进行操作，另外数独模块还可将数独问题转化为SAT问题，调用DPLL模块进行求解，将结果填入空中并显示，供用户参考。

程序详见附录。

## **4.2 系统测试**

### 4.2.1系统测试的软硬件环境

（1）系统软件环境：本系统通过CodeBlocks进行编译链接,生成可执行文件。

（2）系统硬件环境：处理器为i5-7300HQ,内存8G。

### 4.2.2系统功能的测试

系统的测试主要包括检验系统各模块是否按照预期目标运行，以及检验运行的正确性和可靠性，本系统的测试将就主控模块，DPLL模块，数独模块进行测试。

1. 主控，用户交互模块的测试

模块功能和设计目标：本模块主要是调用其他模块和与用户交互进行输入输出。包括加载CNF文件，调用DPLL模块求解CNF，打印CNF公式,输出结果，调用数独菜单和退出。

测试大纲：本模块的测试主要是检验主控模块是否能正常输出和接收信息以及调用CNF解析模块。测试流程为先输入1调用加载CNF文件功能，然后用户输入文件名载入文件，之后菜单界面应显示当前加载文件的变元数和子句数。然后输入3，使用打印CNF功能，用于人工对比载入文件后建立的CNF公式内部结构表示是否正确。

测试结果：输入1后输入文件名后结果如图4-1所示。

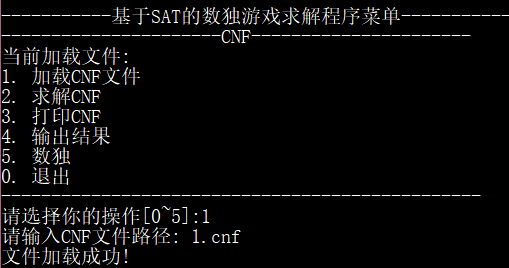


图4-1 测试结果图

系统提示文件加载成功，并在菜单头提示变元数和子句数，如图4-2所示。

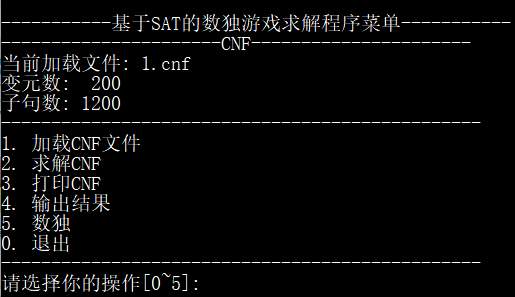


图4-2 测试结果图

输入3后，系统输出CNF数据结构如图4-3所示。

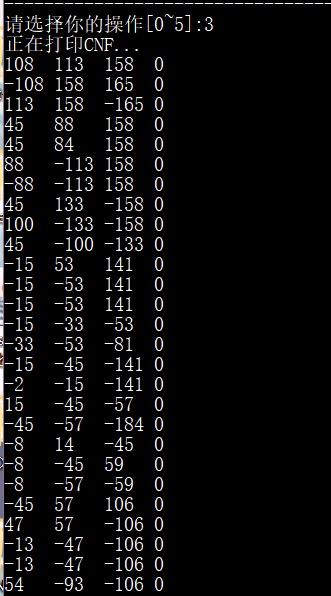


图4-3 测试结果图

查看源CNF文件如图4-4所示。

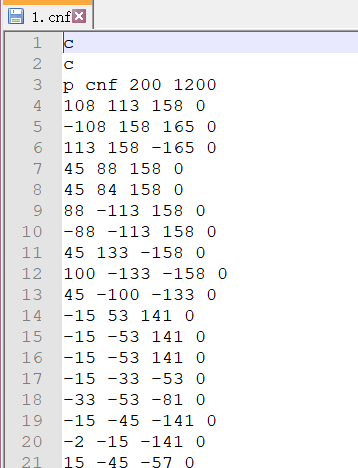


图4-4 cnf文件截图

结果分析：经过对比发现主控模块功能运行正确，能够正常调用CNF解析模块建立正确的物理结构内部表示，模块的设计达到了预期的目标。

1. DPLL过程的测试

模块功能和设计目标：本模块主要是求解SAT问题，然后将结果写入RES文件中。

测试大纲：本模块的测试主要是检验DPLL模块是否能够解决SAT问题并记录DPLL过程的时间以及将解按照正确的格式写入RES文件中。测试流程为在CNF菜单输入2调用求解CNF功能，观察求解结果。将结果写入RES文件中后，打开RES文件，查看结果是否正确。

测试结果：输入2求解SAT问题，运行结果如图4-5所示。

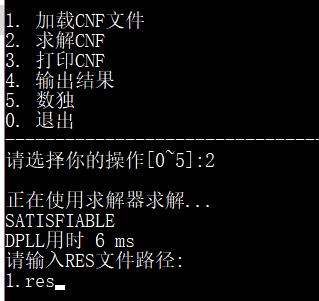


图4-5 测试结果图

输入RES保存路径后，打开相应的RES文件，结果如图4-6所示。

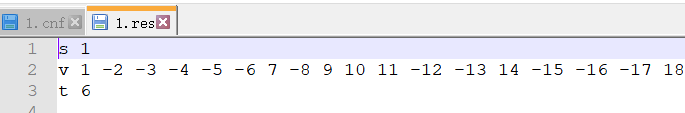


图4-6 RES文件截图

结果分析：经过对比发现DPLL模块功能可以解决SAT问题并进行计时，能够将结果按照规定格式写入RES文件中，模块的设计达到了预期的目标。

1. 数独模块的测试

模块功能和设计目标：本模块用于生成不同难度具有可玩性的数独，使用SAT求解器求解数独和与人交互让人玩数独。

测试大纲：本模块的测试主要是检验数独模块是否能够生成正确的数独并用不同的颜色显示以方便人玩数独，还要检验在与人交互过程中数独模块是否能够识别非法的输入以及将用户合法的输入复制到数独中，最后检验数独模块能否将数独正确地转化为SAT问题写入CNF文件中并调用DPLL模块进行求解，然后将结果写入数独展示给用户。测试流程为先输入5进入数独模块，然后选择难度生成数独。接着输入1玩数独，先在一个位置输入一个不合理的数，观察程序运行结果，再输入一个合理的数，观察程序运行结果。最后输入2，调用SAT求解器求解数独问题并显示出求解结果。

测试结果：

输入5调用数独功能并选择难度后，输入1玩数独，并输入一组不合理的值，界面如图4-7所示。

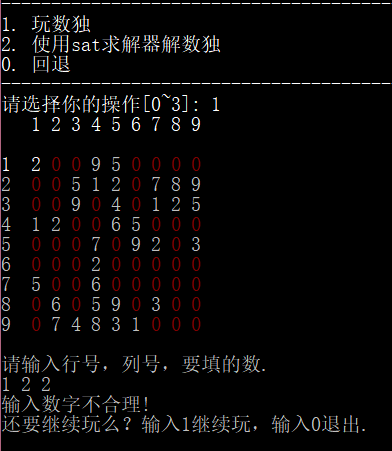


图4-7 测试结果图

输入一组合理的值，结果如图4-8和图4-9所示。

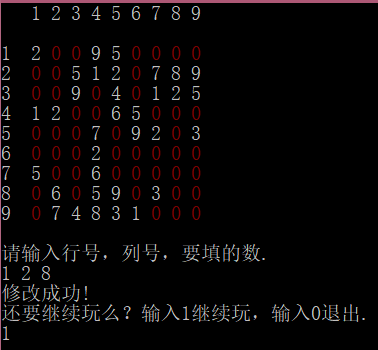


图4-8 测试结果图

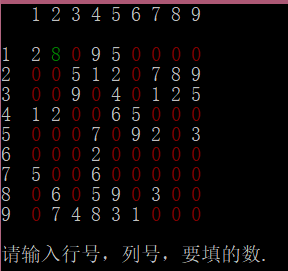


图4-9 测试结果图

输入2后结果如图4-10所示。

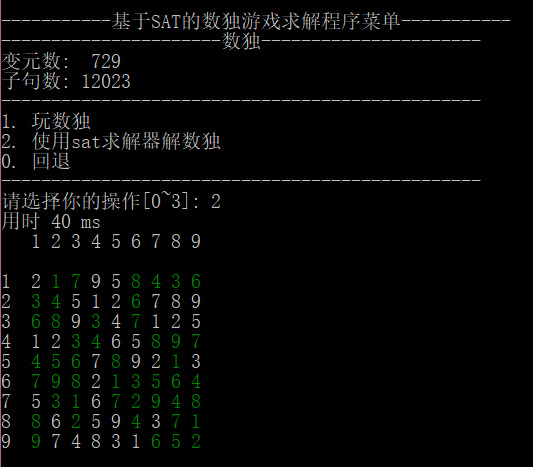


图4-10 测试结果图

打开在解决数独时生成的CNF文件如图4-11所示。

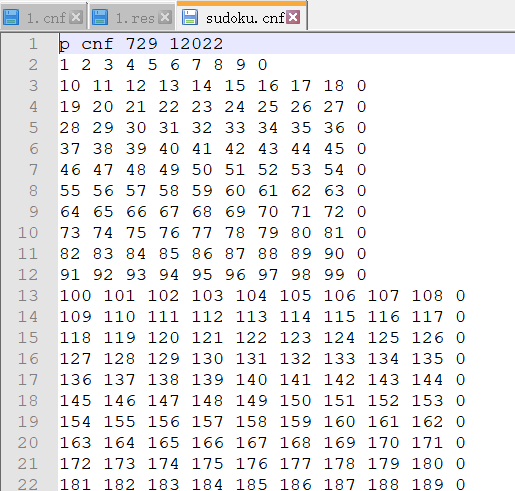


图4-11 CNF文件截图

结果分析：调用数独功能后，可以选择难度生成数独，生成的数独挖空的为红色。在数独界面玩数独如果输入的值不合法会提示错误，如果输入合法会将原来的空修改并用绿色打印。调用SAT求解器求解数独后发现结果正确，观察CNF文件发现CNF公式正确。模块的设计达到了预期的目标。

### 4.2.3系统优化前后的对比

不少于30个SAT算例，其中可满足的算例不少于25个，不满足的算例不少于5个，大中小算例各占三分之一。鉴于大家实现的可能只是初级求解器，对算例规模的要求为：小型算例变元数为100个左右；中型算例变元数介于200-500个； 大型算例变元数600个以上。本设计提供部分cnf算例集,同学们可寻找与选择、扩充测试算例。在设计报告的测试分析部分列表给出每个测试算例下列信息：算例名、算例变元数、子句数与变元数比值、满足还是不满足或不确定、DPLL求解时间(t与to)以及优化率等信息。

1. 优化方面：本系统DPLL模块在数据结构上进行了优化，子句集采用邻接表，并添加一个文字链接表方便单子句传播过程和回溯过程中文字的遍历与寻找，DPLL过程采用非递归方式，提高了程序运行的速度。在变元选取函数作出改进，再找完单子句后，寻找文字数少的子句选取变元进行赋值，大大减少了某些问题的递归深度，加快了程序的运行。
2. 优化前后对比

测试文件包括30个CNF文件，问题规模分别为小型算例11个，中型算例13个，大型算例6个。其中可满足算例25个，不可满足算例5个。测试过程分别记录了每个算例的文件名，变元数，子句数，子句数与变元数比值，问题规模，问题满足性，优化前用时，优化后用时和优化率，测试结果如表4-1所示。

表4-1系统优化前后结果测试记录表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 算例名 | 变元数 | 子句数 | 子句数与变元数比值 | 问题规模 | 问题满足性 | 优化前用时t0/ms | 优化后用时t/ms | 优化率  (t0-t)/t0\*100% |
| 1.cnf | 200 | 1200 | 6 | 中 | 可满足 | 3452 | 6 | 99.8% |
| 2.cnf | 200 | 220 | 1.1 | 中 | 可满足 | 1 | 1 | 0% |
| 3.cnf | 301 | 2780 | 9.2 | 中 | 可满足 | 377 | 18 | 95.2% |
| 4.cnf | 512 | 9685 | 18.9 | 大 | 不可满足 | 1940 | 3489 | -79.8% |
| 5.cnf | 20 | 1532 | 76.6 | 小 | 可满足 | 31 | 103 | -232.3% |
| 6.cnf | 301 | 2780 | 9.2 | 中 | 可满足 | 385 | 19 | 95.1% |
| 7.cnf | 231 | 1166 | 5.0 | 中 | 可满足 | ---- | 2 | 100% |
| 8.cnf | 1584 | 16587 | 10.5 | 大 | 可满足 | ---- | 440 | 100% |
| 9.cnf | 60 | 936 | 15.6 | 小 | 不可满足 | 440527 | 40068 | 90.9% |
| 10.cnf | 200 | 210 | 1.1 | 中 | 可满足 | 1 | 1 | 0% |
| 11.cnf | 459 | 4675 | 10.2 | 中 | 可满足 | 1092 | 1116 | -0.02% |
| 12.cnf | 100 | 403 | 4.0 | 小 | 可满足 | 116 | 90 | 22.4% |
| 13.cnf | 100 | 403 | 4.0 | 小 | 可满足 | 3446 | 26 | 99.2% |
| 14.cnf | 512 | 9685 | 18.9 | 大 | 可满足 | 1895 | 3568 | -88.3% |
| 15.cnf | 729 | 22060 | 30.3 | 大 | 可满足 | 186 | 62 | 66.7% |
| 16.cnf | 500 | 3100 | 6.2 | 大 | 可满足 | 8636 | 182 | 97.9 |
| 17.cnf | 500 | 3100 | 6.2 | 大 | 可满足 | 7 | 28215 | -402971.4% |
| 18.cnf | 100 | 160 | 1.6 | 小 | 可满足 | 1 | 1 | 0% |
| 19.cnf | 100 | 400 | 4.0 | 小 | 可满足 | 18 | 1 | 94.4% |
| 20.cnf | 100 | 425 | 4.3 | 小 | 可满足 | 46 | 4 | 91.3% |
| 21.cnf | 100 | 200 | 2.0 | 小 | 不可满足 | ---- | 214 | 100% |
| 22.cnf | 50 | 100 | 2.0 | 小 | 不可满足 | 576 | 10 | 98.3% |
| 23.cnf | 60 | 936 | 15.6 | 小 | 不可满足 | ---- | 40876 | 100% |
| 24.cnf | 80 | 370 | 4.6 | 小 | 不可满足 | 12109 | 6200 | 48.8% |
| 25.cnf | 181 | 3151 | 17.4 | 中 | 可满足 | 3 | 2564 | -8536.7% |
| 26.cnf | 303 | 2851 | 9.4 | 中 | 可满足 | 24 | 77 | -220.8% |
| 27.cnf | 224 | 1762 | 7.9 | 中 | 可满足 | 15 | 2 | 86.7% |
| 28.cnf | 308 | 2911 | 9.5 | 中 | 可满足 | 7 | 404 | -5671.4% |
| 29.cnf | 301 | 2810 | 9.3 | 中 | 可满足 | 31 | 55 | -77.4% |
| 30.cnf | 297 | 2721 | 9.2 | 中 | 可满足 | 39 | 12 | 69.2% |

# **5 总结与展望**

## **5.1 全文总结**

本次课程设计完成了以下工作：

1. 了解了SAT问题的发展过程。研究了SAT问题的构成和求解方法。学习了解决SAT问题的著名算法DPLL算法以及它的原理。
2. 实现DPLL过程，选择合理的结构和函数，构成SAT求解器。
3. 实现数独游戏并使用SAT求解器解决数独。
4. 在数据结构，变元选取，菜单界面等方面进行优化。

## **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）优化单子句传播的过程，减少遍历的过程。

（2）尝试优化变元选取策略，比如采用冲突子句学习等方法。

（3）研究SAT问题在现实生活中的应用，尝试将现实问题转化为SAT问题。

# 6体会

通过这次的课程设计，我学到了很多知识。这是大学中第一次写正式的课程设计报告，在写报告中，我遇到了很多问题，也在克服这些问题的过程中学到了很多。

这次课程设计中，我查阅了很多论文，了解这方面的学者的研究，并不断构思我的系统的架构与组成。在这个过程中，我了解SAT问题的发展并不是一帆风顺的，在研究过程中一波三折。它作为最早证明的NP完全问题，是计算机科学发展的重要里程碑。不断计算机科学家为该理论的发展做出了重要的贡献。

在写DPLL过程中，我考虑到单子句传播过程中遍历查找会消耗大量的时间。所以对数据结构进行了优化。在写回溯过程时，我遇到了逻辑上的问题，这是由于我逻辑不严密导致的错误。这也提醒我写程序的时候一定注意逻辑的严密性。

通过这次课程设计，我意识到自己还有很多不足，以后还要多学习。

# 参考文献

1. 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

[2] 360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

# 附 录

程序源代码：

//主控、交互与显示模块

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <windows.h>

#include <winnt.h>

//solver.h

#define SATISFIABLE 1

#define UNSATISFIABLE -1

#define UNCERTAIN 0

#define CONFLICT -2

#define false 0

#define true 1

//定义颜色

#define GREEN 2 //绿色

#define RED 4 //红色

#define WHITE 7 //白色

int \*valuation;//记录各变量赋值的数组，初值初始化为-1，表示不确定

int variableNumber,clauseNumber;//记录变元数和子句数量

//int \*sord;

struct Literal {//文字节点

//struct Literal \*prev;//指向子句的上一文字结点

struct Literal \* next; // 指向子句的下一文字结点

int index;//索引

//int c\_flag;//1表示文字被假删除,当前回溯层

// int p\_flag;//1表示文字被假删除，前一回溯层

};

struct Clause {

struct Literal \* head; // 指向子句的第一个文字

//struct Clause \* next; // 指向集合中的下一个子句

int Cvalue;//当前回溯层子句状态，初始为uncertain

//int Pvalue;//上一回溯层子句状态，初始为uncertain

int Literal\_number;// 子句中文字数量

//int Clause\_order;//标记子句次序

};

struct Clause \*ClauseSet;//创建一个子句集邻接表

struct LiterValueList{//检查表

int positive;//值为正的数目

int negative;//值为负的数目

};

struct LiterValueList \*LookupValue;//记录所有变量正负文字数目的检查表

//struct LiterValueList \*canChangeList;//可变化的检查表

struct LiteralLink{

int \* posOcccur;//标记正文字出现的子句索引

int \* negOccur;//标记负文字出现的子句索引

int chooseflag;//标记该层回溯次数，初始为0

};

struct LiteralLink \* LiteralSet;

struct Literal\_stack{

int \*base;

int \*top;//栈顶指针，指向栈顶元素上一个位置

};

struct Literal\_stack var\_stack;

//创建，初始化并返回一个空子句

struct Clause \* createClause();

// 创建，初始化并返回一个空文字

struct Literal \* createLiteral();

//初始化文字邻接表

int InitLiteralSet();

//初始化栈

int InitStack(struct Literal\_stack\* L\_stack);

//入栈函数

int Push(struct Literal\_stack\* L\_stack,int var);

//出栈函数，栈不空返回栈顶元素，栈为空返回false

int Pop(struct Literal\_stack\* L\_stack);

//读取cnf文件

struct Clause \*readfile(char\*filename);

//选择变量

int chooseLiteral();

//移除一个句子

void removeClause(struct Literal \* literal);

//移除子句集

void removeClauseSet();

//移除文字集

void removeLiteralSet();

//将结果写入res文件

int writeSolution(char \* filename,int result,int runtime);

// 实现单子句传播算法,传入子句集和经决策后的文字值，出现冲突返回UNSATISFIABLE

int Bcp(int unit\_var);

//回溯，相当于单子句传播的反过程

int backtrack(int backtrackIndex);

// 检查子句集的当前状态是否代表解决方案

int checkSolution();

//dpll改进算法

int dpll2();

//Sudoku.h

int soduko[9][9];//初始化数独,用来生成cnf文件

int knockout\_flag[9][9];//记录挖空后数组中为0的格子（既可以修改的格子）

int count;

int sodukoanswer[82];//记录数独文件解的数组

int checksodukolist[9][9];//检查表用来标志此位置是否挖过空,1为挖过

int readsodukofile(char \*filename);//读计算机解出的数独文件

void setcolor(int color);//设置字的颜色

int generateRandNum();//生成随机数

int printsoduko();//打印数独

int issodukoFull();//判断数独是否已全部填满

int checkChecklist(int checklist[9]);//检查checklist[9]是否全部非0

void generateFirstLine();//随机生成数独的第一行

int check();//检查填入的数字是否符合条件

int generatesoduko();//生成完整数独

int knockout\_soduko(int k\_soduko[9][9],int hard\_level);//挖空

int playsoduko();//人玩数独

int generatefinalsoduko(int hard\_level);//生成挖空数独

int playwithcnf();//电脑用cnf文件解数独

int sodukoToCnf(char \*sodukotocnffilename);

int printCnf();//逐行打印cnf文件，如果cnf文件不存在，则返回失败

int printRes();//打印sat求解结果

/\*

函数功能：创建，初始化并返回一个空子句

函数参数：无

返回值:分配成功返回结构指针，不成功退出程序

\*/

struct Clause \* createClause(){

struct Clause \* instance = malloc(sizeof(struct Clause));

if(!instance)//分配空间不成功，返回失败

exit(1);

instance->head = NULL;

instance->Cvalue=0;//初始为未确定

instance->Literal\_number=0;

return instance;

}

/\*

函数功能：创建，初始化并返回一个空文字

函数参数：无

返回值:分配成功返回结构指针，不成功退出程序

\*/

struct Literal \* createLiteral(){

struct Literal \* instance = malloc(sizeof(struct Literal));

if(!instance)//分配空间不成功，返回失败

exit(1);

instance->next = NULL;

instance->index = 0;//文字值设为0

return instance;

}

/\*

函数功能：初始化文字邻接表

函数参数：无

返回值:初始化成功返回true

\*/

int InitLiteralSet(){

int i;

for(i=1;i<variableNumber+1;i++){//分配空间

if(LookupValue[i].positive>0)

LiteralSet[i].posOcccur=(int\*)calloc(LookupValue[i].positive,sizeof(int));

else

LiteralSet[i].posOcccur=NULL;//没有正文字为空指针

if(LookupValue[i].negative>0)

LiteralSet[i].negOccur=(int\*)calloc(LookupValue[i].negative,sizeof(int));

else

LiteralSet[i].negOccur=NULL;

LiteralSet[i].chooseflag=0;

}

struct Literal \*tpNode;

int var;

for(var=1;var<variableNumber+1;var++){//遍历每一个文字

int j=0,k=0;

for(i=1;i<clauseNumber+1;i++){//创建文字邻接表

tpNode=ClauseSet[i].head;

while(tpNode){

if(tpNode->index==var){//正文字

LiteralSet[abs(tpNode->index)].posOcccur[j]=i;

j++;

}

else if(tpNode->index==-var){

LiteralSet[abs(tpNode->index)].negOccur[k]=i;

k++;

}

tpNode=tpNode->next;

}

}

}

return true;

}

/\*

函数功能：初始化栈

函数参数：栈指针

返回值:初始化成功返回true，不成功退出程序

\*/

int InitStack(struct Literal\_stack\* L\_stack){

L\_stack->base=(int\*)malloc((variableNumber+1)\*sizeof(int));

if(!L\_stack->base)

exit(1);

L\_stack->top=L\_stack->base;//初始为空

return true;

}

/\*

函数功能：将变量var入栈

函数参数：栈指针,变量值var

返回值:初始化成功返回true

\*/

int Push(struct Literal\_stack\* L\_stack,int var){

\*L\_stack->top=var;

L\_stack->top++;

return true;

}

/\*

函数功能：栈顶元素出栈，并返回栈顶元素值

函数参数：栈指针

返回值:出栈成功返回栈顶元素值，栈空返回失败

\*/

int Pop(struct Literal\_stack\* L\_stack){

if(L\_stack->top==L\_stack->base)//栈为空

return false;

L\_stack->top--;

int var=\*L\_stack->top;

return var;

}

/\*

函数功能：解析，读取cnf文件，基于一定的物理结构，建立公式的内部表示

函数参数：文件名指针

返回值:读取并建立结构成功后返回true，否则返回失败

\*/

struct Clause \*readfile(char\*filename){

FILE \*fp;

char line[600];

fp=fopen(filename,"r");

int debug=1;

if(fp==NULL)

{

return NULL;

}

struct Literal \*cLiteral=NULL,\*pLiteral=NULL;

int i,clauseorder=1;

while(fgets(line,sizeof(line),fp)){

//忽略注释

if(line[0]=='c')

continue;

if(line[0]=='p'){

sscanf(line,"p cnf %d %d",&variableNumber,&clauseNumber);

valuation=(int\*)calloc(variableNumber+1,sizeof(int));

LiteralSet=(struct LiteralLink \*)calloc(variableNumber+1,sizeof(struct LiteralLink));

LookupValue=(struct LiterValueList\*)calloc(variableNumber+1,sizeof(struct LiterValueList));

ClauseSet=(struct Clause \*)calloc(clauseNumber+1,sizeof(struct Clause));

if((valuation==NULL)||(LiteralSet==NULL)||(LookupValue==NULL)||(ClauseSet==NULL))

exit(1);

InitStack(&var\_stack);

for(i=0;i<variableNumber+1;i++){//初始值都赋值-1，表示不确定

valuation[i]=-1;//赋值表初始值都赋值-1，表示不确定

LookupValue[i].negative=0;//检查表初始值都赋值0

LookupValue[i].positive=0;

}

}

//为每一行子句创建链表

else{

char \*t;

t=strtok(line," ");//strtok为分解字符串的函数（以空格为分割）

while(t!=NULL){

int literalIndex=atoi(t);//atoi把字符串转换成整形数

cLiteral=createLiteral();

cLiteral->index=literalIndex;

if(literalIndex!=0){//读到0的时候结束

if(literalIndex>0){

LookupValue[abs(literalIndex)].positive++;

}

else{

LookupValue[abs(literalIndex)].negative++;

}

if(ClauseSet[clauseorder].head==NULL)

ClauseSet[clauseorder].head=cLiteral;

if(pLiteral!=NULL){//创建链表

pLiteral->next=cLiteral;

}

ClauseSet[clauseorder].Literal\_number++;

}

pLiteral=cLiteral;

t=strtok(NULL," ");

}

clauseorder++;

}

}

fclose(fp);

return ClauseSet;

}

/\*

函数功能：选择合适的变量进行单子句传播

函数参数：无

返回值:返回变量索引值

\*/

int chooseLiteral(){

//只返回第一个字面值，它不会改变结果，但最好使用更智能的方法来提高速度

// (e.g. 选择频率最高的文字)

int i,j;

for(i=1;i<clauseNumber+1;i++){//先找单子句

if(ClauseSet[i].Literal\_number==1&&ClauseSet[i].Cvalue==UNCERTAIN){

struct Literal \*tp=ClauseSet[i].head;

while(tp){

if(valuation[abs(tp->index)]==-1){

j= tp->index;

return j;

}

tp=tp->next;

}

}

}

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

if(ClauseSet[i].Literal\_number<3&&ClauseSet[i].Cvalue==UNCERTAIN){

struct Literal \*tp=ClauseSet[i].head;

while(tp){

if(valuation[abs(tp->index)]==-1){

j= tp->index;

return j;

}

tp=tp->next;

}

}

}

for(i=1;i<variableNumber+1;i++){//此版本适应8

if(valuation[i]==-1)

return i;

}

}

/\*

函数功能：移除一个文字之后所有的文字，释放空间

函数参数：文字指针

返回值:无

\*/

void removeClause(struct Literal \* literal){

while (literal != NULL) {

struct Literal \* next = literal->next;

free(literal);

literal = next;

}

}

/\*

函数功能：移除子句集，释放空间

函数参数：无

返回值:无

\*/

void removeClauseSet(){

int i;

if(ClauseSet){

for(i=0;i<clauseNumber+1;i++){

if (ClauseSet[i].head != NULL)

removeClause(ClauseSet[i].head);

}

free(ClauseSet);

}

}

/\*

函数功能：移除文字集，释放空间

函数参数：无

返回值:无

\*/

void removeLiteralSet(){

int i;

if(LiteralSet){

for(i=0;i<variableNumber+1;i++){

free(LiteralSet[i].posOcccur);

free(LiteralSet[i].negOccur);

}

free(LiteralSet);

}

}

/\*

函数功能：将程序求解的结果写入res文件

函数参数：要写入的文件的名字指针，求解结果result，运行时间runtime

返回值:成功返回true,失败返回false

\*/

int writeSolution(char \* filename,int result,int runtime){

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

if (f == NULL) {

return false;

}

if(result==SATISFIABLE)

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

else if(result==UNSATISFIABLE)

fprintf(f, "%c %d\n",'s',0);

else if(result==UNCERTAIN)

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

int i;

if(result==SATISFIABLE){

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

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

if(valuation[i])

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

else

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

}

}

fprintf(f, "\n%c %d\n",'t',runtime);

fclose(f);

return true;

}

/\*

函数功能：实现单子句传播算法

函数参数：要用于单子句传播的文字值

返回值:成功返回true

\*/

int Bcp(int unit\_var){

valuation[abs(unit\_var)]=unit\_var>0?1:0;//根据传入的值的正负改变赋值

int i,j;

if(unit\_var>0){//传入的为正文字

for(i=1;i<(LookupValue[unit\_var].positive)+1;i++){//含正文字的子句Cvalue记为1

if(LiteralSet[unit\_var].posOcccur){

ClauseSet[(LiteralSet[unit\_var].posOcccur[i-1])].Cvalue=1;

/////////////////////////////////////printf("BCP传入的为正文字%d ,第%d个子句值赋值1,有%d个文字\n",unit\_var,LiteralSet[abs(unit\_var)].posOcccur[i-1],ClauseSet[(LiteralSet[abs(unit\_var)].posOcccur[i-1])].Literal\_number);

}

}

for(i=1;i<(LookupValue[unit\_var].negative)+1;i++){//含负文字的子句文字假删，

if(LiteralSet[unit\_var].negOccur){

j=LiteralSet[unit\_var].negOccur[i-1];

ClauseSet[j].Literal\_number--;//文字数Literal\_number减1

if(ClauseSet[j].Literal\_number==0)

ClauseSet[j].Cvalue=CONFLICT;

/////////////////////////////////////////////printf("BCP传入的为正文字%d ,第%d个句子文字数减1，有%d个文字,值为%d\n",unit\_var,j,ClauseSet[j].Literal\_number,ClauseSet[j].Cvalue);

}

}

}

else if(unit\_var<0){//传入的为负文字

for(i=1;i<(LookupValue[-unit\_var].negative)+1;i++){//含负文字的子句Cvalue记为1

if(LiteralSet[-unit\_var].negOccur){

ClauseSet[(LiteralSet[-unit\_var].negOccur[i-1])].Cvalue=1;

//////////////////////////////////printf("BCP传入的为负文字%d ,第%d个子句值赋值1,有%d个文字\n",unit\_var,LiteralSet[abs(unit\_var)].negOccur[i-1],ClauseSet[(LiteralSet[abs(unit\_var)].negOccur[i-1])].Literal\_number);

}

}

for(i=1;i<(LookupValue[-unit\_var].positive)+1;i++){//含正文字的子句文字假删

if(LiteralSet[-unit\_var].posOcccur){

j=LiteralSet[-unit\_var].posOcccur[i-1];

ClauseSet[j].Literal\_number--;//文字数Literal\_number减1

if(ClauseSet[j].Literal\_number==0)

ClauseSet[j].Cvalue=CONFLICT;

///////////////////////////////////////printf("BCP传入的为负文字%d ,第%d个句子文字数减1，有%d个文字,值为%d\n",unit\_var,j,ClauseSet[j].Literal\_number,ClauseSet[j].Cvalue);

}

}

}

return true;

}

/\*

函数功能：回溯，将赋值数组还原，将子句值，子句文字数目还原到单子句传播前的状态

函数参数：用于回溯的文字值

返回值:成功返回true

\*/

int backtrack(int backtrackIndex){

valuation[abs(backtrackIndex)]=-1;

int i,j,Cvalue\_flag;

if(backtrackIndex>0){//正文字

//将含正文字的字句值还原

for(i=1;i<LookupValue[backtrackIndex].positive+1;i++){

Cvalue\_flag=0;

if(LiteralSet[abs(backtrackIndex)].posOcccur){

j=LiteralSet[abs(backtrackIndex)].posOcccur[i-1];

struct Literal\* Literaltp=ClauseSet[j].head;

while(Literaltp){

if(Literaltp->index>0&&Literaltp->index!=backtrackIndex){

if(valuation[Literaltp->index]==1){

//printf("回溯 传入%d 第%d个子句值还原为1",backtrackIndex,j)

Cvalue\_flag=1;

break;

}

}

else if(Literaltp->index<0&&Literaltp->index!=backtrackIndex){

if(valuation[-Literaltp->index]==0){

Cvalue\_flag=1;

break;

}

}

Literaltp=Literaltp->next;

}

if(Cvalue\_flag==0)

ClauseSet[j].Cvalue=UNCERTAIN;

}

}

//将含负文字的字句文字数恢复

for(i=1;i<LookupValue[backtrackIndex].negative+1;i++){

Cvalue\_flag=0;

if(LiteralSet[abs(backtrackIndex)].negOccur){

j=LiteralSet[abs(backtrackIndex)].negOccur[i-1];

ClauseSet[j].Literal\_number++;

//canChangeList[abs(backtrackIndex)].negative++;//还原canchhangelist

struct Literal\* Literaltp=ClauseSet[j].head;

while(Literaltp){

if(Literaltp->index>0&&Literaltp->index!=backtrackIndex){

if(valuation[abs(Literaltp->index)]==1){

//printf("回溯 传入%d 第%d个子句值还原为1",backtrackIndex,j)

Cvalue\_flag=1;

break;

}

}

else if(Literaltp->index<0&&Literaltp->index!=backtrackIndex){

if(valuation[abs(Literaltp->index)]==0){

Cvalue\_flag=1;

break;

}

}

Literaltp=Literaltp->next;

}

if(Cvalue\_flag==0)

ClauseSet[j].Cvalue=UNCERTAIN;

}

}

}

if(backtrackIndex<0){//负文字

//将含负文字的字句值

for(i=1;i<LookupValue[abs(backtrackIndex)].negative+1;i++){

Cvalue\_flag=0;

if(LiteralSet[abs(backtrackIndex)].negOccur){

j=LiteralSet[abs(backtrackIndex)].negOccur[i-1];

struct Literal\* Literaltp=ClauseSet[j].head;

while(Literaltp){

if(Literaltp->index>0&&Literaltp->index!=backtrackIndex){

if(valuation[abs(Literaltp->index)]==1){

Cvalue\_flag=1;

break;

}

}

else if(Literaltp->index<0&&Literaltp->index!=backtrackIndex){

if(valuation[abs(Literaltp->index)]==0){

Cvalue\_flag=1;

break;

}

}

Literaltp=Literaltp->next;

}

if(Cvalue\_flag==0)

ClauseSet[j].Cvalue=UNCERTAIN;

}

}

//将含正文字的字句文字数恢复，将负文字的假删解除

for(i=1;i<LookupValue[abs(backtrackIndex)].positive+1;i++){

Cvalue\_flag=0;

if(LiteralSet[abs(backtrackIndex)].posOcccur){

j=LiteralSet[abs(backtrackIndex)].posOcccur[i-1];

ClauseSet[j].Literal\_number++;

struct Literal\* Literaltp=ClauseSet[j].head;

while(Literaltp){

if(Literaltp->index>0&&Literaltp->index!=backtrackIndex){

if(valuation[abs(Literaltp->index)]==1){

Cvalue\_flag=1;

break;

}

}

else if(Literaltp->index<0&&Literaltp->index!=backtrackIndex){

if(valuation[abs(Literaltp->index)]==0){

Cvalue\_flag=1;

break;

}

}

Literaltp=Literaltp->next;

}

if(Cvalue\_flag==0)

ClauseSet[j].Cvalue=UNCERTAIN;

}

}

}

return true;

}

/\*

函数功能：检查子句集的当前状态,

函数参数：无

返回值:有子句有冲突返回conflict,如果有子句值为uncertain返回uncertain,否则返回sat

\*/

int checkSolution(){

int i;

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

if (ClauseSet[i].Cvalue==CONFLICT)//bcp后子句为空，不满足

return CONFLICT;

}

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

if(ClauseSet[i].Cvalue==UNCERTAIN)

return UNCERTAIN;

}

return SATISFIABLE;

}

/\*

函数功能：dpll主体框架

函数参数：无

返回值:子句集不满足返回unsat,子句集满足返回sat

\*/

int dpll2(){

int Set\_status,debug=1;

int literalIndex,conflictIndex,i;//用来bcp的索引

literalIndex=chooseLiteral();

Push(&var\_stack,literalIndex);//入栈,bcp

LiteralSet[abs(literalIndex)].chooseflag=1;

while(1){

while(1){

Set\_status=Bcp(literalIndex);//进行BCP过程

Set\_status=checkSolution();//判断子句集状态

while(Set\_status==CONFLICT){//存在冲突

conflictIndex=Pop(&var\_stack);

backtrack(conflictIndex);

if(LiteralSet[abs(conflictIndex)].chooseflag==1){

literalIndex=-conflictIndex;//用相反文字

LiteralSet[abs(literalIndex)].chooseflag=2;

Push(&var\_stack,literalIndex);//入栈,bcp

break;

}

else if(LiteralSet[abs(conflictIndex)].chooseflag==2){

if(!(var\_stack.base==var\_stack.top)){//栈非空

LiteralSet[abs(conflictIndex)].chooseflag=0;

}

else//栈空,unsat

return UNSATISFIABLE;

}

}

if(Set\_status==SATISFIABLE){///////////////????????

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

if (valuation[i]==-1)

valuation[i]=1;

}

return SATISFIABLE;

}

else if(Set\_status==UNCERTAIN){

literalIndex = chooseLiteral();//选一个未入栈的变元

LiteralSet[abs(literalIndex)].chooseflag=1;

Push(&var\_stack,literalIndex);//入栈

break;

}

}

}

}

//Sudoku.c

/\*

函数功能：读数独的解文件

函数参数：要打开的文件名指针

返回值:成功返回true,失败返回false

\*/

int readsodukofile(char \*filename) //

{

FILE \*fp;

fp=fopen(filename,"r");

if (fp==NULL) //打开文件失败

{

printf("File open error!\n ");

return false;

}

char c\_result,gettt;

int num\_result;

fscanf(fp,"%c %d\n%c ",&c\_result,&num\_result,&gettt);

int num\_value;

int i=1;

for(i=1;i<=729;i++){

fscanf(fp,"%d ",&num\_value);

if(num\_value>0)

sodukoanswer[(abs(num\_value)-1)/9+1]=(abs(num\_value)-1)%9+1;

}

fclose(fp); //关闭文件

return true;

}

/\*

函数功能：为打出来的字设置颜色

函数参数：颜色的色号

返回值:无

\*/

void setcolor(int color)

{

HANDLE hConsoleWnd;

hConsoleWnd = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsoleWnd,color);

}

/\*

函数功能：生成1-9的随机数

函数参数：颜色的色号

返回值:返回生成的随机数的值

\*/

int generateRandNum(){

return rand()%9+1;

}

/\*

函数功能：打印数独

函数参数：无

返回值:成功返回true

\*/

int printsoduko(){

int i=0,j=0;

printf(" ");

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

printf("%2d",i+1);

}

printf("\n\n");

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

printf("%-2d",i+1);

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

if(soduko[i][j]&&knockout\_flag[i][j]==0)//原数字

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

else if(soduko[i][j]==0){//0为红色

setcolor(RED);

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

}

else{

setcolor(GREEN);

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

}

setcolor(WHITE);//还原系统色

}

printf("\n");

}

printf("\n");

return true;

}

/\*

函数功能：判断数独是否已全部填满

函数参数：无

返回值:填满返回true,没有填满返回false

\*/

int issodukoFull(){

int flag[81] = {0};

int i,j;

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

{

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

{

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

flag[i\*9 + j] = 1;

}

}

for(i = 0; i < 81; i++)

{

if(flag[i] == 0)

return false;//没填满

}

return true;

}

/\*

函数功能：检查检查表是否有0

函数参数：检查表数组

返回值:没有0返回true,有0返回false

\*/

int checkChecklist(int checklist[9]){

int i = 0;

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

{

if(checklist[i] == 0)

{

return false;//有0

}

}

return true;

}

/\*

函数功能：随机生成数独第一行

函数参数：无

返回值:无

\*/

void generateFirstLine(){

int checklist[9]={0},randnum,addnum=0;

srand((unsigned)time(NULL));

while(!checkChecklist(checklist)){

randnum=generateRandNum();

if(checklist[randnum - 1]==0)

{

checklist[randnum - 1] = 1;

soduko[addnum/9][addnum % 9] = randnum;

addnum++;

}

}

}

/\*

函数功能：检查当前数独格局是否满足数独要求

函数参数：无

返回值:满足返回true,否则返回false

\*/

int check()

{

int i,j,m,n;

//检查行

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

int tempArr[9] = {0};

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

if(soduko[i][j] != 0){

if(tempArr[soduko[i][j] - 1] == 0)

tempArr[soduko[i][j] - 1] = 1;

else

return false;

}

}

}

//检查列

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

int tempArr[9] = {0};

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

if(soduko[i][j] != 0){

if(tempArr[soduko[i][j] - 1] == 0)

tempArr[soduko[i][j] - 1] = 1;

else

return false;

}

}

}

//检查九宫格

for(m = 0; m < 9; m += 3){

for(n = 0; n < 9; n += 3){

int tempArr[9] = {0};

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

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

if(soduko[m + i][n + j] != 0){

if(tempArr[soduko[m + i][n + j] - 1] == 0)

tempArr[soduko[m + i][n + j] - 1] = 1;

else

return false;

}

}

}

}

}

return true;

}

/\*

函数功能：生成完整的未挖空的数独

函数参数：行号

返回值:数独填满返回true，否则返回false

\*/

int generatesoduko(int rownum){

if(issodukoFull()){//判断数独是否已全部填满

return true;

}

int i;

srand((unsigned int)time(0));

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

//printf("%3d",rownum/9);

soduko[(int)(rownum/9)][rownum%9] = i;

//if(isNumberFill((rownum/9),rownum%9))

//printf("%3d",rownum/9);

if(check()){

if(generatesoduko(rownum+1))

goto mdzz;

else

soduko[(int)(rownum/9)][rownum%9] = 0;

}

else

soduko[(int)(rownum/9)][rownum%9] = 0;

}

return false;

mdzz:

return true;

}

/\*

函数功能：将数独挖空

函数参数：完整数独，难度等级

返回值:挖空成功返回true

\*/

int knockout\_soduko(int k\_soduko[9][9],int hard\_level){//挖空

int rand\_row,rand\_col;//随机行号和列号

int k=0,knockout\_num=hard\_level;//knockout\_num为挖空数量

while(k<knockout\_num){

rand\_row=generateRandNum()-1;

rand\_col=generateRandNum()-1;

if(checksodukolist[rand\_row][rand\_col]==0){

checksodukolist[rand\_row][rand\_col]=1;

k\_soduko[rand\_row][rand\_col]=0;

knockout\_flag[rand\_row][rand\_col]=1;//挖过的格子记为1

k++;

}

}

return true;

}

/\*

函数功能：玩数独

函数参数：无

返回值:返回true

\*/

int playsoduko(){

int row=0,col=0,num=0,tep,p;

while(issodukoFull()==0){

printsoduko();//打印数独

printf("请输入行号，列号，要填的数.\n");

scanf("%d %d %d",&row,&col,&num);

if(row>0&&row<10&&col>0&&col<10&&num>0&&num<10&&knockout\_flag[row-1][col-1]==1){

tep=soduko[row-1][col-1];

soduko[row-1][col-1]=num;

if(!check()){

printf("输入数字不合理!\n");

soduko[row-1][col-1]=tep;

}

else

printf("修改成功!\n");

}

else{

printf("输入非法!\n");

}

printf("还要继续玩么？输入1继续玩，输入0退出.\n");

scanf("%d",&p);

if(!p){

printf("游戏结束!\n");

goto mdzz;

}

system("cls");

}

printf("游戏结束!\n");

mdzz:

return true;

}

/\*

函数功能：生成最终的挖空数独

函数参数：难度系数hard\_level

返回值:返回true

\*/

int generatefinalsoduko(int hard\_level){

generateFirstLine();

generatesoduko(9);

knockout\_soduko(soduko,hard\_level);//挖空

return true;

}

/\*

函数功能：电脑解数独

函数参数：无

返回值:返回true

\*/

int playwithcnf(){//机器玩数独

int i,j,k=1;

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

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

if(knockout\_flag[i][j]==1)

soduko[i][j]=sodukoanswer[k];

k++;

}

}

return true;

}

/\*

函数功能：将挖空的数独转化成sat问题，写入cnf文件

函数参数：写入的文件名指针

返回值:成功返回true，失败返回false

\*/

int sodukoToCnf(char \*sodukotocnffilename){

FILE \*fp;

fp=fopen(sodukotocnffilename,"w");

if(fp==NULL)

return FALSE;

//写开头信息

int i,j,notzeronum=0;

for(i=0;i<9;i++){ //每格不为值valueA或值valueB，共36\*81=2916句

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

if(soduko[i][j]>0)

notzeronum++;

}

}

int total\_clauses=11988+notzeronum;

fprintf(fp,"%c %s %d %d\n",'p',"cnf",729, total\_clauses);

int row,col,putnum;

int debug=0;

//写单个格子

for(row=1;row<10;row++){//每格可以为1-9，共9\*9=81句

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

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

//fprintf(fp,"%d ",row\*100+col\*10+putnum);

fprintf(fp,"%d ",((row-1)\*9+col-1)\*9+putnum);

}

fprintf(fp,"0\n");

debug++;

}

}

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

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

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

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

//fprintf(fp,"%d %d 0\n",-(row\*100+col\*10+i),-(row\*100+col\*10+j));

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

debug++;

}

}

}

}

//写每一行

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

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

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

//fprintf(fp,"%d ",(row\*100+col\*10+i));

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

}

fprintf(fp,"0\n");

debug++;

}

}

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

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

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

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

//fprintf(fp,"%d %d 0\n",-(row\*100+col\*10+i),-(row\*100+j\*10+i));

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

}

}

}

}

//写每一列

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

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

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

//fprintf(fp,"%d ",(row\*100+col\*10+i));

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

}

fprintf(fp,"0\n");

}

}

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

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

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

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

//fprintf(fp,"%d %d 0\n",-(row\*100+col\*10+i),-(j\*100+col\*10+i));

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

}

}

}

}

int k,l;

//写每一个宫

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

for(row=1;row<10;row+=3){

for(col=1;col<10;col+=3){

for(k=row;k<row+3;k++){

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

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

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

}

}

fprintf(fp,"%d\n",0);

}

}

}

int m;

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

for(row=1;row<10;row+=3){

for(col=1;col<10;col+=3){

for(l=col;l<col+3;l++){

for(k=row;k<row+3;k++){

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

for(m=k;m<row+3;m++){

//fprintf(fp,"%d %d 0\n",-(k\*100+l\*10+i),-(k\*100+j\*10+i));

if(m==k&&j<=l){

}

else{

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

}

}

}

}

}

}

}

}

//写已经填的空

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

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

if(soduko[i][j]>0){

//fprintf(fp,"%d 0\n",(i+1)\*100+(j+1)\*10+soduko[i][j]);

fprintf(fp,"%d 0\n",((i)\*9+j)\*9+soduko[i][j]);

}

}

}

fclose(fp);

return true;

}

/\*

函数功能：逐行打印cnf文件

函数参数：无

返回值:成功返回true，如果cnf文件不存在，失败返回false

\*/

int printCnf(){

if(ClauseSet==NULL)

return false;

else{

int i;

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

struct Literal \* printfCnf\_tp=ClauseSet[i].head;

while(printfCnf\_tp){

printf("%-5d",printfCnf\_tp->index);

printfCnf\_tp=printfCnf\_tp->next;

}

printf("%-5d\n",0);

}

}

return true;

}

/\*

函数功能：打印结果

函数参数：无

返回值:失败返回false

\*/

int printRes(){

if(LiteralSet==NULL)

return false;

else{

int i;

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

if(valuation[i]==0)

printf("%-5d",-i);

else if(valuation[i]==1)

printf("%-5d",i);

else

printf("%-5d",i);

}

printf("\n");

}

}

/\*

函数功能：生成程序菜单界面

函数参数：无

返回值:成功返回true

\*/

int main(){

int op=1;

char filepath[256]="\0";

char savepath[256];

clock\_t start,end;

int result=2;

int runtime;

while(op) {

system("cls");

printf("\n\n");

printf("-----------基于SAT的数独游戏求解程序菜单-----------\n");

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

printf("当前加载文件: %s\n",filepath);

if(ClauseSet) {

printf("变元数: %4d\n",variableNumber);

printf("子句数: %4d\n",clauseNumber);

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

}

printf("1. 加载CNF文件\n");

printf("2. 求解CNF\n");

printf("3. 打印CNF\n");

printf("4. 输出结果\n");

printf("5. 数独\n");

printf("0. 退出\n");

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

printf("请选择你的操作[0~5]:");

scanf("%d",&op);

switch(op) {

case 1:

if(ClauseSet) {//初始化两个邻接表指针

removeClauseSet();

removeLiteralSet();

free(LookupValue);

// free(canChangeList);

ClauseSet=NULL;

LiteralSet=NULL;

LookupValue=NULL;

// canChangeList=NULL;

}

printf("请输入CNF文件路径: ");

scanf("%s",filepath);

if(!readfile(filepath)){//读入cnf文件))

printf("文件打开失败！\n");

}

else

printf("文件加载成功!\n");

getchar();

getchar();

break;

case 2:

if(!ClauseSet) {

printf("CNF不存在！\n");

}

else {

printf("\n正在使用求解器求解...\n");

InitLiteralSet();

start = clock();

result=dpll2();

end = clock();

runtime=(int)(end-start)\*1000/CLOCKS\_PER\_SEC;

if(result== SATISFIABLE)

printf("SATISFIABLE\n");

else if(result==UNSATISFIABLE)

printf("UNSATISFIABLE\n");

else

printf("UNCERTAIN\n");

printf("DPLL用时 %d ms\n",runtime);

printf("请输入RES文件路径:\n");

scanf("%s",savepath);

if(!writeSolution(savepath,result,runtime))//读入cnf文件))

printf("文件保存失败！\n");

else

printf("文件保存成功!\n");

}

getchar();

getchar();

break;

case 3:

if(!ClauseSet)

printf("CNF不存在！\n");

else{

printf("正在打印CNF...\n");

printCnf();

printf("打印完成！\n");

}

getchar();

getchar();

break;

case 4:

if(result==2)

printf("RES不存在！\n");

else{

if(result== SATISFIABLE){

printf("SATISFIABLE\n");

printRes();

}

else if(result==UNSATISFIABLE)

printf("UNSATISFIABLE\n");

else

printf("UNCERTAIN\n");

}

getchar();

getchar();

break;

case 5:

system("cls");

sudokupage();

case 0:

break;

}

}

if(ClauseSet) {//初始化两个邻接表指针

removeClauseSet();

removeLiteralSet();

free(LookupValue);

// free(canChangeList);

ClauseSet=NULL;

LiteralSet=NULL;

LookupValue=NULL;

// canChangeList=NULL;

}

return true;

}

/\*

函数功能：生成数独菜单界面

函数参数：无

返回值:无

\*/

void sudokupage(){

int op=1;

clock\_t sudoku\_start,sudoku\_end;

int sudoku\_result=2,sudoku\_runtime;

int hard\_level;

printf("请输入难度等级(1~5):\n");

scanf("%d",&hard\_level);

while(hard\_level<1||hard\_level>5){

printf("输入不合法\n");

scanf("%d",&hard\_level);

}

generatefinalsoduko(17+hard\_level\*10);

printf("生成数独成功\n");

while(op){

system("cls");

printf("\n\n");

printf("-----------基于SAT的数独游戏求解程序菜单-----------\n");

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

if(ClauseSet) {

printf("变元数: %4d\n",variableNumber);

printf("子句数: %4d\n",clauseNumber);

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

}

printf("1. 玩数独\n");

printf("2. 使用sat求解器解数独\n");

printf("0. 回退\n");

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

printf("请选择你的操作[0~3]: ");

scanf("%d",&op);

switch(op) {

case 1:

playsoduko();

getchar();

getchar();

break;

case 2:

if(ClauseSet) {//初始化两个邻接表指针

removeClauseSet();

removeLiteralSet();

free(LookupValue);

// free(canChangeList);

ClauseSet=NULL;

LiteralSet=NULL;

LookupValue=NULL;

// canChangeList=NULL;

}

sodukoToCnf("sudoku.cnf");

readfile("sudoku.cnf");

InitLiteralSet();

sudoku\_start = clock();

sudoku\_result=dpll2();

sudoku\_end= clock();

sudoku\_runtime=(int)(sudoku\_end-sudoku\_start)\*1000/CLOCKS\_PER\_SEC;

printf("用时 %d ms\n",sudoku\_runtime);

writeSolution("sudoku.res",sudoku\_result,sudoku\_runtime);

readsodukofile("sudoku.res");

playwithcnf();

printsoduko();//打印数独

getchar();

getchar();

break;

case 0:

break;

}

}

}