

**课程设计报告**

**题目：基于SAT的二进制数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： CS1906**

**学 号： U201915116**

**姓 名： 廖翔**

**指导教师： 纪俊文**

**报告日期： 2021/03/08**

**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：**将二进制数独游戏[5，6]问题转化为SAT问题[6]，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[6-9]。(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]TanbirAhmed.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]CarstenSinz.Visualizing SAT Instances and Runsof the DPLL Algorithm.JAutom Reasoning (2007) 39:219–243

[5] Binary Puzzle：<http://www.binarypuzzle.com/>

[6] Putranto H. Utomo and Rusydi H. Makarim. Solving a Binary Puzzle. Mathematics in Computer Science,(2017) 11:515–526

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

[8]InsLynce and JolOuaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

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

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

<http://zhangroup.aporc.org/images/files/Paper_3485.pdf>

**目 录**

**[任 务 书 3](#_Toc1809954636)**

[设计内容 3](#_Toc801707497)

[设计要求 3](#_Toc985500801)

[参考文献 4](#_Toc1918076743)

**[1 引言](#_Toc1238794484)** [6](#_Toc1238794484)

[1.1 课题背景与意义 6](#_Toc564934923)

[1.1.1 课程目标 6](#_Toc836047474)

[1.1.2 课程任务 7](#_Toc464393197)

[1.2 国内外研究现状 7](#_Toc1100888781)

[1.3 课程设计的主要研究工作 9](#_Toc2066123362)

**[2 系统需求分析与总体设计](#_Toc524773144)** [10](#_Toc524773144)

[2.1 系统需求分析 10](#_Toc146892979)

[2.2 系统总体设计 10](#_Toc1371587650)

**[3 系统详细设计](#_Toc1184166652)** [12](#_Toc1184166652)

[3.1 有关数据结构的定义 12](#_Toc1557963415)

[3.2主要算法设计 13](#_Toc423008034)

[3.2.1 CNF文件处理 13](#_Toc1325155868)

[3.2.2 DPLL算法处理 14](#_Toc341770439)

[3.2.3 DPLL算法优化方案 17](#_Toc1764496195)

[3.2.4 二进制数独处理 17](#_Toc1285867942)

**[4 系统实现与测试](#_Toc1454561433)** [20](#_Toc1454561433)

**[5.总结与展望](#_Toc2007650630)** [41](#_Toc2007650630)

[5.1 全文总结 41](#_Toc1321076746)

[5.2工作展望 41](#_Toc503443689)

**[6. 体会](#_Toc292511843)** [43](#_Toc292511843)

**[参考文献](#_Toc656477318)** [44](#_Toc656477318)

**[附录](#_Toc1790788987)** [45](#_Toc1790788987)

# **1 引言**

**1.1 课题背景与意义**

对于计算机科学与技术、信息安全与物联网专业大二学生，在前三个学期已经学习了C语言程序设计、数据结构两门面向编程知识与技术的基础理论课，以及C语言程序设计实验、数据结构实验两门编程实践课程，不仅具有较为系统性的C语言、常用数据结构基本知识，而且具有初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。但两门实验课仍属于对基本编程模型与技术的验证性训练，而“程序设计”综合课程设计正是使大家从简单验证到综合应用，甚至在编程中实现智慧与风格升华的重要实践环节，为后续学习与进行计算机系统编程打下坚实的基础，让综合编程技能成为大家的固有能力与通向未来专业之门的钥匙。

**1.1.1 课程目标**

基于“综合程序设计”课程设计实践课程规划原则及其在计算机相关专业人才培养中的地位，其应该体现与达到如下目标：

⑴综合性训练目标：在该课程中涉及C语言的主要编程要素，如典型的数据类型与控制结构；覆盖多种典型的数据结构如线性结构、二叉树与树结构、图结构及查找表结构等。从先前实验课的单要素或单一结构训练向多要素，多结构综合应用训练转变。

⑵培养应用问题的求解能力：程序设计是为问题求解服务的，提高对应用问题进行分析，数据抽象与建模，及问题定义与功能划分等综合分析与表示能力。

⑶程序编写向程序设计转化：在实验课程中，老师基本描述了相关数据结构，程序框架及主要算法，基于此进行程序编写训练，其属于验证与复现性编程实践。“综合程序设计”要求同学们基于对应用问题的分析，建立求解模型，设计数据结构与主要算法，从而进行程序设计，更多地体现“设计”的内涵与份量。

⑷进一步培养编程规范性与工程化素养：通过“综合程序设计”实践进一步培养良好的规范性编程习惯，以及一定的程序设计与软件开发的工程化素养，按照问题定义、必要的需求分析、系统设计、编程实现、程序测试分析及编制综合程序设计课程设计报告的流程组织本实践课程的开展与进行，形成初步的工程化程序设计素养。

**1.1.2 课程任务**

在选择与确定了“综合程序设计”课程设计问题之后，按工程化的基本流程分别完成如下任务：

⑴阅读“综合程序设计”课程设计任务书，熟悉问题，查阅文献，了解问题背景及相关知识。

⑵对设计问题进行需求分析，分析问题中所涉及的数据对象，划分功能，人机交互需求与数据文件读写等，并对问题进行形式化表示。

⑶基于上述需求分析，进行系统设计，明确程序的模块结构；设计数据结构（逻辑结构及其物理结构），参考并设计主要子问题的求解算法。

⑷程序实现，基于系统设计，制定相应的实现方案，编写各程序模块，完成程序编写与调试任务。

⑸程序测试，设计测试用例对程序进行功能测试，性能测量及理论分析。

⑹程序优化，对设计方案中的结构，算法进行一定优化，测试与分析性能改善结果。

⑺设计总结，按规范化要求撰写“综合程序设计”课程设计报告。

⑻成果提交：将程序源代码/工程文件、可独立运行的可执行程序、简要操作手册及“综合程序设计”课程设计报告电子版打包，文件夹名称格式为“专业班级-学号姓名”,如：CS1702-U201714999李某某。并将设计报告打印为纸质版（A4双面打印），然后以班为单位在指定时间（一般在设计课结束后两周内）集体提交到指导老师。

后面将对一个或多个候选设计问题进行问题与要求描述及设计指导，每个同学选择其中一题作为自己的综合程序设计课程设计问题。

**1.2 国内外研究现状**

Bart Selman和Henry Kautz分别于1997年和2003年在人工智能第五届国际合 作会议上提出了 SAT 问题面临的十大挑战性问题[1]，并在 2001 年和 2007 年先后 对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出 对于 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。 SAT 解决器的实现是我们关心的主要问题，目前的 SAT 算法大致可以归结为 两大类：完备算法（也称回溯搜索算法）和局部搜索算法。其中，完备算法大都 是基于回溯搜索的，局部搜索算法是基于局部随机搜索的。完备算法基于穷举法 思想，它的优点是能保证找到对应 SAT 问题的解或证明公式不可满足，但是效率 极低，它的平均时间复杂度虽是多项式级的，但是最坏情况下的时间复杂度却是 指数级的。一些完备算法采用了精巧的技术来减小搜索空间和问题规模，提高了 算法的时间效率，这一类算法将是本文研究的重点。局部搜索算法相对于完备算 法而言，由于采用了启发式策略来指导搜索，使得求解速度相对较快，但是在某 些实例上可能得不到解，它不保证一定能够找到对应 SAT 问题的解，即它不能证 明 SAT 问题的不可满足性。局部搜索算法的研究热潮是在最近几年才兴起的，本文没有重点研究这类算法，而只是将其作为一个必要步骤应用到了完备算法的求 解过程中。 最经典的求解 SAT 问题的完备算法是 DPLL 算法，它是由 Davis 和 Putnam 等 人在 1960 年提出[3]，其它的完备算法大都是在 DPLL 算法的基础上衍生出来的， 是对 DPLL 算法的改进。由于 SAT 问题本身的特性使得其最坏情况下的时间复杂 度是指数级别，最初这使得许多的研究者望而却步。而后，S.A.Cook 在 1971 年证 明了 SAT 问题是 NP 完全问题，这更加削弱了许多学者研究 SAT 问题的兴趣，从 而导致了 SAT 问题在很长的一段时间里都没有得到较好的重视，发展非常缓慢， 研究成果较少。但是1996年以后，很多国家都相继举办了一些 SAT 竞赛和研讨会，

这使得越来越多的人开始关注并研究 SAT 问题，所以这段时间也涌现出了众多新的高效的SAT 算法如 MINISAT[2]、SATO[4]、CHAFF[5]、POSIT[6]和 GRASP[7]等， SAT 算法的研究成果显著，求解算法也越来越多地应用到了实际问题领域。这些新兴的算法大都是基于 DPLL 算法的改进算法，改进的方面包括：采用新的数据 结构、新的变量决策策略或者新的快速的算法实现方案。国内也涌现出了许多高 效的求解算法，如 1998 年作者梁东敏提出了改进的子句加权 WSAT 算法[8]，2000 年金人超和黄文奇提出的并行 Solar 算法[9]，2002 年作者张德富在文献[10]中，提出模拟退火算法。

**1.3 课程设计的主要研究工作**

本设计要求精心设计问题中变元、文字、子句、公式等有效的物理存储结构，基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。在此基础上，通过改进算法、使用不同的选择策略和物理存储结构来优化求解过程，使求解器能更快地求解。完成基于此算法的数独游戏，通过将数独归约成SAT问题进行求解。

# **2 系统需求分析与总体设计**

## **2.1 系统需求分析**

1. 基于DPLL算法实现一个高效的SAT求解器，对于给定的中小规模测试算例进行求解，能输出求解结果和求解时间。
   1. 能够读取文件, 输出执行时间和执行结果
   2. 能够保存文件结果
   3. 能够对计算出的结果进行正确性验证
   4. 能够输出读取的文件
2. 数独游戏。将数独游戏转化为SAT问题进行求解，并具有一定的可玩性和交互性，要求如下：
   1. 能够生成数独格局
   2. 能将数独转化为SAT文件并导出CNF文件
   3. 能随机生成数独格局
   4. 能将求解结果化为数独的解

**2.2 系统总体设计**

本系统主要有3个模块，如下：

1. SAT求解模块
   1. 读取CNF文件并将其转化为设计的数据结构
   2. 可以查看并输出读入的CNF文件及相关信息(子句数，变元数)
   3. 保存求解结果
   4. 根据DPLL算法输出求解结果
2. 输出模块
   1. 将SAT求解结果和求解时间输出到控制台(包括是否可解)
   2. 将最终结果以文件形式保存
3. 数独模块
   1. 随机生成数独布局
   2. 能够选择特定空位置输入游玩
   3. 将数独转化为SAT问题求解
   4. 使用Qt实现的GUI界面，可玩性较强

流程图如下所示：

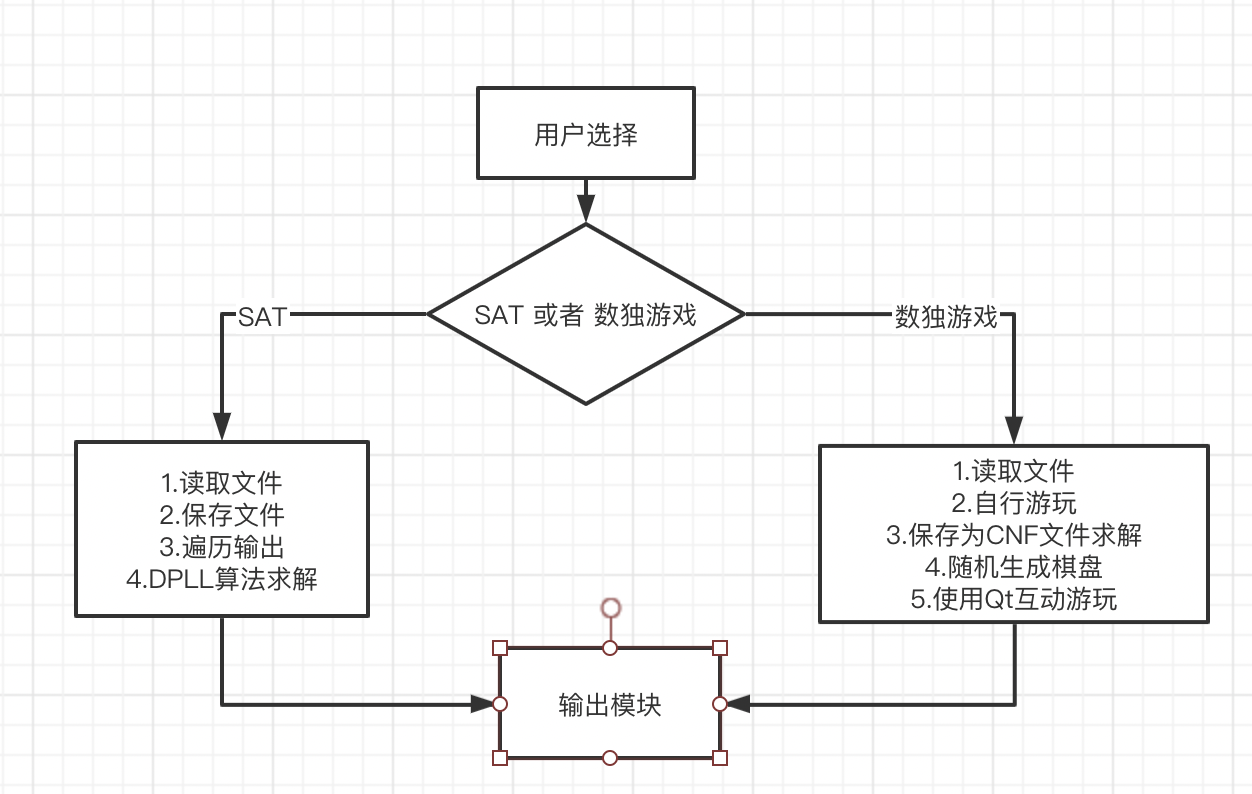


图2-2 系统模块流程图

**3 系统详细设计**

**3.1 有关数据结构的定义**

1. 对CNF文件进行读取与解析，处理CNF文件中的文字和公式。对于文字用一个结构体来定义，数据项有int型的文字值，还有文字结构型的指针用来指向下一个文字;对于公式也是用一个结构体来定义，数据项有文字结构型的指针用来指向公式中的第一个文字，还有一个公式结构型的指针用来指向下一个公式，此外还有一个表示该文字bool值的int型指针，具体如下：

表3-1 CNF文件处理时的数据结构

|  |  |  |
| --- | --- | --- |
| 数据名 | 数据类型 | 数据项 |
| 文字 | 结构型 | int型：文字的值  文字结构型：指针(指向下一个文字) |
| 公式/子句 | 结构型 | 文字结构型：指针(指向公式的第一个文字)  公式结构型:指针(指向下一个公式) |
| 表示文字bool值的指针 | Int \* | 无 |

更直观的数据结构图如下：

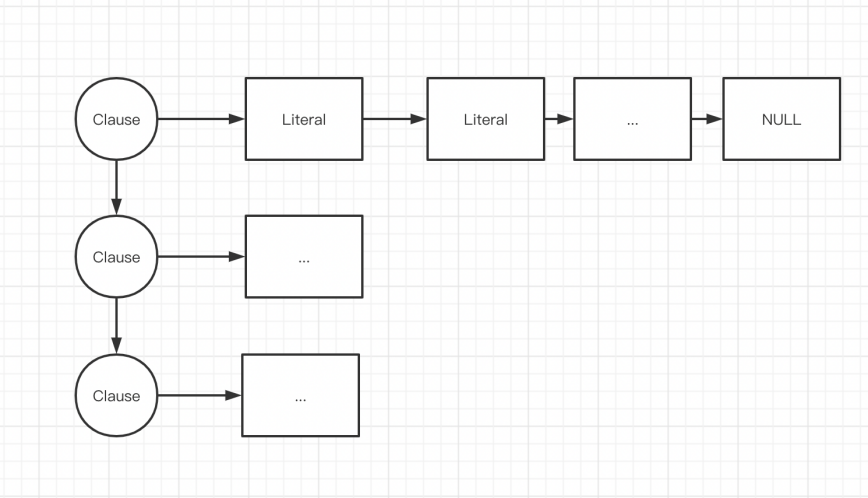


图3-1 CNF文件数据结构直观图

1. 另一个数据结构就是用来保存棋盘信息的数据结构，是一个结构型数组，每一个结构单元有两个数据项，一个int型的数据用来标志该位置是否为初始时就已经确定的位置;另一个int型的数据用来表示该位置存放的是0或1或空缺，如下图所示:

表2处理棋盘时的数据结构

|  |  |  |
| --- | --- | --- |
| 数据名 | 数据类型 | 数据项 |
| 棋盘 | 结构型数组 | int型：标志位，用来判断该位置是否可改  int型：用来判断该位置放置的值 |

## 3.2主要算法设计

### 3.2.1 CNF文件处理

**1. CNF文件读取**

利用fgets按行读取和strtok分割字符串来进行文件解析，先通过读取字符串行的首字符来判断是否已经读到目标内容，再进行构造存储结构的操作，在构建存储结构之前要对表示文字bool值的数组进行初始化，均设为-1，表示文字还未确定真值。

1. **CNF文件遍历输出**

利用while嵌套循环，循环条件就是相应指针不为NULL，按行一行一行输出cnf文件里的公式。

1. **CNF文件保存**

关键点与CNF文件遍历输出相同，利用while嵌套循环，循环条件就是相应指针不为NULL，按行一行一行将公式输入到指定的文件内。

### 3.2.2 DPLL算法处理

**1. 判断单子句**

寻找只包含一个文字的子句，即子句指向的第一个文字结构里指向下一个文字的指针为NULL，这时我们可以返回该单子句里唯一文字的数值，如果没有则返回0。

1. **处理单子句**

首先要调用判断单子句的函数来看是否存在单子句，如果不存在，返回ERROR，表示没有单子句可以处理；如果存在，继续操作。先将返回的文字值对应的标志bool值数组的位置修改为真值（返回文字为正，标志bool数组相应位置为1，若为负，则为0，即该返回文字的这种出现形式为true）。接下来就是删除包含该文字的子句以及删除以相反数形式出现的位于其他子句里的文字（主要用while()循环），如下图所示：

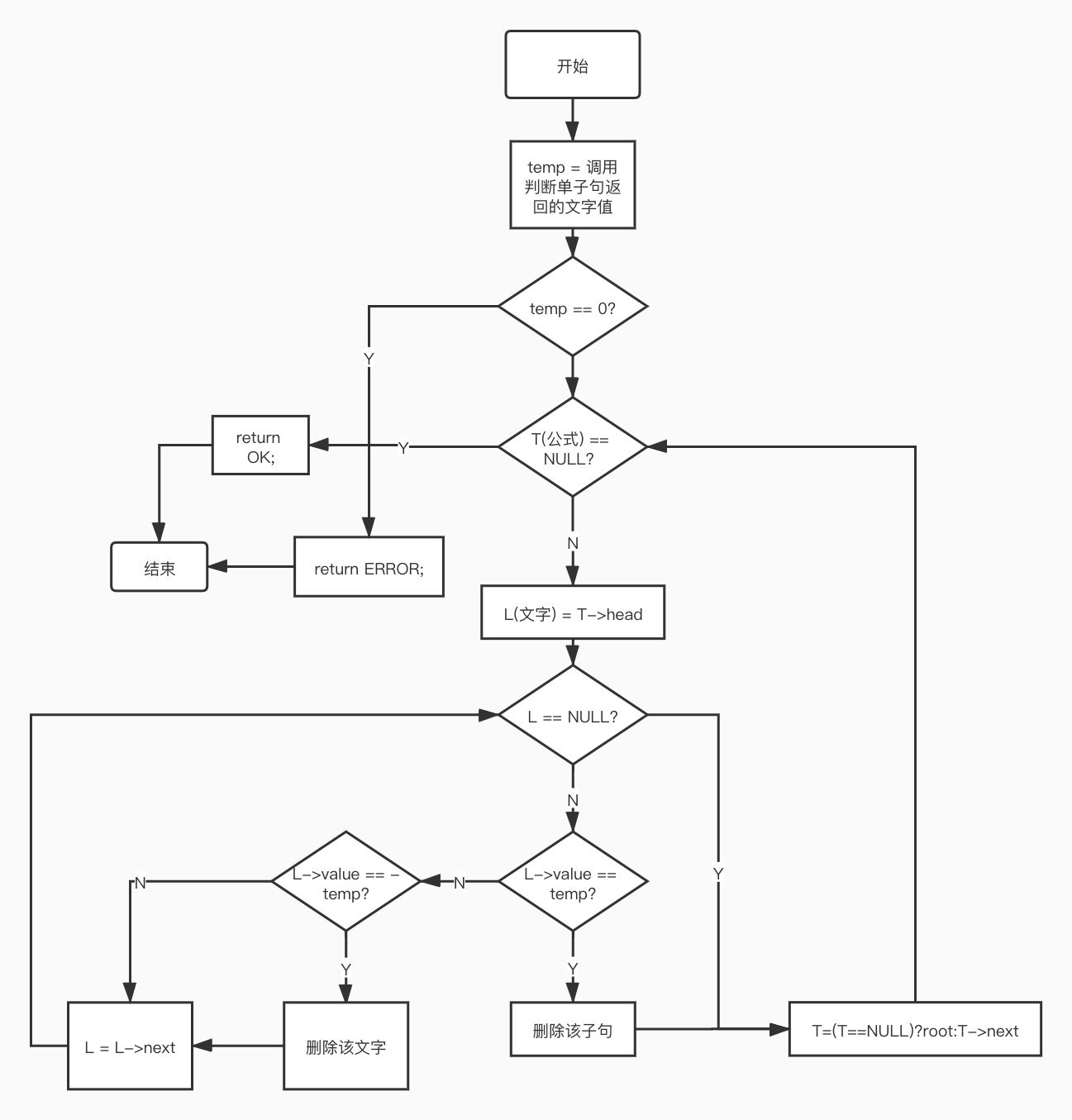


图3-2 处理单子句

1. **判断只有一种出现形式的文字**

这是优化选取文字策略的部分，如果一个文字在所有的子句里只有一种出现形式，那么可以确定它的bool值。这里需要一个标志数组来判断某一个文字上次出现时的符号，如果已经检测出某些文字有两种出现形式，那么相应的标志数组里的值为2；如果有一个文字它的符号一直没有变过，那么它就是该函数要返回的文字，如果没有则返回0，如图4所示：

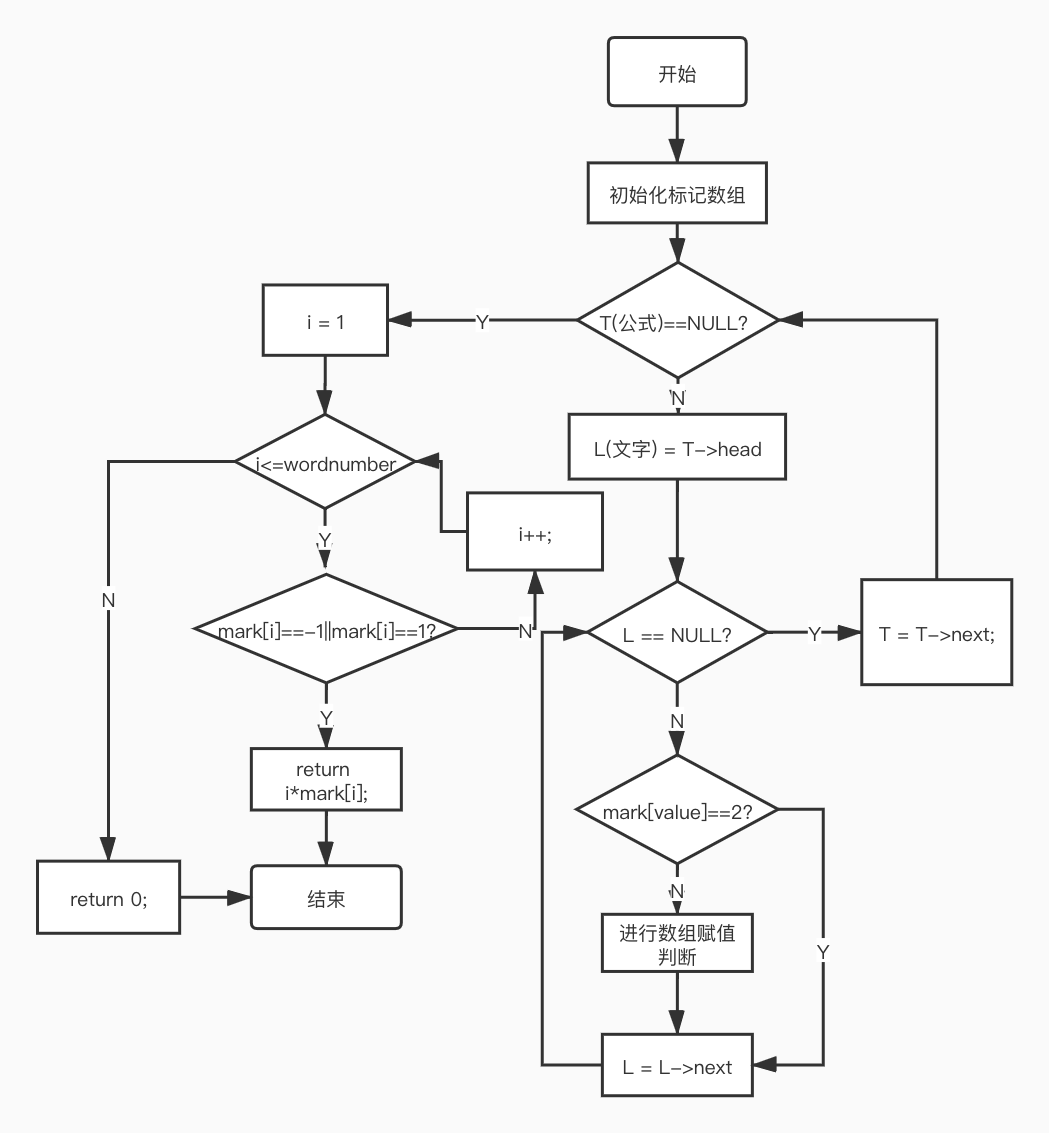


图3-3 寻找只有一种出现形式的文字

1. **处理只有一种出现形式的文字**

这里跟处理单子句是非常相似的，就是删除包含该文字的子句就可以了。首先还是调用寻找函数，如果返回值为0，那么该函数直接返回ERROR，表示没有相应的文字可以处理；如果返回值不为0，那么操作同处理单子句的第一种情况一样，即删除相应子句即可，返回OK，表示处理完毕；

1. **选取假设真值文字**

这个函数直接返回当前第一个公式的第一个文字即可。

1. **克隆子句**

这是下一个函数的辅助函数，即克隆一个子句，主要用的就是while循环，循环条件即为相应的指针不为NULL。

1. **克隆全部结构并将选取的文字设为单子句加入存储结构中**

先用while()循环来将存储结构复制，循环的判断条件还是相应指针不为NULL，然后创建新的子句，新的文字结点，文字的值从函数参数那里获得，将新的子句放在存储结构的最前。

1. **销毁子句**

用while()循环和free()函数将存储结构销毁。

1. **判断文字是否均为单一形式出现**

这里也是优化策略的一个方面，如果所有的文字均为一种出现形式，那么该cnf算例肯定可解，即不用判断最后是否为空集，提前就可以知道结果。这里的算法跟寻找只有一种出现形式的文字十分相似，在3.里发现一个文字与它上一次出现形式不同时是将标志数组相应位置设为2，而在这里直接返回ERROR，表示目前不满足所有文字均以单一形式出现的情况；如果满足，则根据文字出现形式来判断这些剩余文字的bool值，返回OK。

1. **判断是否存在空子句**

关键点就是判断是否存在公式结点不为NULL，但它指向的第一个文字结点为NULL的情况，如果存在，即存在空子句，返回OK；反之，返回ERROR。

1. **检查当前是否满足**

这个函数就是集中调用判断函数，先调用判断空子句，如果返回OK，则返回ERROR;如果判断空子句返回ERROR，那么再调用判断是否文字均为单一形式函数，如果返回OK，则返回OK；如果判断是否文字均为单一形式返回ERROR，那么则返回Untrue，表示暂时不确定。

1. **Dpll算法实现总函数**

这里先做简单表述，在下一章系统实现里将会用流程图直观描述。首先就是调用检查函数看是否满足，若是OK或者是ERROR，直接返回结果；如果为Untrue，那么接下来先是处理单子句，再看是否选择的是优化后的方案，这里我设置了一个全局变量WAY来判断，如果WAY = 2，那么接下来处理以一种形式出现的文字，如果WAY = 1，那么跳过第二阶段；接下来进行文字选取，如果不满足，测试选取文字的相反值。这里我避开了回溯的思想，而是在进行文字选取之后用克隆函数复制一个新的存储结构，用完之后就进行销毁，因此不用考虑回溯。

1. **将结果写入同名文件**

同名文件是用strcpy来实现的，文件名传入函数之后，先添加.res的后缀，然后依次按照是否可解，求解结果，求解时间的顺序将内容输入到文件内，所需要的数据为参数传入，求解结果是根据标志bool值的数组来判断的。

### 3.2.3 DPLL算法优化方案

1. **寻找只有一种出现形式的文字**

因为如果某一文字在所有的子句中只有一种出现形式，那么则可以直接判断它的bool值，然后再用类似于处理单子句的操作删除相应子句，这为确定文字bool值除寻找单子句以外又提供了一个新的选择，所以对于一些算例来说有很好的优化效果。

1. **判断是否所有文字均为一种出现形式**

如果所有的文字均为一种出现形式，那么该算例肯定满足，这相对于处理算例直到判断是否为空集来说省掉了部分时间。

1. **从出现频率最高的文字开始处理**

统计初始时频次最高的文字，优先从这个文字开始处理。

### 3.2.4 二进制数独处理

1. **读取棋盘文件**

我自己设立的棋盘文件形如图3-4所示：

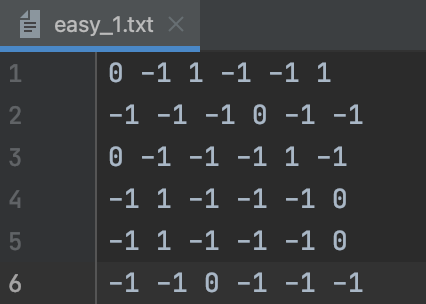


图3-4 DIY棋盘文件

在初始化棋盘的时候我用的是fscanf一个一个数字地读取，0表示该位置填充为0，1表示该位置填充为1，-1表示该位置为空，等待填充；并且在初始化的时候要在每一个已有数值填充的位置设置一个标志位（前面已解释），来标志其不可改。

1. **输出当前棋盘**

将棋盘信息输出到屏幕上，用-1表示的位置输出”\_”，其他的输出数值，输出6个位置换行即可。

1. **初始化棋盘**

将棋盘恢复到初始化的状态，这里还是根据标志位来判断，如果标志位不为1，说明这不是初始棋盘就已经填充的位置，那么就将数值改为-1即可。

1. **数独交互**

关键点就是如何填充的问题，我设计的是分别输入行，列，1或0来填补空缺。如果想要重新开始则输入三个0，结束填写则输入Ctrl+Z，这时候是用while()来判断是否继续填充的，循环条件就是输入的不为Ctrl+Z。这里的三个数我是用数组暂存的，然后用（行-1）\*6+列这个公式来进行棋盘改变。其中有利用标志位来提示某些位置不可改的操作，还有如果选择重新开始，则会调用初始化棋盘的函数。

1. **检查棋盘是否填充满**

这个只需要判断存储棋盘信息的结构数组里数值位没有-1即可，如果存在-1，说明还有空缺，返回ERROR；如果不存在-1，返回OK。

1. **转化为CNF文件**

这是二进制数独里最重要的一个函数，这里主要分成四部分转化，如下所示：

1. 根据棋盘信息：根据棋盘上填充的值（1和0）来给文字赋上符号并输入到cnf文件里，这里均为单子句；
2. 约束1（行，列中没有连续的三个0或1）：这里将形如”1 2 3 0“，”-1 -2 -3 0“这样的公式输出到文件里，这里比较容易处理，用for的嵌套循环实现；
3. 约束2（行，列中0，1个数相同）：这里我用一个二维数组将1，2，3，4，5，6这6个数15个不同的4组合设置出来，然后还是利用for的嵌套循环实现输出公式到文件里；
4. 约束3（没有相同的行或列）：总的来讲，我的附加变元转化是直接按顺序来的。通过总结每两行或两列之间的比较会产生61个子句，需要19个附加变元的规律，在这里我不妨先比较行，第一行与第二行，第一行与第三行...依次比较下去，利用for的嵌套循环，并设置好各个变量累加的值，比如每次需要添加19个附加变元，所以控制附加变元的相应变量每次累加值为19。还有一些规律如通过观察发现，这样按顺序添加附加变元之后有一个7个文字的子句，里面的文字为等差数列，差值为3，所以就可以利用这些类似的规律实现公式输出，列的比较紧跟在行的后面，同理。

**4 系统实现与测试**

#### 4.1 系统实现

##### 4.1.1 **软硬件环境**

1. **硬件环境**

处理器：Apple M1

机带RAM：16G

系统类型：64位操作系统，基于ARM架构处理器

1. **软件环境**

macOS下 CLion

macOS下 Qt Creator

##### 4.1.2 **数据类型定义**

1. **宏定义**

#define OK 1

#define ERROR 0

#define Untrue -1

1. **数据类型定义**

typedef int status;

1. **全局变量**

int wordnumber, formulanumber;//文字数，公式数

int \*variate;//用于判别文字的真值

int i;//用于循环

int temp;//中间变量

int WAY;//用于选择求解方式

1. **存储结构**

struct Literal{

struct Literal \* next;

int value;//文字值

};

struct Clause{

struct Clause \* next;

struct Literal \* head;

};

1. **保存棋盘的结构**

struct Qipan{

int flag;//用于标记是否为初始棋盘已确定的位置

int num;

};

struct Qipan qipan[36];//用于保存6阶棋盘信息

1. **其他**

char filename\_setcnf[15];//用于保存数独转化cnf文件名

##### 4.1.3 **函数声明以及函数调用关系**

1. **CNF文件处理**

/\*新建子句\*/

struct Clause \* createClause();

/\*新建文字\*/

struct Literal \* createLiteral();

/\*cnf文件读取\*/

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

/\*遍历输出CNF文件公式\*/

void Traversefile(struct Clause \* root);

/\*CNF文件保存\*/

status Savefile(struct Clause \* root);

1. **DPLL算法**

/\*辅助函数，判断正负\*/

int judgesign(int a);

/\*判断单子句，返回单子句里的文字\*/

int isUnitClause(struct Clause \* root);

/\*处理单子句\*/

status removeClause(struct Clause \* root);

/\*判断只有一种出现形式的文字\*/

int isPureLiteral(struct Clause \* root);

/\*处理只有一种出现形式的文字\*/

status removePureLiteral(struct Clause \* root);

/\*选取假设真值文字\*/

int chooseLiteral(struct Clause \*root);

/\*克隆子句\*/

struct Clause \* cloneClause(struct Clause \* headp);

/\*克隆全部结构并将选取的文字设为单子句加入存储结构中\*/

struct Clause \* cloneAll(int a, struct Clause \* root);

/\*销毁存储结构\*/

void destoryClauses(struct Clause \* root);

/\*判断是否文字均为单值\*/

status isAllPureLiteral(struct Clause \* root);

/\*判断是否存在空子句\*/

status isEmptyClause(struct Clause \* root);

/\*检查当前是否满足\*/

status check(struct Clause \* root);

/\*dpll算法\*/

status dpll(struct Clause \* root);

/\*将结果写入同名文件\*/

void writeResult(char \* filename, int flag, int time);

1. **二进制数独**

/\*读取棋盘文件\*/

void Readqipan(char \*filename);

/\*初始化棋盘\*/

void Resetqipan(struct Qipan qipan[]);

/\*输出当前棋盘\*/

void Showqipan(struct Qipan qipan[]);

/\*数独交互\*/

void PlaySuDoKu(struct Qipan qipan[]);

/\*检查棋盘是否填充满\*/

status Checkqipan(struct Qipan qipan[]);

/\*转化为CNF文件\*/

void SetCNFfile(struct Qipan qipan[]);

1. **Dpll函数调用关系，如图4-1所示：**

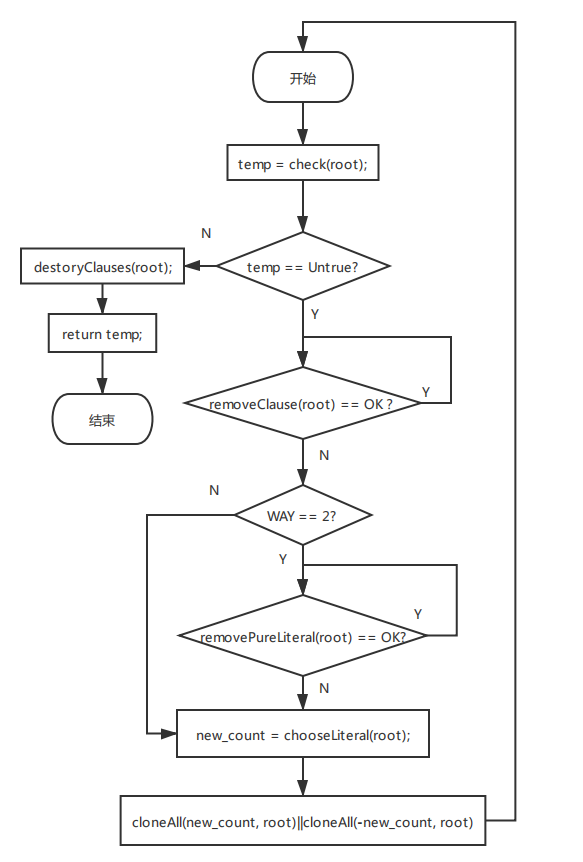


图4-1 Dpll函数调用关系

#### 4.2 系统测试

##### 4.2.1 交互系统展示

主交互界面，SAT求解界面，数独游戏界面如下：

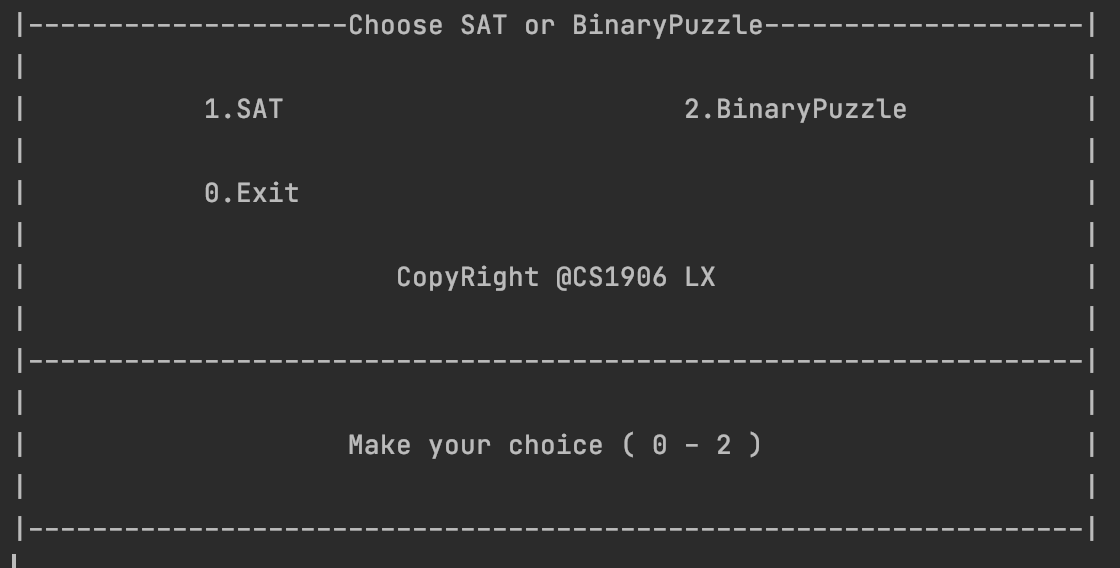


图4-2 主交互界面



图4-3 SAT求解界面

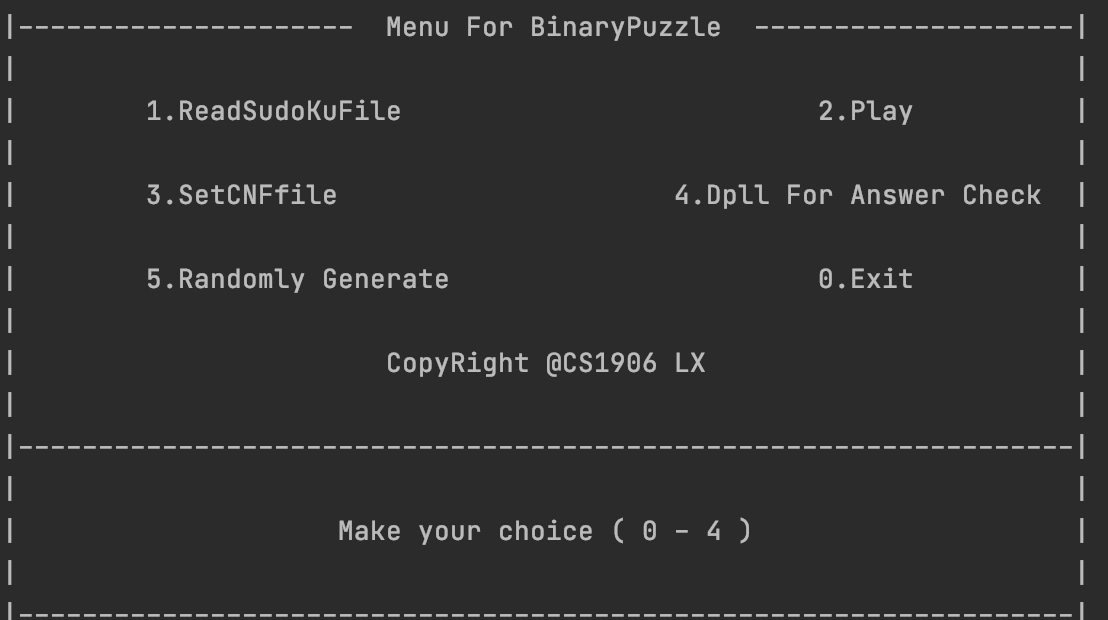


图4-4 数独界面

##### 4.2.2 CNF文件解析模块测试

1. **CNF文件的读取**

预计结果：输出文件的简要解析和读取成功提示

实际结果：如图4-5

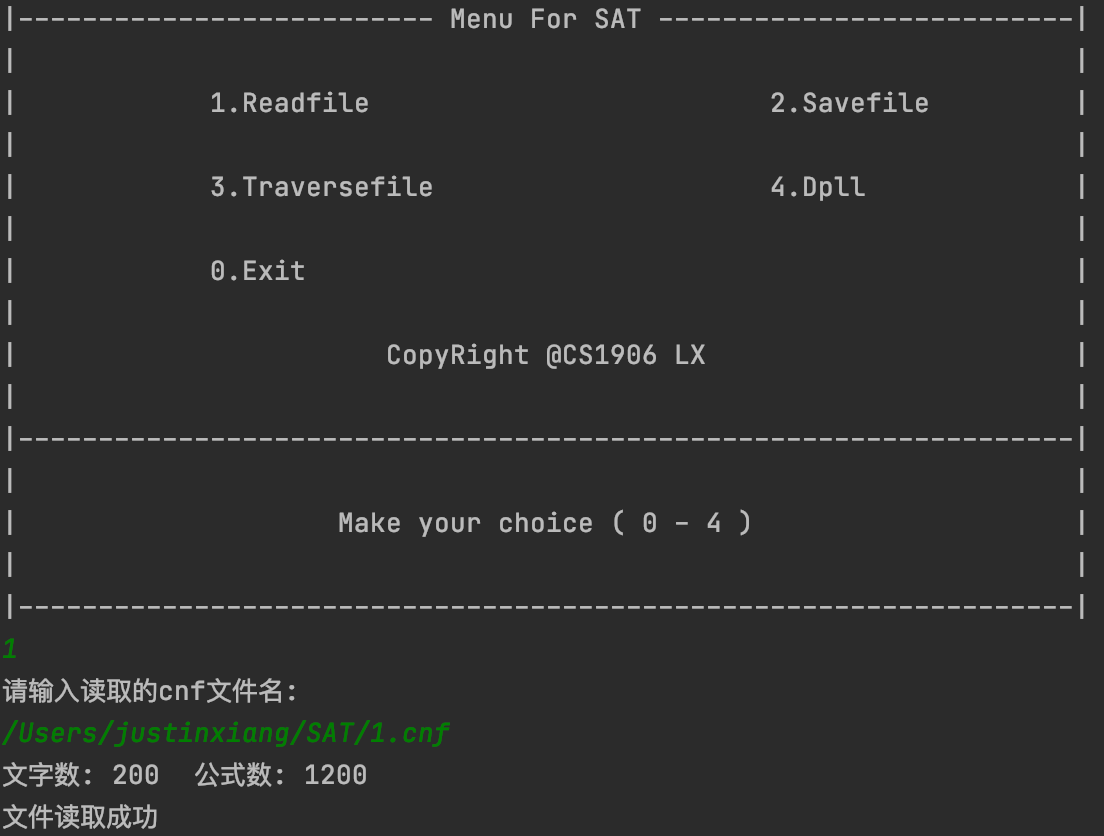


图4-5 CNF文件读取

1. **CNF文件的保存**

预期结果：输出保存成功提示

实际结果：如图4-6，4-7所示



图4-6 文件保存操作

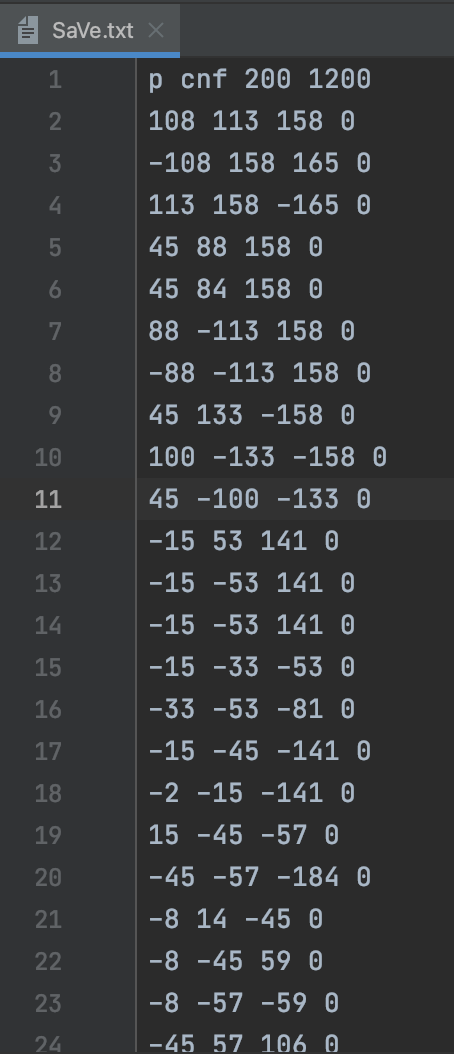


图4-7 打开文件验证

1. **CNF文件遍历输出**

实际结果：如图4-8，并与上文保存文件作对比

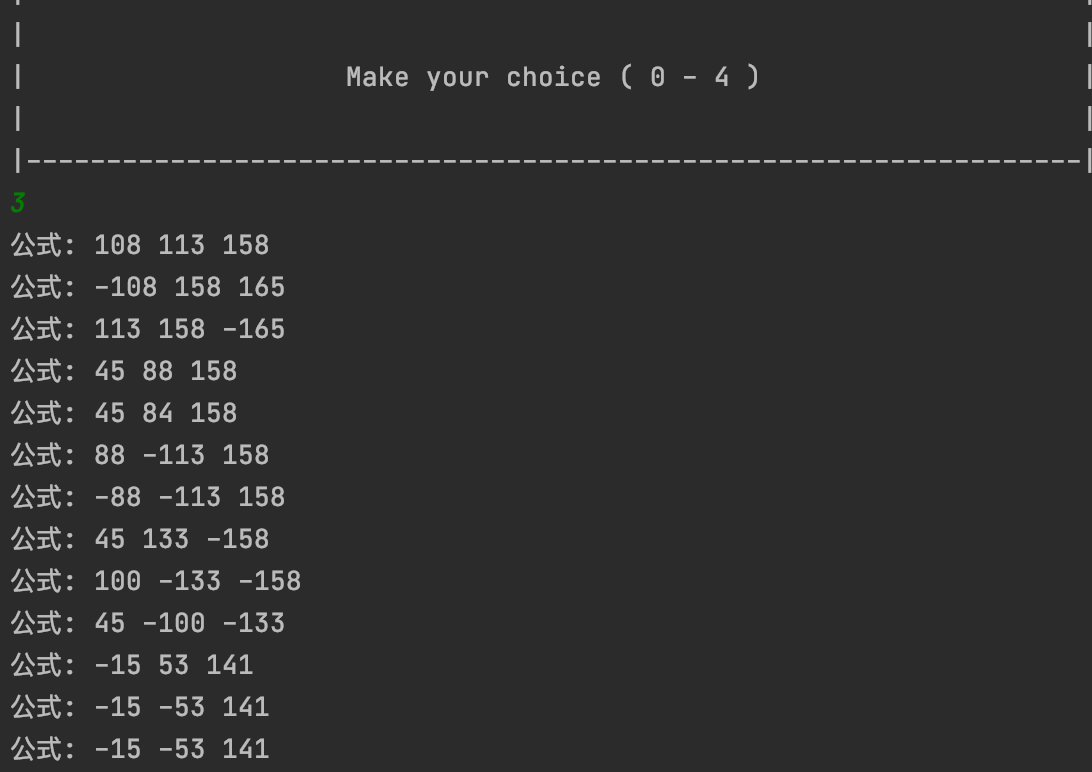


图4-8 CNF文件遍历输出

##### 4.2.3 DPLL算法模块测试

首先简介Dpll过程：先读取文件，如果未读取文件会提示错误信息；选择Dpll，然后还会有一个选项，可以选择用优化前的Dpll还是用优化后的Dpll来求解。

1. **测试算例1:sat-20.cnf 测试结果如图**

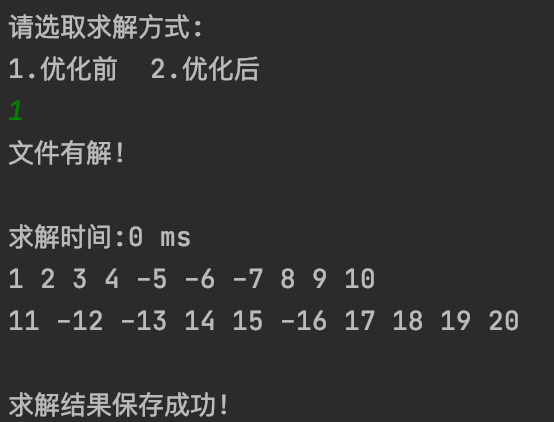


图4-9 优化前测试结果

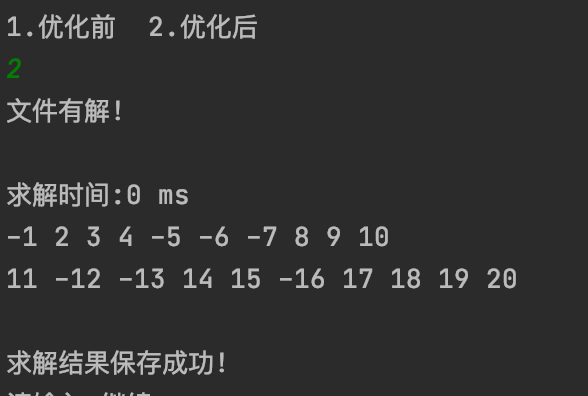


图4-10 优化后测试结果

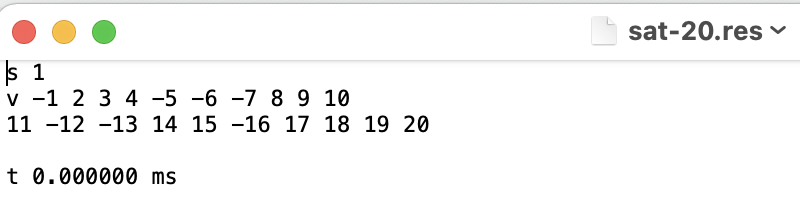


图4-11 保存的文件(后将省略)

优化率：0 %

1. **测试算例2:unsat-5cnf-30.cnf 测试结果如图**

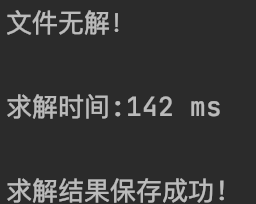
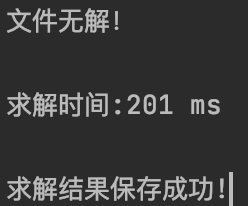


图4-12 测试结果

优化率：29.4%

1. **测试算例3:ais10.cnf 测试结果如图**

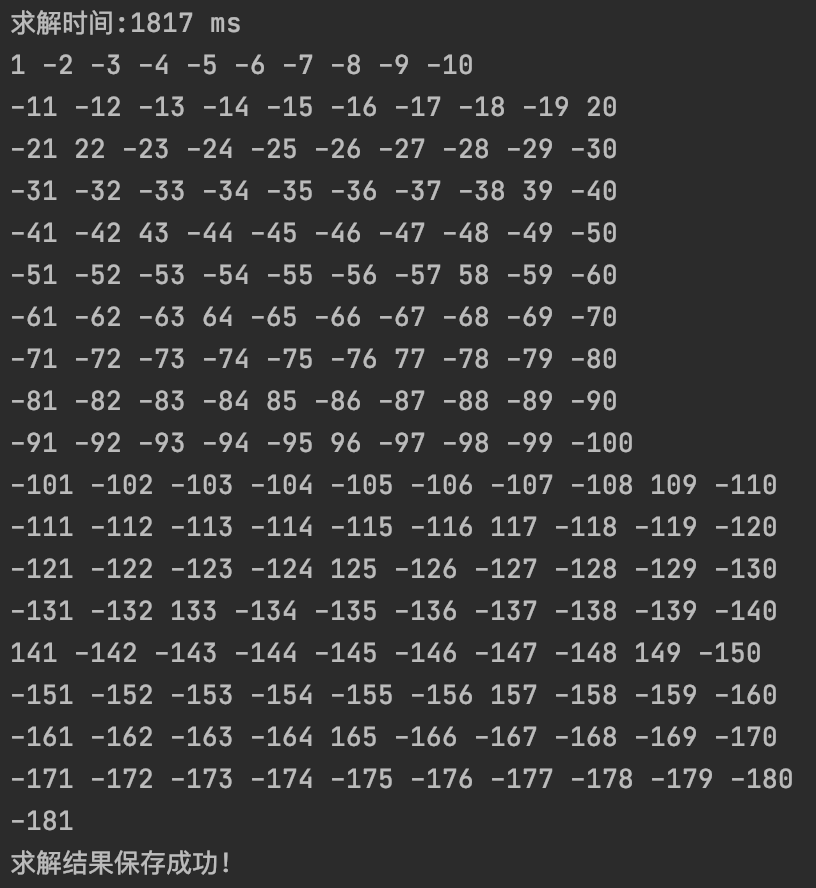
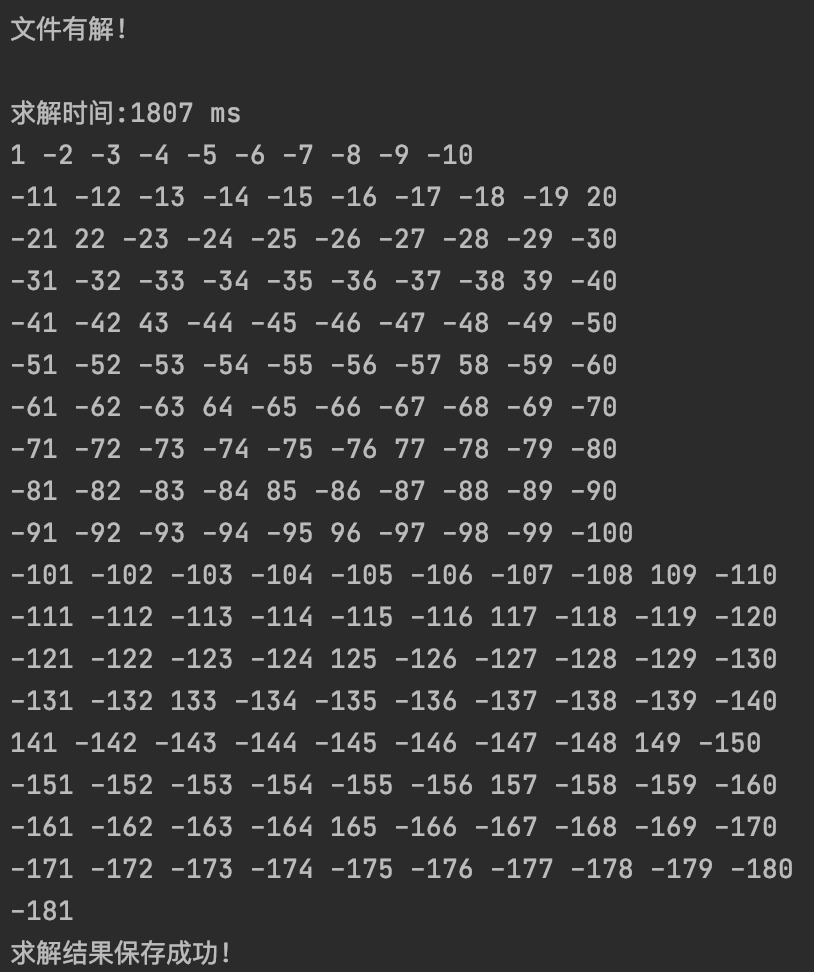


图4-13 测试结果

优化率：0.55%

1. **测试算例4:sud00009.cnf 测试结果如图**

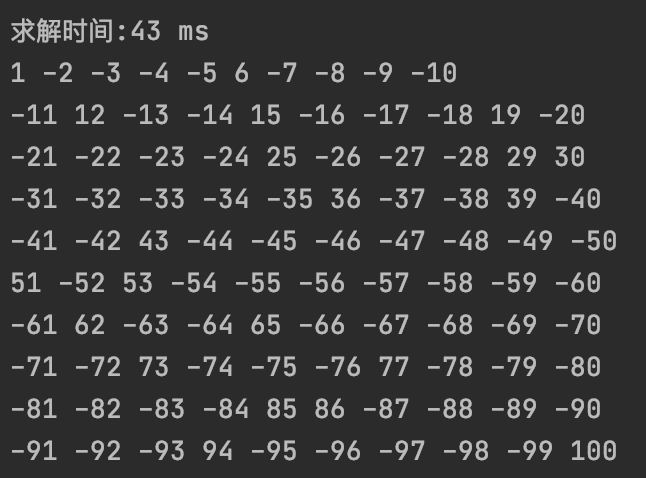
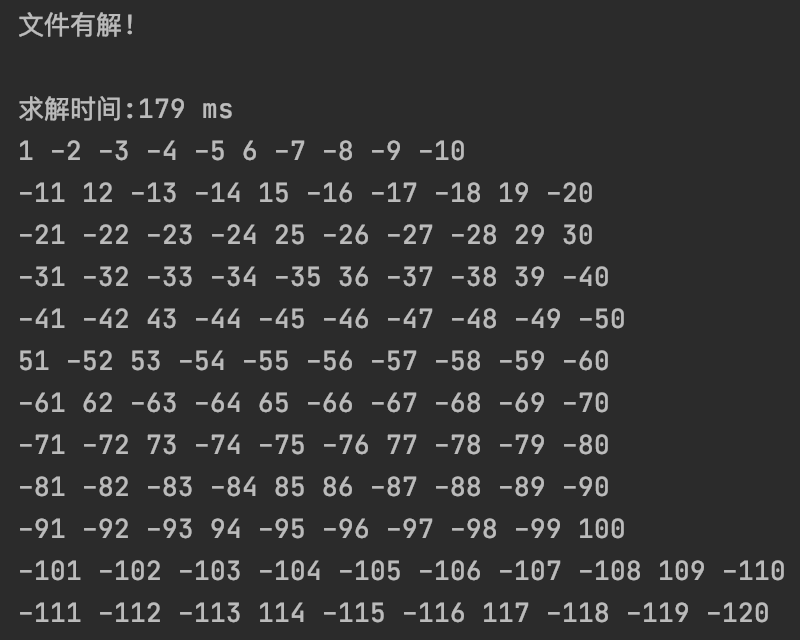


图 4-14 测试结果

优化率：75.98%

1. **测试算例5: 2.cnf(助教给的12个样例中较大的) 测试结果如图**

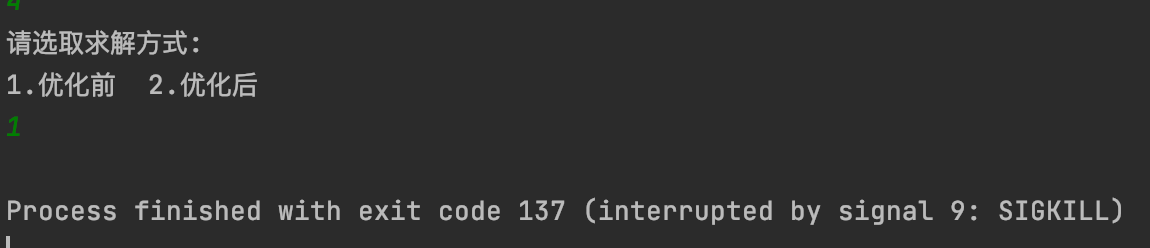


图4-15 优化前因占用大量内存程序自动被关闭

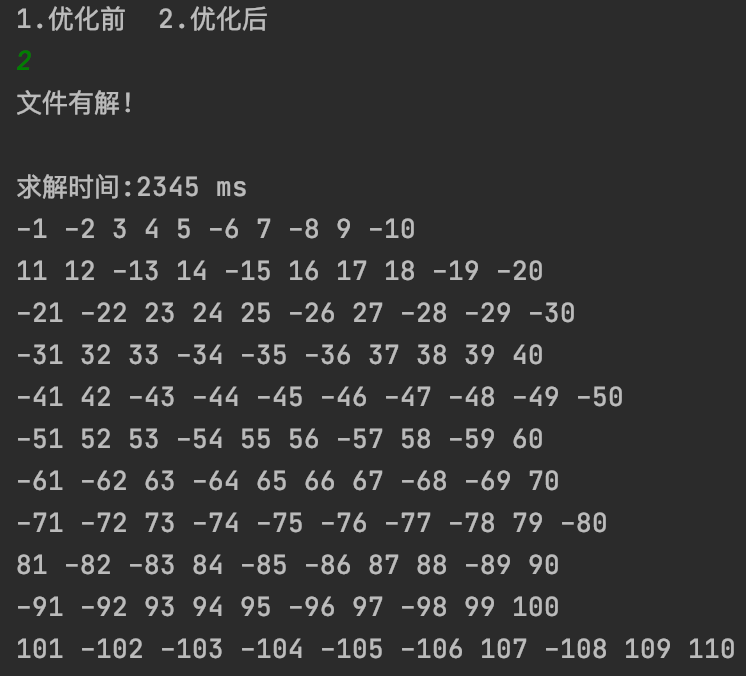


图4-16 优化后仅用2s

优化率: 无法计算。

**6.测试算例6: problem11-100.cnf 测试结果如图**

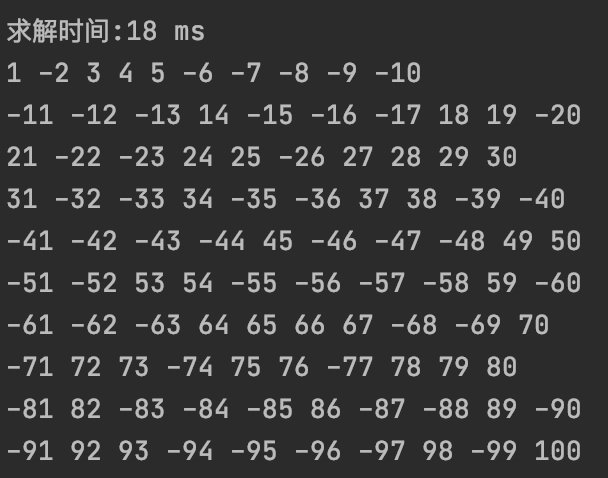
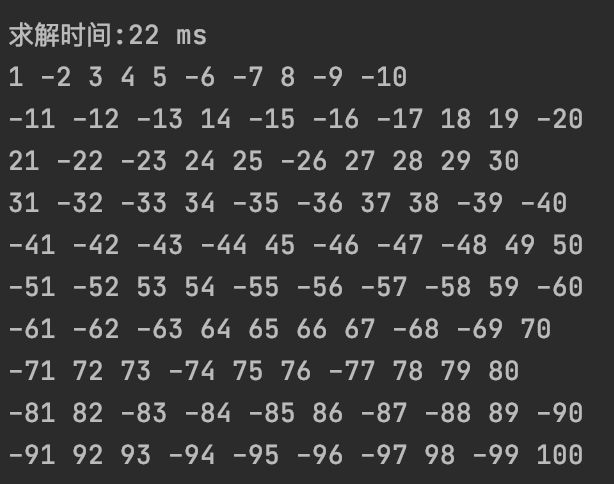


图 4-17 测试结果

优化率：18.2%

1. **测试算例7: sud00079.cnf 测试结果如图**

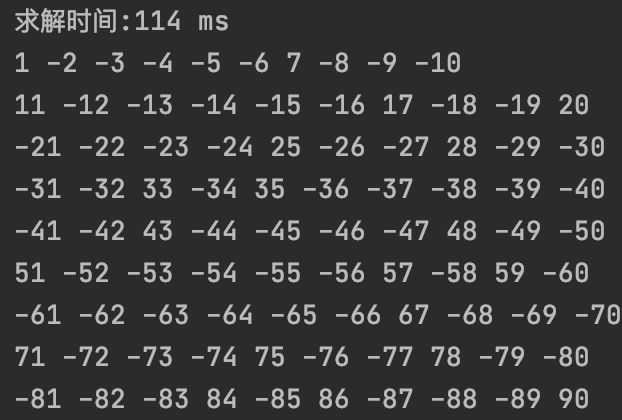
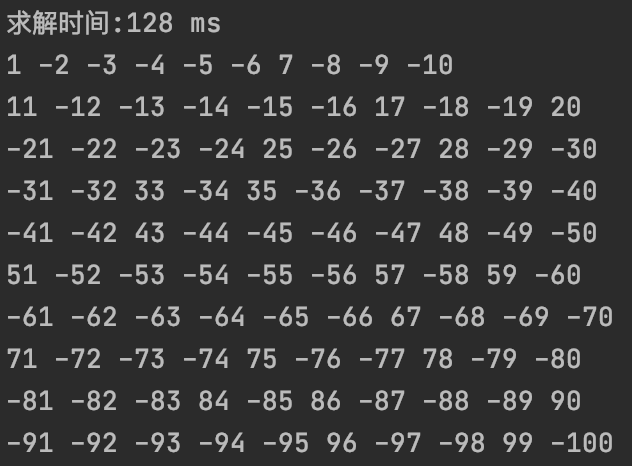


图 4-18 测试结果

优化率：10.94%

**8.测试算例8: u-5cnf\_3500\_3500\_30f1.resffled-30.cnf 测试结果如图**

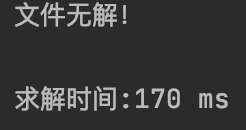
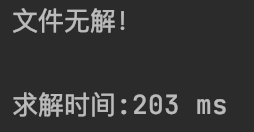


图 4-19 测试结果

优化率: 16.26%

后面省略 都整合到下面的表格中

##### 4.2.4 CNF测试算例总结表格

表3 cnf算例测试总结

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算例名 | 变元数 | 子句数 | 子句数/变元数 | 求解结果 | 求解时间  (优化前/后) | 优化率 |
| Sat-20.cnf | 20 | 91 | 4.55 | 满足 | 0ms / 0ms | 0% |
| unsat-5cnf-30.cnf | 30 | 420 | 14 | 不满足 | 201ms /142ms | 29.4% |
| Ais10.cnf | 181 | 3151 | 17.4 | 满足 | 1817ms/1807ms | 0.55% |
| Sud00009.cnf | 303 | 2851 | 9.41 | 满足 | 179ms/43ms | 75.98% |
| 2.cnf | 1075 | 3151 | 2.93 | 满足 | NULL/2345ms | NULL |
| sud00079.cnf | 301 | 2810 | 9.34 | 满足 | 128ms/114ms | 10.84% |
| problem11-100.cnf | 100 | 600 | 6 | 满足 | 22ms/18ms | 18.2% |
| u-5cnf\_3500\_3500\_30f1.resffled-30.cnf | 30 | 420 | 14 | 不满足 | 203ms/170ms | 16.26% |
| sud00012.cnf | 232 | 1901 | 8.19 | 满足 | 151ms/56ms | 62.9% |
| sud00861.cnf | 297 | 2721 | 9.16 | 满足 | 优化前：64ms  优化后：63ms | 1.56% |
| u-problem10-100.cnf | 100 | 200 | 2 | 不满足 | 优化前：77ms  优化后：62ms | 19.48% |
| tst\CBS\_k3\_n100\_m403\_b10\_0.cnf | 100 | 403 | 4.03 | 满足 | 优化前：2572ms  优化后：1652ms | 35.77% |
| tst\qg7-09.cnf | 729 | 22060 | 30.26 | 满足 | 优化前：701ms  优化后：668ms | 4.71% |
| tst\bw\_large.a.cnf | 459 | 4675 | 10.19 | 满足 | 优化前：3562ms  优化后：2745ms | 22.94% |

##### **4.2.5 二进制数独模块测试**

首先简介一下二进制数独部分：第一步是从文件里读取初始棋盘，第二步可以选择直接转化CNF文件求解，或者是先PLAY填充之后再转化为CNF文件，也可以PLAY一半后选择直接求解，，第三步就是Dpll求解出结果或者是验证结果。

1. **初始读取棋盘并展示**

预期结果：输出读取成功并展示

实际结果：如图所示

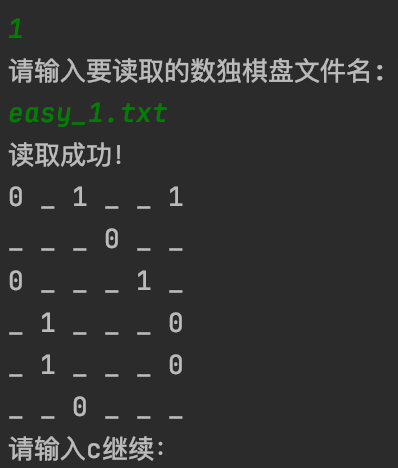


图4-20 读取数独文件

1. **转化为CNF文件求解**

实际结果：如图所示



图4-21 转化为CNF文件

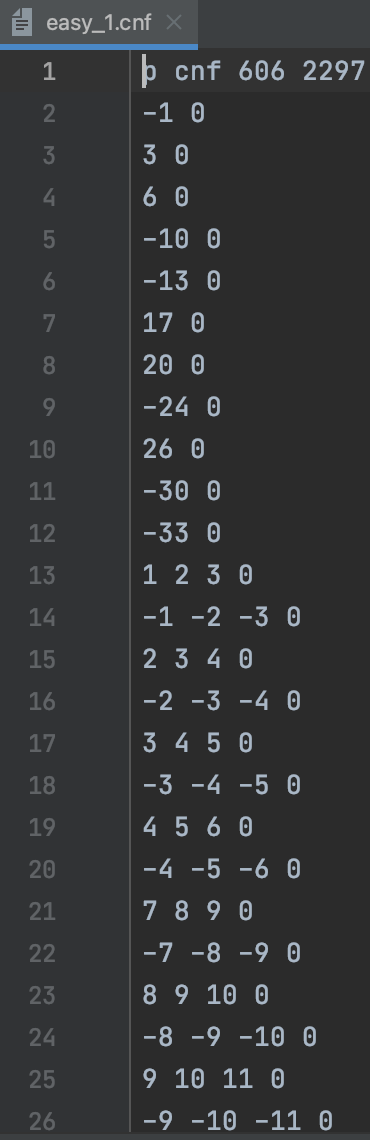


图4-22 查看CNF文件

3. **DPLL求解并展示**

如图所示



图4-23 DPLL求解

3. PLAY交互展示

选择PLAY后，会有棋盘展示和操作说明

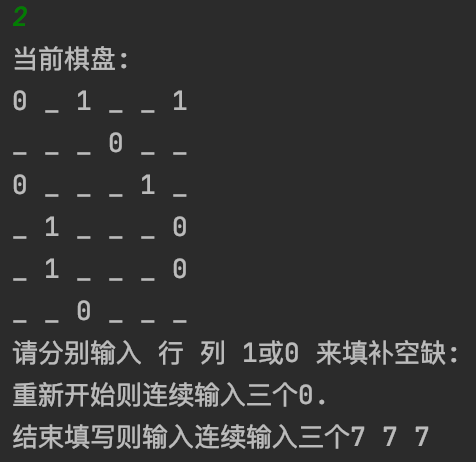


图4-24 操作说明

如果选择在初始位置填空，会有相应报错信息提示不能进行填空

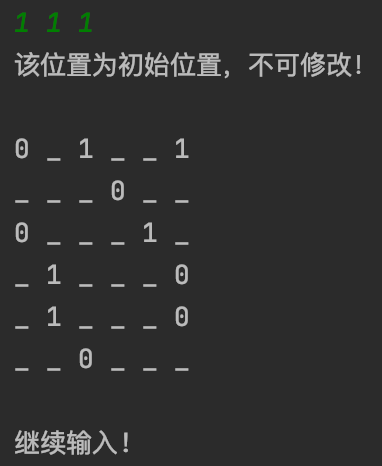


图4-25 提示信息

正常游玩界面



图4-26 正常游玩界面

1. PLAY未填满的报错信息

预计结果：如果棋盘未填满就验证是否正确，会提示棋盘未填满

实际结果：如图所示

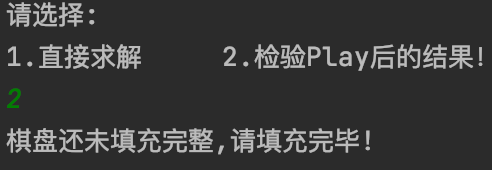


图4-27 未填满提示信息

1. 随机生成棋盘供游玩

使用了6个模板棋盘作为随机生成棋盘的素材来源，用户每一次选择随机生成都是随机选择一个棋盘并随机挖空，确保了棋盘的多样性。演示如图：

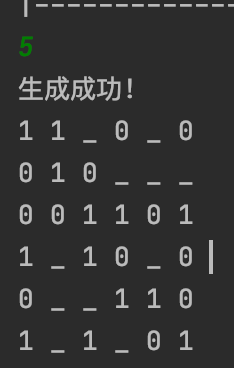


图4-28 随机生成棋盘游玩

##### 4.2.6 用Qt实现的数独GUI



图 4-29 初始界面

当点击生成数独就会生成一个随机数独界面



图 4-30 数独界面

点击初始空格，即已经有数字的地方，无任何变化(即不能改变)

当点击需要填空的格子时，会按照 空->1->0->空 的顺序变化 具有较强的可玩性

比如我此时随机的选取了几个空格子进行游玩



图4-31 游玩过程

点击展示结果，就会展示最后结果，如果为空或者有错就会以红色显示

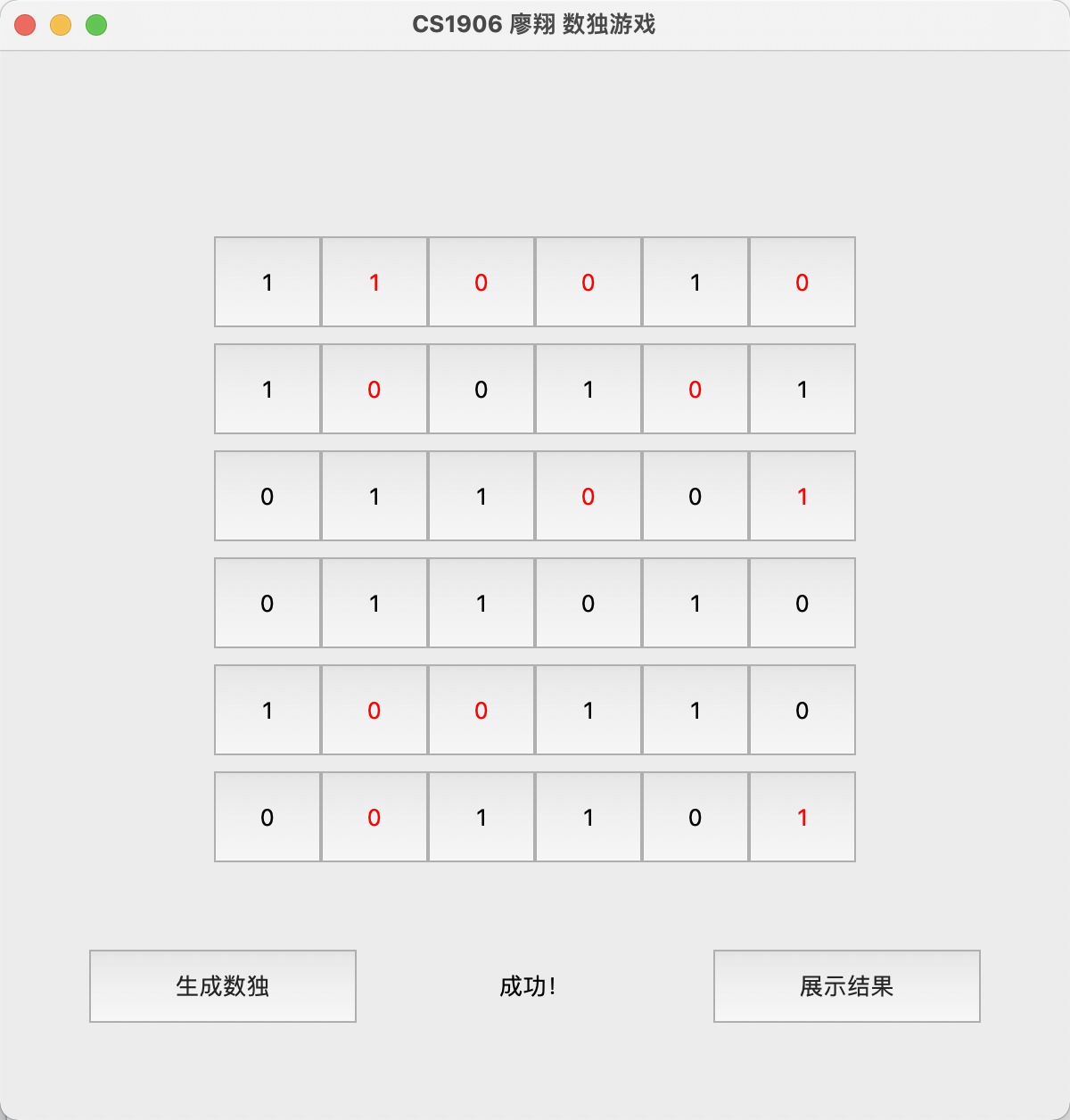


图4-32 展示结果

**5.总结与展望**

**5.1 全文总结**

对自己所作的工作做一个总结，如下：

1. 实现了一个基于DPLL算法的SAT求解器
2. 在选词策略和判断是否满足方式上对求解器做了一定优化
3. 完成了二进制游戏的初始化，CNF文件的转化，归约于DPLL求解，并具有一定的可玩性和交互性
4. 使用Qt实现了数独游戏的GUI，显著改善了用户体验，交互性和可玩性也更强
5. 设计了测试方案并完成了多个算例的测试

**5.2工作展望**

在今后的研究工作中，围绕这几个方面展开：

1. 本次SAT求解器测试的最大样例为1075个文字和3152个公式，对于一些large的样例还无法求解或者会使系统自动终止进程，这表明该系统性能还不足够好，还有很大的优化空间，在今后的研究工作中，希望能通过更多的项目和编程经验，学习更多的优化策略来优化这一系统，使之能够求解更加复杂的算例
2. 在实现DPLL算法的过程中最先使用的是回溯算法，例如将删除的单子句恢复等等这样的操作，最后发现设计出的数据结构会非常复杂，由于空间换时间的缘故，求解时间确实会大大缩短，但是结果会出现某些变元缺失的情况，最后更改了存储结构为任务书上的之后问题得到了解决。在今后的工作中，希望能接触到更能体现回溯思想的DPLL算法
3. 棋盘生成选择的是指导书上的挖洞法，没有考虑通过其他的一些回溯约束策略来生成只有唯一解的棋盘，因此有一些缺陷，在后续工作中希望能够完善这一功能
4. 虽然在二进制数独中使用了Qt来实现交互界面，相较于终端命令行的提示确实在美观性和交互性上提升了许多，但是还是有一些不足，例如可以增加更多功能例如8\*8数独，以及按钮字体等等，这些都是可以在后续工作中优化的内容
5. **体会**

本次课程设计是第一次学校课程需要的工作量较大的个人项目，在整个项目实施过程中，也更加熟悉了C语言以及一些基本的算法思想如回溯递归等等，总体来说本次项目是一次收获颇丰的课程设计，但是当然在整个过程中也碰到了一些问题，例如性能不太够，求解时间太慢等等，按照课设实施时间顺序列出一些体会如下：

1. 首先要搞清楚题目。老师在上学期期末考试后将课程设计指导书发给我们，两个题目初看过去都不太好理解，沉下心来认真厘清课设要求之后选择第一个作为这次的课题。
2. 在二进制数独棋盘的构造过程中，没有考虑通过约束算法来生成棋盘，而是使用在现有的棋盘上挖洞来随机生成棋盘，因此没有考虑到唯一解的问题。
3. 整合头文件和函数实现文件。刚开始没有太考虑引用关系，胡乱地将函数声明和实现丢在不同文件中，引起了许多错误，花了一些时间重新规范这些内容，也为以后的工作提供了一点教训
4. 多动手，多查资料，多思考才是做出课设的王道。

**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]TanbirAhmed.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]CarstenSinz.Visualizing SAT Instances and Runsof the DPLL Algorithm.JAutom Reasoning (2007) 39:219–243

[5] Binary Puzzle：<http://www.binarypuzzle.com/>

[6] Putranto H. Utomo and Rusydi H. Makarim. Solving a Binary Puzzle. Mathematics in Computer Science,(2017) 11:515–526

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

[8]InsLynce and JolOuaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

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

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

<http://zhangroup.aporc.org/images/files/Paper_3485.pdf>

**附录**

(1)define.h

//

// Created by JustinXiang on 2021/2/26.

//

#ifndef DSCOURSE\_DESIGN\_DEFINE\_H

#define DSCOURSE\_DESIGN\_DEFINE\_H

//头文件

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<time.h>

//状态码

typedef int status;

#define OK 1

#define ERROR 0

#define Untrue -1

//全局变量定义

int word\_number, formula\_number; //文字数 公式数

int \*variable; //用于判别文字的真值

int i; //循环变量

int temp; //中间(临时)变量

int Way; //选择求解方式

int nums[2000];

//存储结构

struct Literal{

struct Literal \*next;

int value; //文字值

};

struct Clause{

struct Clause \* next;

struct Literal \* head;

};

struct Qipan{

int flag; //用于标记是否为初始棋盘已确定的位置

int num;

};

struct Qipan qipan[36]; //用于保存6阶棋盘信息

char filename\_setcnf[15]; //用于保存数独转化cnf文件名

/\*---------------CNF文件处理------------------\*/

/\*单个结点初始化函数\*/

/\*新建子句\*/

struct Clause \* createClause();

/\*新建文字\*/

struct Literal \* createLiteral();

/\*cnf文件读取\*/

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

/\*遍历输出CNF文件公式\*/

void Traversefile(struct Clause \* root);

/\*CNF文件保存\*/

status Savefile(struct Clause \* root);

/\*-----------------DPLL算法-----------------\*/

/\*辅助函数，判断正负\*/

int judgesign(int a);

/\*判断单子句，返回单子句里的文字\*/

int isUnitClause(struct Clause \* root);

/\*处理单子句\*/

status removeClause(struct Clause \* root);

/\*判断只有一种出现形式的文字\*/

int isPureLiteral(struct Clause \* root);

/\*处理只有一种出现形式的文字\*/

status removePureLiteral(struct Clause \* root);

/\*选取假设真假文字\*/

int chooseLiteral(struct Clause \* root);

/\*克隆子句\*/

struct Clause \* cloneClause(struct Clause \* headp);

/\*克隆全部结构并将选取的文字设为单子句加入存储结构中\*/

struct Clause \* cloneAll(int a,struct Clause \* root);

/\*销毁存储结构\*/

void destroyClauses(struct Clause \* root);

/\*判断是否文字均为单值\*/

status isAllPureLiteral(struct Clause \* root);

/\*判断是否存在空子句\*/

status isEmptyClause(struct Clause \* root);

/\*检查当前是否满足\*/

status check(struct Clause \* root);

/\*dpll算法\*/

status dpll(struct Clause \* root);

/\*将结果写入同名文件\*/

void writeResult(char \* filename, int flag, int time);

/\*-----------------BinaryPuzzle-----------------\*/

/\*读取棋盘文件\*/

void Readqipan(char \* filename);

/\*初始化棋盘\*/

void Resetqipan(struct Qipan qipan[]);

/\*输出当前棋盘\*/

void Showqipan(struct Qipan qipan[]);

/\*数独交互\*/

void PlaySuDoKu(struct Qipan qipan[]);

/\*检查棋盘是否填满\*/

status Checkqipan(struct Qipan qipan[]);

/\*转化为cnf文件\*/

void SetCNFfile(struct Qipan qipan[]);

/\*随机生成数独\*/

void RandGener(char \* filename);

#endif //DSCOURSE\_DESIGN\_DEFINE\_H

1. CNFfileparser.c

//

// Created by JustinXiang on 2021/2/26.

//

#include "define.h"

/\*单个结点初始化函数\*/

/\*新建子句\*/

struct Clause \* createClause(){

struct Clause \* new\_clause = malloc(sizeof(struct Clause));

new\_clause->head = NULL;

new\_clause->next = NULL;

return new\_clause;

}

/\*新建文字\*/

struct Literal \* createLiteral()

{

struct Literal \* new\_literal = malloc(sizeof(struct Literal));

new\_literal->next = NULL;

new\_literal->value = 0;

return new\_literal;

}

/\*cnf文件读取\*/

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

int length = strlen(filename);

//printf("%d",length);

if(filename[length-3]!='c' && filename[length-2]!='n' && filename[length-1]!='f')

{

printf("文件格式不正确\n");

exit(1);

}

FILE \* fp;

fp = fopen(filename,"r");

if(fp == NULL){

printf("文件打开失败!\n");

exit(1);

}

char s[100];

char \*c;

struct Clause \* root = NULL, \* current\_Clause = NULL, \* pre\_Clause = NULL;

struct Literal \* current\_Literal = NULL, \* pre\_Literal = NULL;

while(fgets(s, 100, fp)){//按行读取

if(s[0] == 'c') continue;

else if(s[0] == 'p'){

sscanf(s, "p cnf %d %d", &word\_number, &formula\_number);//按格式读入文字数与公式数

printf("文字数: %d 公式数: %d\n", word\_number, formula\_number);

variable = (int \*)calloc(word\_number + 1, sizeof(int));

for(i = 1; i <= word\_number; i++) variable[i] = -1;//初始化标记数组，-1表示未被确定真值

}

else{

//构建存储结构

current\_Clause = createClause();

if(root == NULL) root = current\_Clause;

if(pre\_Clause != NULL) pre\_Clause->next = current\_Clause;

c = strtok(s, (const char \*) " ");//分割字符串

while(c != NULL){

current\_Literal = createLiteral();

temp = atoi(c);

current\_Literal->value = temp;

if(temp != 0){

nums[abs(temp)-1] +=1;

if(current\_Clause->head == NULL) current\_Clause->head = current\_Literal;

if(pre\_Literal != NULL) pre\_Literal->next = current\_Literal;

}

pre\_Literal = current\_Literal;

c = strtok(NULL, (const char \*) " ");

}

pre\_Clause = current\_Clause;

}

}

fclose(fp);

return root;

}

/\*遍历输出CNF文件格式\*/

void Traversefile(struct Clause \* root)

{

struct Clause \* T = root;

while(T!=NULL){

struct Literal \* current\_Literal = T->head;

printf("公式: ");

while(current\_Literal != NULL)

{

printf("%d ",current\_Literal->value);

current\_Literal = current\_Literal->next;

}

if(T->next!=NULL) printf("\n");

T = T->next;

}

printf("文件输出成功!\n");

}

/\*文件保存成功\*/

status Savefile(struct Clause \* root)

{

FILE \* fp;

char filename[20];

struct Clause \* T = root;

printf("请输入文件名：\n");

scanf("%s",filename);

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

printf("文件打开失败！\n");

return ERROR;

}

else{

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

fprintf(fp,"%d ",word\_number);

fprintf(fp,"%d\n",formula\_number);

while(T){

struct Literal \* current\_Literal = T->head;

while(current\_Literal){

fprintf(fp,"%d ",current\_Literal->value);

current\_Literal = current\_Literal->next;

}

if(T->next!=NULL) fprintf(fp,"0\n");

T = T->next;

}

fprintf(fp,"0");

}

fclose(fp);

return OK;

}

1. BinaryPuzzle.c

//

// Created by JustinXiang on 2021/2/27.

//

#include "define.h"

int j, k ,z, final, add, num1, num2, r, l; //r为行 l为列

/\*读取棋盘文件\*/

void Readqipan(char \* filename)

{

FILE \* fp;

fp = fopen(filename,"r");

if(fp == NULL)

{

printf("文件打开失败!\n");

}

else

{

int integer; //用于接受数字

i = 0;

while(fscanf(fp,"%d",&integer) != EOF)

{

qipan[i].num = integer;

if(integer !=-1 ) qipan[i].flag = 1;

else qipan[i].flag = 0;

i++;

}

}

fclose(fp);

printf("读取成功！\n");

}

/\*随机生成数独\*/

void RandGener(char \* filename)

{

FILE \* fp;

fp = fopen(filename,"r");

if(fp == NULL)

{

printf("文件打开失败");

}

else{

int integer;

srand((unsigned)time(NULL));

int x =11+rand()%11;

int j;

i = 0;

while(fscanf(fp,"%d",&integer) != EOF)

{

qipan[i].num = integer;

qipan[i].flag = 1;

i++;

}

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

{

int r = rand()%6;

int c = rand()%6;

qipan[r\*6+c].num = -1;

qipan[r\*6+c].flag = 0;

}

}

fclose(fp);

printf("生成成功!\n");

}

/\*输出当前棋盘\*/

void Showqipan(struct Qipan qipan[])

{

for(i=0;i<36;i++)

{

if(qipan[i].num == -1) printf("\_ ");

else printf("%d ",qipan[i].num);

if((i+1)%6 == 0) putchar('\n');

}

}

/\*初始化棋盘\*/

void Resetqipan(struct Qipan qipan[])

{

for(i=0;i<36;i++)

{

if(qipan[i].flag != 1) qipan[i].num = -1;

}

}

/\*数独交互\*/

void PlaySuDoKu(struct Qipan qipan[])

{

printf("当前棋盘:\n");

Showqipan(qipan);

printf("请分别输入 行 列 1或0 来填补空缺:\n");

printf("重新开始则连续输入三个0.\n结束填写则输入连续输入三个7 7 7\n");

int answer[3] = {-1};

while(scanf("%d",&answer[0]) != EOF)

{

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

{

scanf("%d",&answer[i]);

}

if(answer[0]==0 && answer[1]==0 && answer[2] == 0)

{

Resetqipan(qipan);

printf("重新开始!\n");

}

else if(answer[0]==7 && answer[1]==7 && answer[2] == 7)

{

printf("结束!\n");

return;

}

else

{

temp = (answer[0]-1)\*6 + answer[1];

if(qipan[temp-1].flag == 1){

printf("该位置为初始位置，不可修改！\n");

}

else

{

qipan[temp-1].num = answer[2];

}

}

putchar('\n');

Showqipan(qipan);

printf("\n继续输入!\n");

}

}

/\*检查棋盘是否填满\*/

status Checkqipan(struct Qipan qipan[])

{

for(i=0;i<36;i++)

{

if(qipan[i].num == -1) return ERROR;

}

return OK;

}

/\*转化为CNF文件\*/

void SetCNFfile(struct Qipan qipan[])

{

printf("输入要转化为CNF文件的文件名:\n");

scanf("%s",filename\_setcnf);

FILE \* fp;

fp = fopen(filename\_setcnf,"w");

if(fp == NULL)

{

printf("文件打开失败!\n");

exit(1);

}

int f\_sum = 0; //公式数计数器

for(i = 0; i <36;i++)

{

if(qipan[i].num!= -1) f\_sum++;

}

fprintf(fp,"p cnf 606 %d\n",f\_sum+2286);

/\*---------------------------------------\*/

//根据棋盘信息

/\*---------------------------------------\*/

for(i = 0; i<36;i++)

{

if(qipan[i].num == 1)

{

fprintf(fp,"%d 0\n",i+1);

}

else if(qipan[i].num == 0)

{

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

}

continue;

}

/\*---------------------------------------\*/

//约束1：行列中无连续的0或1

/\*---------------------------------------\*/

//处理6行

for(r = 1;r<=6;r++)

{

for(l = 1;l<=4;l++)

{

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

{

temp = (r-1)\*6 + l +i;

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

if(i==2){

fprintf(fp,"0\n");

}

}

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

{

temp = (r-1)\*6 + l + i;

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

if(i==2)

{

fprintf(fp,"0\n");

}

}

}

}

//处理6列

for(l = 1; l<=6;l++)

{

for(r = 1;r<=4;r++)

{

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

{

temp = (r+i-1)\*6 + l;

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

if(i==2)

fprintf(fp,"0\n");

}

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

{

temp = (r+i-1) \*6 +l;

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

if(i==2)

fprintf(fp,"0\n");

}

}

}

/\*---------------------------------------\*/

//约束2：行，列中0，1个数相同

/\*---------------------------------------\*/

int order[16][5] = {{3,4,5,6},{2,4,5,6},{2,3,5,6},{2,3,4,6},{2,3,4,5}

,{1,4,5,6},{1,3,5,6},{1,3,4,6},{1,3,4,5}

,{1,2,5,6},{1,2,4,6},{1,2,4,5}

,{1,2,3,6},{1,2,3,5}

,{1,2,3,4}};

//处理6行

for (r = 1; r <= 6; r++){

for (k = 0; k < 15; k++){

for(j = 0; j < 4; j++){

temp = (r - 1)\*6 + order[k][j];

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

if (j == 3) fprintf(fp, "0\n");

}

}

}

for (r = 1; r <= 6; r++){

for (k = 0; k < 15; k++){

for(j = 0; j < 4; j++){

temp = (r - 1)\*6 + order[k][j];

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

if (j == 3) fprintf(fp, "0\n");

}

}

}

//处理6列

for (l = 1; l <= 6; l++){

for (k = 0; k < 15; k++){

for(j = 0; j < 4; j++){

temp = (order[k][j] - 1)\*6 + l;

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

if (j == 3) fprintf(fp, "0\n");

}

}

}

for (l = 1; l <= 6; l++){

for (k = 0; k < 15; k++){

for(j = 0; j < 4; j++){

temp = (order[k][j] - 1)\*6 + l;

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

if (j == 3) fprintf(fp, "0\n");

}

}

}

/\*---------------------------------------\*/

//约束3：没有相同的行或列

/\*---------------------------------------\*/

add = 36;

for(i=1;i<=6;i++)

{

for(j = i+1;j<=6;j++)

{

final = add +19;

for(k=1;k<=6;k++)

{

num1 = (i-1)\*6 + k;

num2 = (j-1)\*6 + k;

fprintf(fp,"%d -%d 0\n%d -%d 0\n-%d -%d %d 0\n", num1,add+1,num2,add+1,num1,num2,add+1);

fprintf(fp,"-%d -%d 0\n-%d -%d 0\n%d %d %d 0\n", num1,add+2,num2,add+2,num1,num2,add+2);

fprintf(fp,"-%d %d 0\n-%d %d 0\n%d %d -%d 0\n", add+1,add+3,add+2,add+3,add+1,add+2,add+3);

add+=3;

}

for(z = 1;z<=6;z++)

{

fprintf(fp,"%d %d 0\n",final-3\*(z-1)-1,final);

}

fprintf(fp,"-%d -%d -%d -%d -%d -%d -%d 0\n",final-16,final-13,final-10,final-7,final-4,final-1,final);

add = final;

}

}

//处理6列

for (i = 1; i <= 6; i++){

for (j = i + 1; j <= 6; j++){

final = add + 19;

for (k = 1; k<= 6; k++){

num1 = (k - 1)\*6 + i;

num2 = (k - 1)\*6 + j;

fprintf(fp,"%d -%d 0\n%d -%d 0\n-%d -%d %d 0\n", num1,add+1,num2,add+1,num1,num2,add+1);

fprintf(fp,"-%d -%d 0\n-%d -%d 0\n%d %d %d 0\n", num1,add+2,num2,add+2,num1,num2,add+2);

fprintf(fp,"-%d %d 0\n-%d %d 0\n%d %d -%d 0\n", add+1,add+3,add+2,add+3,add+1,add+2,add+3);

add+=3;

}

for (z = 1; z <= 6; z++){

fprintf(fp,"%d %d 0\n", final - 3\*(z - 1) - 1,final);

}

fprintf(fp,"-%d -%d -%d -%d -%d -%d -%d 0\n",final-16,final-13,final-10,final-7,final-4,final-1,final);

add = final;

}

}

fclose(fp);

printf("文字数：606 公式数：%d\n", f\_sum + 2286);

}

1. Dpllsolver.c

//

// Created by JustinXiang on 2021/2/27.

//

#include "define.h"

/\*判断单子句，返回单子句里的文字\*/

int isUnitClause(struct Clause \* root)

{

struct Clause \* T = root;

while(T!=NULL){

if(T->head == NULL) continue;

if(T->head->next==NULL) return T->head->value;

T = T->next;

}

return 0;

}

/\*处理单子句\*/

status removeClause(struct Clause \* root){

temp = isUnitClause(root);

if(temp==0) return ERROR; //表示没有单子句

variable[abs(temp)] = temp >0?1:0;

struct Clause \* T = root,\*pre\_T = createClause();

while(T!=NULL){

struct Literal \* current\_Literal = T->head, \* pre\_Literal = createLiteral();

while(current\_Literal!=NULL){

if(current\_Literal->value == temp){

//删除包含该文字的子句

if(T==root){

\*(root) = \*(root->next);

T = NULL;

}

else{

pre\_T->next = T->next;

T = pre\_T;

}

break;

}

else if (current\_Literal->value == -temp){ //删除互为相反数的文字

if(current\_Literal == T->head) T->head = current\_Literal->next;

else pre\_Literal->next = current\_Literal->next;

current\_Literal = pre\_Literal;

continue;

}

pre\_Literal = current\_Literal;

current\_Literal = current\_Literal->next;

}

pre\_T = T;

T = (T==NULL) ?root :T->next;

}

return OK;

}

/\*判断正负\*/

int judgesign(int a){

return a>0?1:-1;

}

/\*判断只有一种出现形式的文字，返回文字的值\*/

int isPureLiteral(struct Clause \* root){

int \* mark = (int \*)calloc(word\_number+1,sizeof(int)); //标志数组，初始化0;

struct Clause \* T = root;

while(T!=NULL){

struct Literal \* current\_Literal = T->head;

while(current\_Literal != NULL){

if(mark[abs(current\_Literal->value)]!=2){

temp = mark[abs(current\_Literal->value)];

if(temp==0) mark[abs(current\_Literal->value)] = judgesign(current\_Literal->value);

else if(temp ==1 && judgesign(current\_Literal->value) == -1) mark[abs(current\_Literal->value)] = 2;

else if(temp==-1 && judgesign(current\_Literal->value) == 1) mark[abs(current\_Literal->value)] = 2;

//有两种存在形式标记为2

}

current\_Literal = current\_Literal->next;

}

T = T->next;

}

for(i=1;i<=word\_number;i++){

if(mark[i] == -1 || mark[i] == 1) return i\*mark[i];

}

return 0; //没有以一种形式出现的文字

}

/\*处理只有一种出现形式的文字\*/

status removePureLiteral(struct Clause \* root)

{

temp = isPureLiteral(root);

if(temp == 0) return ERROR;

variable[abs(temp)] = temp >0?1:0;

struct Clause \* T = root,\* pre\_T = createClause();

while( T != NULL){

struct Literal \*current\_Literal = T->head;

while(current\_Literal != NULL){

if(current\_Literal->value == temp){

if(T == root){

\* root = \*(root->next);

T = NULL;

}

else{

pre\_T ->next = T->next;

T = pre\_T;

}

break;

}

current\_Literal = current\_Literal->next;

}

pre\_T = T;

T = (T == NULL) ?root:T->next;

}

return OK;

}

/\*选取假设真值文字\*/

int chooseLiteral(struct Clause \* root)

{

if(Way == 1)

{

return root->head->value;

}

else

{

int x,max = 0;

struct Literal \* l = root->head;

struct Clause \* c = root;

while(c->head == NULL)

{

c = c->next;

}

l = c ->head;

while(l)

{

if(nums[abs(l->value)-1]>max)

{

max = nums[abs(l->value)-1];

x = abs(l->value);

}

l = l->next;

}

return x;

}

}

/\*克隆子句\*/

struct Clause \* cloneClause(struct Clause \* re\_clause)

{

struct Clause \* clone\_Clause = createClause();

struct Literal \* current\_Literal = re\_clause->head, \* pre\_Literal = NULL;

while(current\_Literal != NULL){

struct Literal \* clone\_Literal = createLiteral();

clone\_Literal->value = current\_Literal->value;

if(clone\_Clause->head == NULL) clone\_Clause->head = clone\_Literal;

if(pre\_Literal != NULL) pre\_Literal->next = clone\_Literal;

pre\_Literal = clone\_Literal;

current\_Literal = current\_Literal->next;

}

return clone\_Clause;

}

/\*克隆全部结构并将选取的文字设为单子句加入存储结构中\*/

struct Clause \* cloneAll(int a, struct Clause \* root)

{

variable[abs(a)] = a >0?1:0;

struct Clause \* T = root, \* new\_root = NULL, \*pre\_Clause = NULL;

while(T != NULL){

struct Clause \* current\_Clause = cloneClause(T);

if(new\_root == NULL) new\_root = current\_Clause;

if(pre\_Clause != NULL) pre\_Clause->next = current\_Clause;

pre\_Clause = current\_Clause;

T = T->next;

}

struct Clause \* new\_Clause = createClause();

struct Literal \* new\_Literal = createLiteral();

new\_Literal->value = a;

new\_Clause->head = new\_Literal;

new\_Clause->next = new\_root;

return new\_Clause;

}

/\*销毁存储结构\*/

void destroyClauses(struct Clause \* root)

{

while(root != NULL){

struct Clause \* next\_r = root->next;

if(root->head != NULL){

struct Literal \* current\_Literal = root->head;

while(current\_Literal != NULL){

struct Literal \* next\_Literal = current\_Literal->next;

free(current\_Literal);

current\_Literal = next\_Literal;

}

}

free(root);

root = next\_r;

}

}

/\*判断文字是否均为单一形式出现\*/

status isAllPureLiteral(struct Clause \* root)

{

int \*mark = (int \*)calloc(word\_number+1,sizeof(int));

struct Clause \* T = root;

while(T != NULL)

{

struct Literal \* current\_Literal = T->head;

while(current\_Literal != NULL){

temp = mark[abs(current\_Literal->value)];

if(temp == 0) mark[abs(current\_Literal->value)] = judgesign(current\_Literal->value);

else if(temp != judgesign(current\_Literal->value))

{

free(mark);

return ERROR;

}

current\_Literal = current\_Literal->next;

}

T = T->next;

}

T = root;

while(T != NULL){

struct Literal \* current\_Literal = T->head;

while(current\_Literal != NULL){

if(variable[abs(current\_Literal->value)]==-1) variable[abs(current\_Literal->value)] = current\_Literal->value>0 ?1:0;

current\_Literal = current\_Literal->next;

}

T = T->next;

}

return OK;

}

/\*判断是否存在空子句\*/

status isEmptyClause(struct Clause \* root){

struct Clause \* T = root;

while(T != NULL){

if(T->head == NULL) return OK;

T = T->next;

}

return ERROR;

}

/\*检查当前是否满足\*/

status check(struct Clause \* root)

{

if(isEmptyClause(root)) return ERROR; //表示不满足

if(isAllPureLiteral(root)) return OK; //表示满足

return Untrue; //表示未确定

}

/\*dpll算法\*/

status dpll(struct Clause \* root){

temp = check(root);

if(temp != Untrue){

destroyClauses(root);

return temp;

}//不是未确定的情况，返回结果

while(1){//处理单子句

temp = check(root);

if(temp != Untrue){

destroyClauses(root);

return temp;

}//同上

if(!removeClause(root)) break;//处理单子句，没有的话跳出循环

}

int new\_count = chooseLiteral(root);//选取假设真值

if(dpll(cloneAll(new\_count, root)) == OK) return OK;

else{

temp = dpll(cloneAll(-new\_count, root));//如果不成功，选取假设真值的相反数

destroyClauses(root);

return temp;

}

}

/\*将结果写入同名文件\*/

void writeResult(char \* filename, int flag, int time)

{

int temp = 0;

for(i=0;i<200;i++){

if(filename[i] == '.')

{

filename[i+1] = 'r';

filename[i+2] = 'e';

filename[i+3] = 's';

break;

}

}

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

if(fp == NULL)

{

printf("求解结果保存失败!\n");

exit(1);

}

if(flag == 1)

{

fprintf(fp,"s 1\n");

fprintf(fp,"v ");

for(i=1; i<=word\_number;i++)

{

temp = (variable[i]==0 ?-1:1) \*i;

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

if(i%10 == 0) fprintf(fp,"\n");

}

fprintf(fp,"\nt %f ms\n",time);

}

else

{

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

fprintf(fp,"v \n");

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

}

printf("求解结果保存成功!\n");

fclose(fp);

}

1. main.c

#include "define.h"

int main(void)

{

for(i=0;i<2000;i++)

{

nums[i] = 0;

}

char filename[200], copy\_filename[200]; //cnf文件读取 与结果保存

char sudo\_filename[20]; //数独文件

int op = 1, SAT = 1, SuDoKu = 1; //用于switch分支

int flag\_cnf = 0; //用于判断cnf文件是否读取成功

int flag\_sudo = 0; //用于判断数独文件是否读取成功

int way\_sudo = 0; //sudo处理方式

struct Clause \* root;

while(op)

{

printf("|--------------------Choose SAT or BinaryPuzzle--------------------|\n");

printf("| |\n");

printf("| 1.SAT 2.BinaryPuzzle |\n");

printf("| |\n");

printf("| 0.Exit |\n");

printf("| |\n");

printf("| CopyRight @CS1906 LX |\n");

printf("| |\n");

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

printf("| |\n");

printf("| Make your choice ( 0 - 2 ) |\n");

printf("| |\n");

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

scanf("%d",&op);

switch(op)

{

case 1:

while(SAT){

printf("|-------------------------- Menu For SAT --------------------------|\n");

printf("| |\n");

printf("| 1.Readfile 2.Savefile |\n");

printf("| |\n");

printf("| 3.Traversefile 4.Dpll |\n");

printf("| |\n");

printf("| 0.Exit |\n");

printf("| |\n");

printf("| CopyRight @CS1906 LX |\n");

printf("| |\n");

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

printf("| |\n");

printf("| Make your choice ( 0 - 4 ) |\n");

printf("| |\n");

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

scanf("%d",&SAT);

switch(SAT) {

case 1:

for(i=0;i<2000;i++)

{

nums[i] = 0;

}

printf("请输入读取的cnf文件名:\n");

scanf("%s", filename);

strcpy(copy\_filename, filename);

root = Readfile(filename);

if (root != NULL) {

printf("文件读取成功\n");

flag\_cnf = 1;

} else {

printf("文件读取失败！\n");

}

getchar();

printf("请输入c继续：\n");

getchar();

break;

case 2: {

if (flag\_cnf) {

if (Savefile(root)) {

printf("文件保存成功！\n");

} else {

printf("文件保存失败！\n");

}

} else printf("文件还未读取！\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

case 3:

{

if(flag\_cnf) Traversefile(root);

else printf("文件还未读取！\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

case 4:

{

if(flag\_cnf)

{

printf("请选取求解方式:\n");

printf("1.优化前 2.优化后\n");

scanf("%d",&Way);

clock\_t start,end;

start = clock();

if(dpll(root)==OK)

{

end = clock();

printf("文件有解!\n");

printf(" \n");

printf("求解时间:%d ms\n",(end-start)\*1000/CLOCKS\_PER\_SEC);

for(i=1;i<=word\_number;i++)

{

temp = (variable[i] ==0 ?-1:1)\*i;

printf("%d ",temp);

if(i%10==0) putchar('\n');

}

putchar('\n');

writeResult(copy\_filename,1,(end-start)\*1000/CLOCKS\_PER\_SEC);

}

else

{

end = clock();

printf("文件无解！\n");

printf(" \n");

printf("求解时间:%d ms\n",(end-start)\*1000/CLOCKS\_PER\_SEC);

putchar('\n');

writeResult(copy\_filename,0,(end-start)\*1000/CLOCKS\_PER\_SEC);

}

flag\_cnf = 0;

//destroyClauses(root);

}

else{

printf("文件还未读取!\n");

}

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

case 0:{

break;

}

default:

{

printf("输入错误！\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

}

}

getchar();

printf("请输入c继续：\n");

getchar();

break;

case 2:

while(SuDoKu)

{

//system("cls");

printf("|--------------------- Menu For BinaryPuzzle --------------------|\n");

printf("| |\n");

printf("| 1.ReadSudoKuFile 2.Play |\n");

printf("| |\n");

printf("| 3.SetCNFfile 4.Dpll For Answer Check |\n");

printf("| |\n");

printf("| 5.Randomly Generate 0.Exit |\n");

printf("| |\n");

printf("| CopyRight @CS1906 LX |\n");

printf("| |\n");

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

printf("| |\n");

printf("| Make your choice ( 0 - 4 ) |\n");

printf("| |\n");

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

scanf("%d",&SuDoKu);

switch(SuDoKu)

{

case 1:

printf("请输入要读取的数独棋盘文件名:\n");

scanf("%s",sudo\_filename);

Readqipan(sudo\_filename);

Showqipan(qipan);

flag\_sudo = 1;

getchar();

printf("请输入c继续：\n");

getchar();

break;

case 2:

if(flag\_sudo) PlaySuDoKu(qipan);

else printf("数独文件还未读取!\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

case 3:

if(flag\_sudo)

{

printf("请选择:\n1.直接求解 2.检验Play后的结果！\n");

scanf("%d",&way\_sudo);

if(way\_sudo==1)

{

Resetqipan(qipan);

SetCNFfile(qipan);

}

else

{

if(Checkqipan(qipan)) SetCNFfile(qipan);

else printf("棋盘还未填充完整,请填充完毕!\n");

}

}

else printf("数独文件还未读取!\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

case 4:

if(flag\_sudo){

char filename\_sudocopy[15];

strcpy(filename\_sudocopy,filename\_setcnf);

struct Clause \* root\_sudo = Readfile(filename\_sudocopy);

clock\_t t1,t2;

t1 = clock();

if(dpll(root\_sudo) == OK){

t2 = clock();

char file\_sudoresult[15];

strcpy(file\_sudoresult,filename\_sudocopy);

if(way\_sudo == 1){

printf("Dpll求解成功！结果为:\n");

for(i = 0 ; i<36; i++)

{

printf("%d ", variable[i+1]);

if((i+1) % 6 == 0) putchar('\n');

}

}

else

{

printf("Dpll求解成功，结果正确!\n");

writeResult(file\_sudoresult,1,t2-t1);

//destroyClauses(root\_sudo);

}

}

else printf("Dpll求解失败,结果错误!\n可继续尝试！\n");

}

else printf("数独文件还未读取！\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

case 5: {

char \*random\_filename[]={"StanSudo\_1.txt","StanSudo\_2.txt","StanSudo\_3.txt","StanSudo\_4.txt","StanSudo\_5.txt"};

srand((unsigned)time(NULL));

int z =rand()%5;

RandGener(random\_filename[z]);

Showqipan(qipan);

flag\_sudo = 1;

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

case 0:

break;

default:

printf("输入错误!\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

}

case 0:

break;

default:

printf("输入错误！\n");

getchar();

printf("请输入c继续：\n");

getchar();

break;

}

}

printf("下次再见！\n");

return 0;

}