

**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： CS1804**

**学 号： U201814596**

**姓 名： 左振**

**指导教师： 袁凌**

**报告日期： 2020年4月3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%)

目录

[1引言 4](#_Toc4119)

[1.1课题背景与意义 4](#_Toc28332)

[1.1.1 SAT 4](#_Toc12478)

[1.1.2 二进制数独 4](#_Toc4362)

[1.2国内外研究现状 5](#_Toc31952)

[1.3课程设计的主要研究工作 6](#_Toc21233)

[2系统需求分析与总体设计 7](#_Toc1504)

[2.1系统需求分析 7](#_Toc2060)

[2.2系统总体设计 7](#_Toc16331)

[3系统详细设计 9](#_Toc28728)

[3.1有关数据结构的定义 9](#_Toc24738)

[3.2主要算法设计 11](#_Toc12354)

[3.2.1 CNF文件处理 11](#_Toc11514)

[3.2.2 Dpll算法处理 11](#_Toc18813)

[3.2.3 Dpll优化方案 15](#_Toc31648)

[3.2.4二进制数独处理 15](#_Toc20259)

[4系统实现与测试 18](#_Toc21506)

[4.1系统实现 18](#_Toc20382)

[4.1.1软硬件环境 18](#_Toc25806)

[4.1.2数据类型定义 18](#_Toc22822)

[4.1.3函数声明以及函数调用关系 19](#_Toc15506)

[4.2系统测试 22](#_Toc31841)

[4.2.1交互系统展示 22](#_Toc15465)

[4.2.2 CNF文件解析模块测试 22](#_Toc1250)

[4.2.3 DPLL算法模块测试 23](#_Toc15604)

[4.2.4 CNF算例测试总结表格 35](#_Toc7865)

[4.2.5二进制数独模块测试 37](#_Toc24622)

[5总结与展望 41](#_Toc6516)

[5.1全文总结 41](#_Toc7906)

[5.1工作展望 41](#_Toc15429)

[6体会 42](#_Toc22217)

[参考文献 44](#_Toc31778)

[附录 45](#_Toc1690)

# 1引言

### 1.1课题背景与意义

#### 1.1.1 SAT

SAT问题即布尔可满足性问题，是确定是否存在满足给定布尔公式解释的问题。如果对于给定的布尔公式变量可以一致地用TRUE或者FALSE替换，那么该布尔公式可满足，相反则不满足。

SAT问题是计算机科学与人工智能领域的经典问题，研究成果广泛应用于电子设计自动化，人工智能等领域，因此研究SAT问题有助于拓展知识面以及今后的实际应用。

#### 1.1.2 二进制数独

数独起源于18世纪瑞士数学家欧拉等人研究的拉丁方阵，是一个经典游戏。而二进制数独则是数独游戏演化的一个分支，基本规则也与传统的数独不同。二进制数独的基本规则为行列中没有连续的三个1或0，每行每列1与0的数目相同，没有相同的行或列。

研究二进制数独归约为基于Dpll的SAT求解器问题有利于更深化理解SAT求解器的应用，也有利于理解数独问题的一种特殊求解方法，同时有利于提升自己的综合编程能力，不是像在学习C语言和数据结构的时候一样仅仅局限于语法和算法上，而是有做一定小型项目的能力与经验。

### 1.2国内外研究现状

求解SAT问题的经典算法——Dpll算法，它在1962年由马丁·戴维斯、希拉里·普特南、乔治·洛吉曼和多纳·洛夫兰德共同提出，作为早期戴维斯-普特南算法的一种改进。戴维斯-普特南算法是戴维斯与普特南在1960年发展的一种算法。DPLL是一种高效的程序，并且经过40多年还是最有效的SAT解法，以及很多一阶逻辑的自动定理证明的基础。

在之后Bart Selman和Henry Kautz分别于1997年和2003年在人工智能第五届国际合作会议上提出了SAT问题面临的十大挑战性问题，并在2001年和2007年先后 对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出对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。这使得越来越多的人开始关注并研究SAT问题，所以这段时间也涌现出了众多新的高效的SAT算法如MINISAT、SATO、CHAFF、POSIT和GRASP等，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年梁东敏提出了改进的子句加权WSAT算法，2000年金人超和黄文奇提出了并行Solar算法，2002年张德富提出了模拟退火算法。

SAT国际竞赛从2002年开始每隔一到两年举办一次，这也极大地推动了SAT问题的研究，由此可见SAT求解问题仍在继续被人们所探索。

### 1.3课程设计的主要研究工作

1.首先对DPLL算法，SAT求解问题的背景，原理进行深入了解，根据相关资料对于项目做一个整体的设计；

2.设计相应数据结构与算法来完成基于DPLL的SAT求解器的实现，并用提供的算例做相应测试；

3.从改变存储结构或者选取文字策略等方面来实现算法的优化，设计测试方案来总结优化的效果；

4.设计问题转化策略将二进制数独问题归约为SAT问题并求解；

5.将各个分支整合为项目并调试直到能正确运行。

## 2系统需求分析与总体设计

### 2.1系统需求分析

基于DPLL算法的SAT求解器可以求解部分布尔算例，可以对CNF文件进行解析和结果输出；二进制数独有一定的交互性，可以用来求解和尝试可视化自主求解二进制数独游戏。

### 2.2系统总体设计

系统总体设计分为两个大的模块：基于DPLL算法的SAT求解器和二进制数独，各自模块下面还有一些小的功能，大致介绍如下：

1. 基于DPLL算法的SAT求解器，能够完成如下功能：

（1）CNF的读取解析，遍历输出，保存；

（2）DPLL求解，计算求解时间并显示，将结果保存到同名.res文件里。

2. 二进制数独，能够完成如下功能：

（1）数独原始棋盘文件读取构造；

（2）PLAY交互系统；

（3）将原始棋盘或者PLAY交互后的棋盘转化为CNF文件；

（4）DPLL求解或者进行结果验证。

系统总体设计流程图如下图1所示：

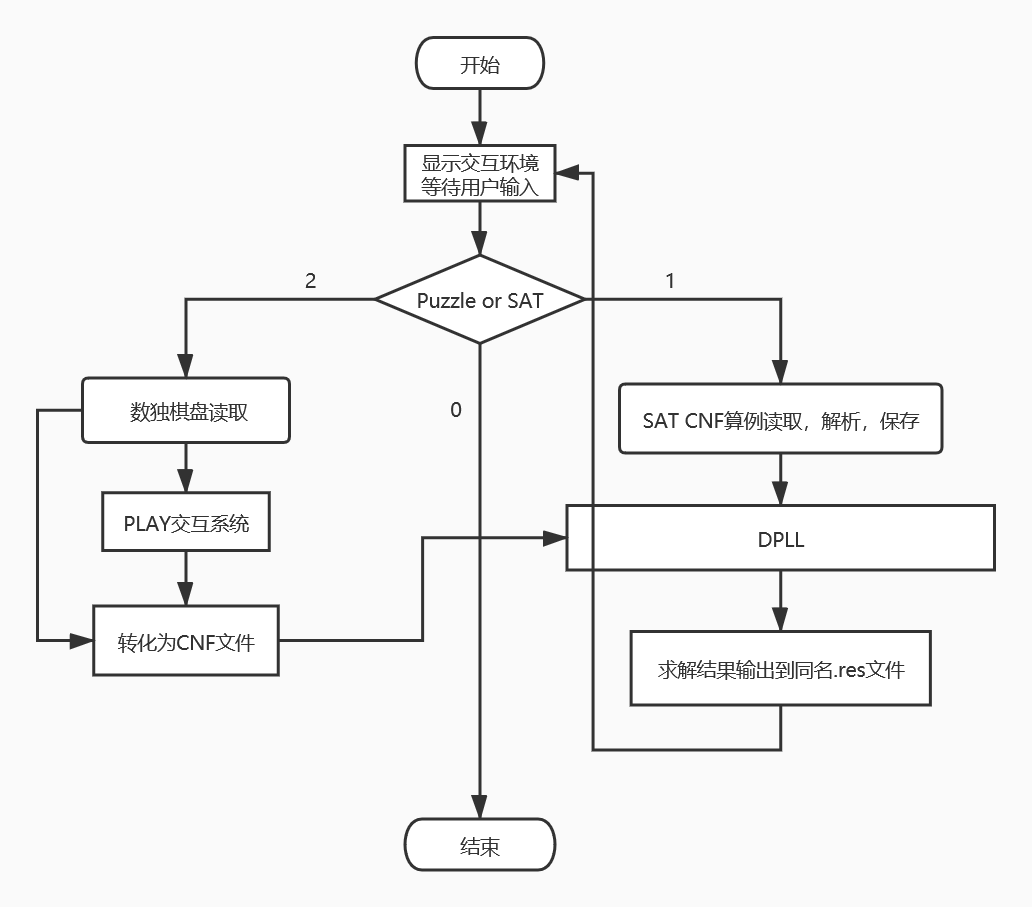


图1系统总体设计

## 3系统详细设计

### 3.1有关数据结构的定义

1.首先是要对CNF文件进行读取和解析，所以就要处理CNF文件里的文字和公式，对于文字我们可以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；对于公式我们也是用一个结构体来定义的，数据项有文字结构型的指针用来指向公式里的第一个文字，还有一个公式结构型的指针用来指向下一个公式；还有一个作为表示该文字bool值的int型指针，如表1所示：

表1cnf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 |

关于数据结构的图直观表示，如图2表示：

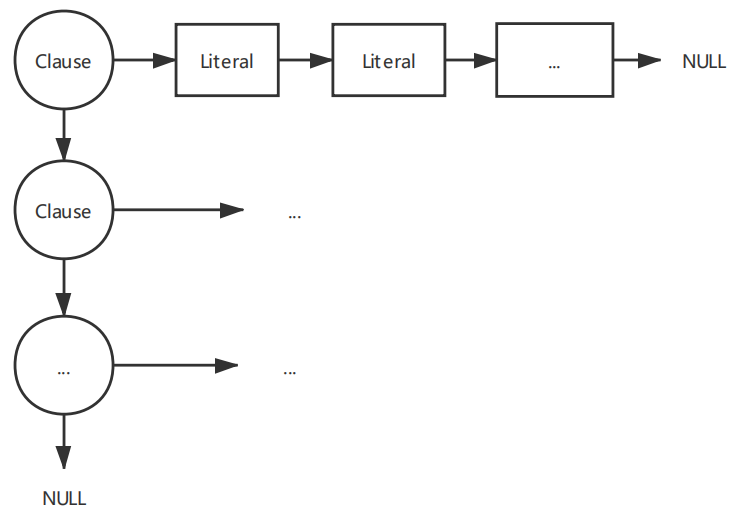


图2cnf文件文字与公式数据结构直观图

2.再就是实现二进制数独的时候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型的数据用来保存棋盘该位置放置的是0或1或者暂时空缺，如表2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

关键的点就是利用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

关键点就是利用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

关键点是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），如图3所示：

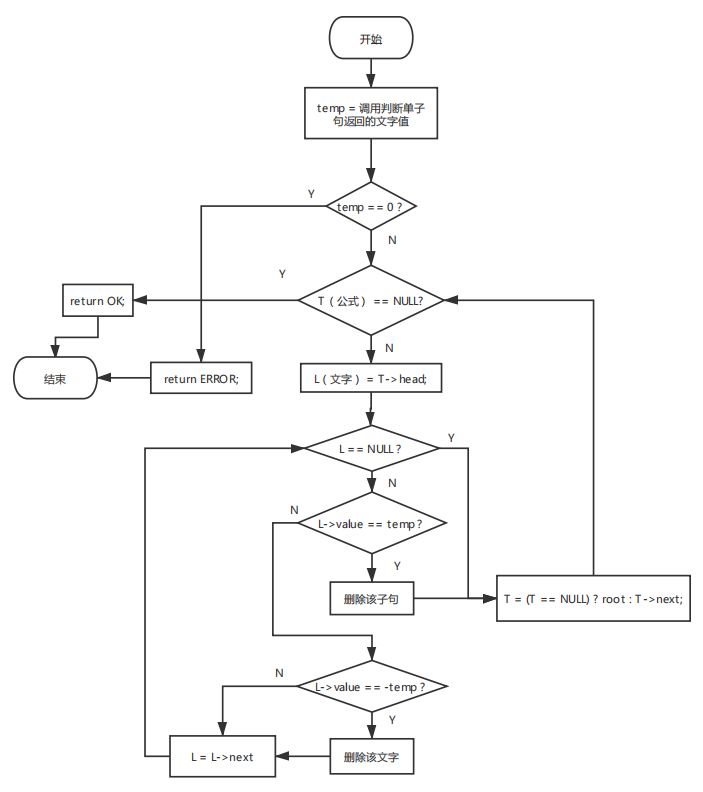


图3 处理单子句

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：

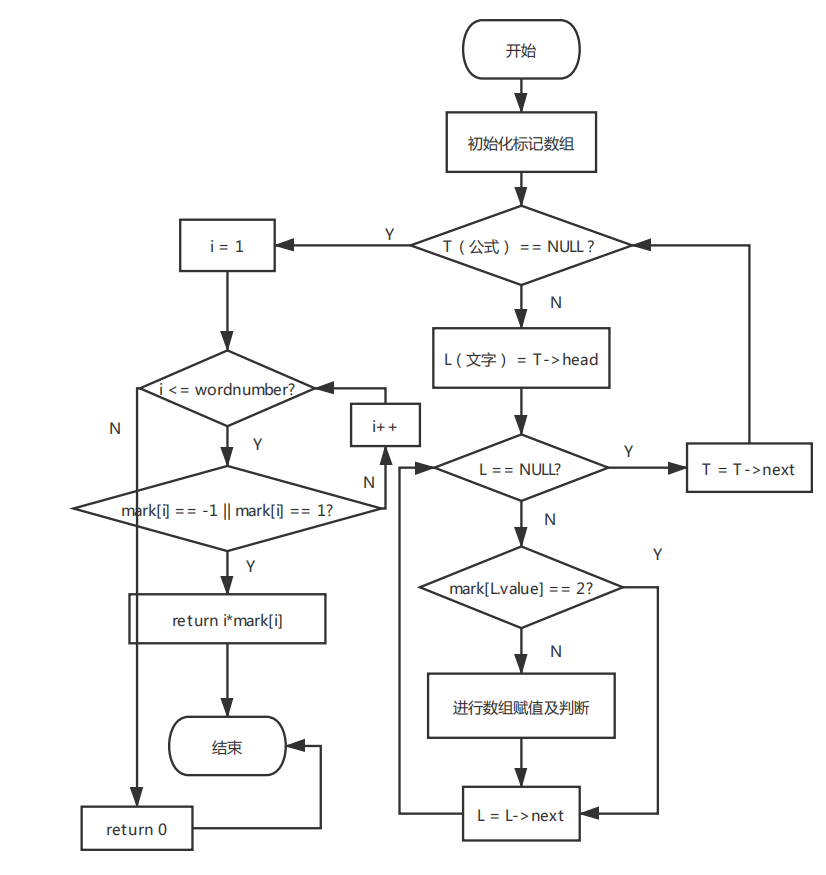


图4寻找只有一种出现形式的文字

1. 处理只有一种出线形式的文字

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优化方案

前面的Dpll算法处理已经提到了优化的方案，就是在于文字的选取策略和判断最后是否满足这两方面上，主要还是文字的选取方案。

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。

#### 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5所示：

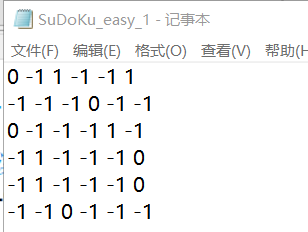


图5 DIY棋盘文件

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：Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz

机带RAM：8.00GB

系统类型：64位操作系统，基于x64的处理器

1. 软件环境：

Windows10下Dev-C++ 5.11

#### 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6所示：

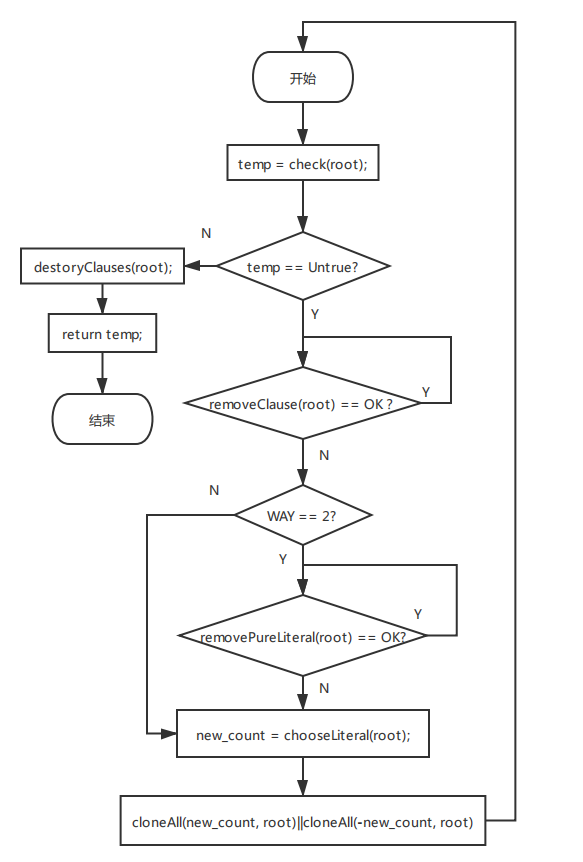


图6 Dpll函数调用关系

### 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，二进制数独界面展示，如下图7，8，9所示：

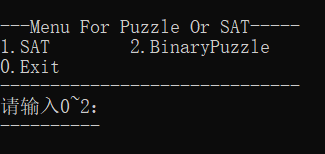


图7 主交互界面

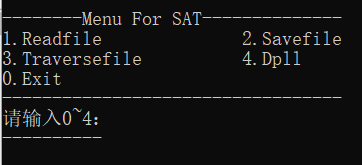


图8 SAT界面

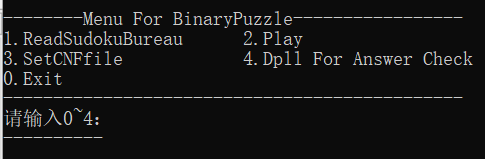


图9 二进制数独界面

#### 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，如图10所示：

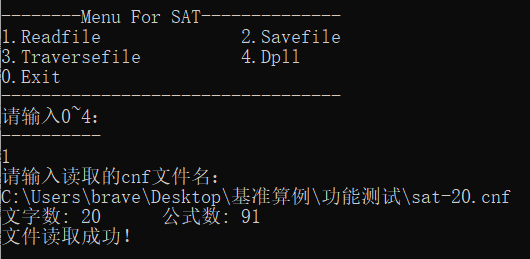


图10 cnf文件读取

2.cnf文件的保存，预计结果：输出保存成功提示，打开文件并验证，如图11，12所示：

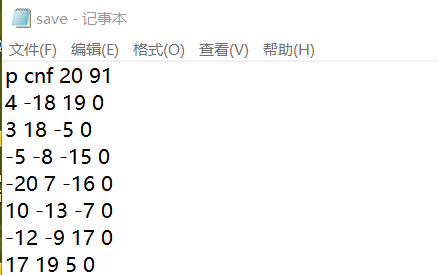
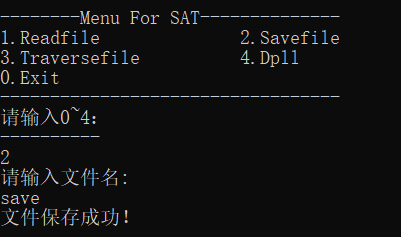


图11 cnf文件保存 图12 保存后的文件

1. cnf公式解析与验证，输出结果跟源文件对比后完全一致，如图13所示：

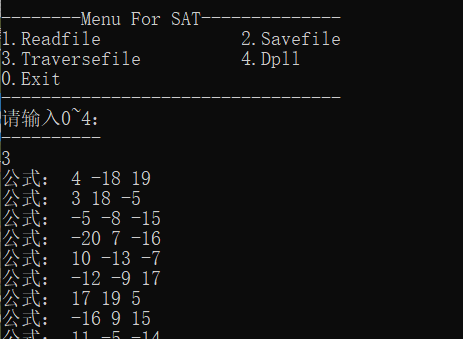


图13 cnf文件解析与验证

#### 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：基准算例\功能测试\sat-20.cnf，测试结果如图14，15所示：

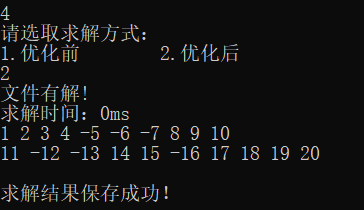
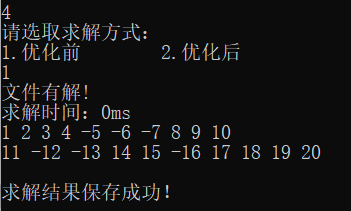


图14 优化前 图15 优化后

优化率：0

保存结果的文件，如图16所示：

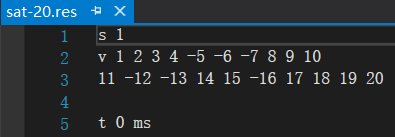


图16 保存结果的文件

1. 测试算例2：基准算例\功能测试\unsat-5cnf-30.cnf，测试结果如17，18所示：

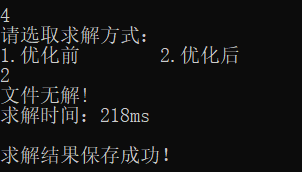
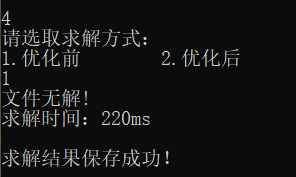


图17 优化前 图18 优化后

优化率：[(220-218)/220]\*100%=0.91%

保存结果的文件，如图19所示：

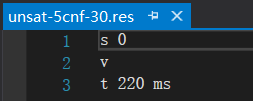


图19 保存结果的文件

提示：接下来保存结果的文件将省略。

1. 测试算例3：基准算例\性能测试\ais10.cnf，测试结果如图20，21所示：

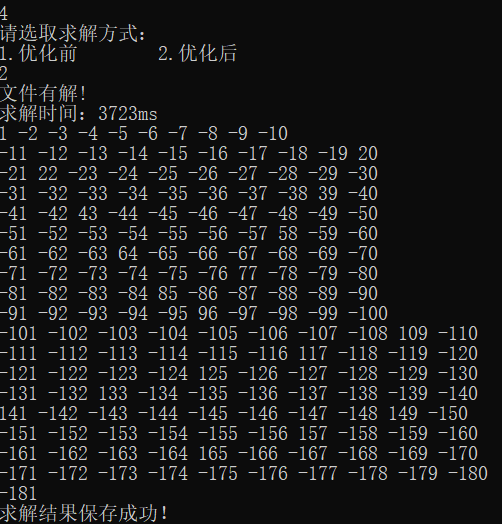
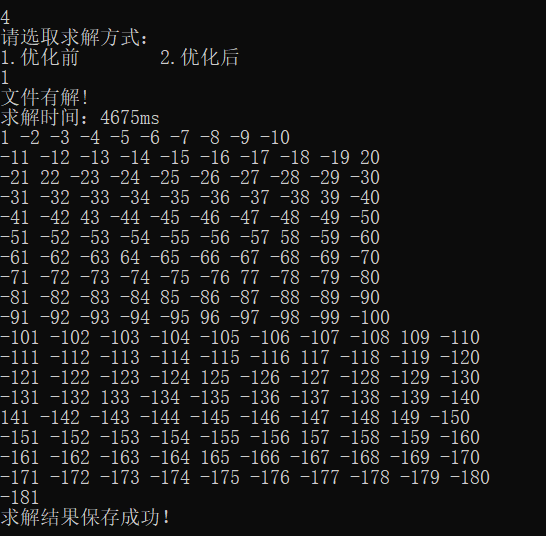


图20 优化前 图21 优化后

优化率：[(4675-3723)/4675]\*100%=20.36%

1. 测试算例4：基准算例\性能测试\sud00009.cnf，测试结果如图22，23所示：

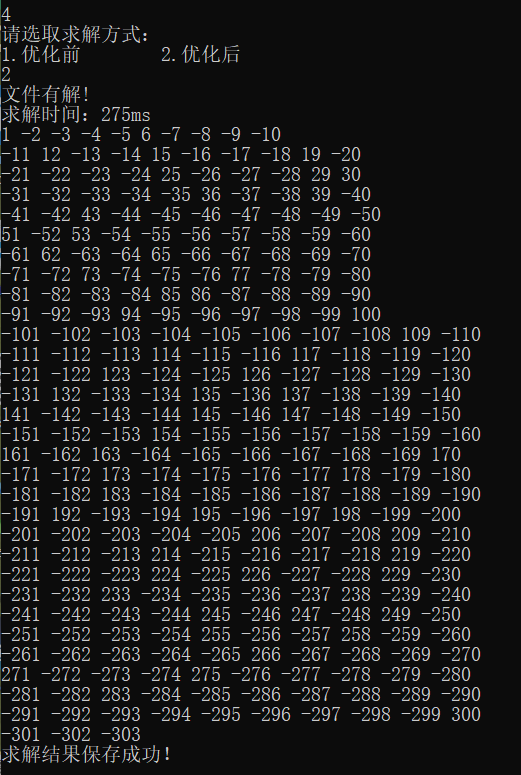
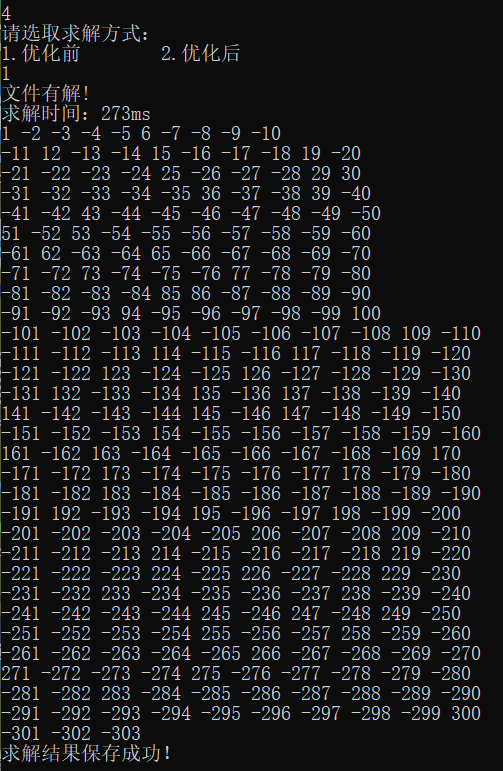


图22 优化前 图23优化后

优化率：[(273-275)/273]\*100%=-0.73%

提示：以上为基准算例四个算例的求解，下面为自己挑选的部分算例测试结果，最后会总结成表格的形式。

1. 测试算例5：满足算例\S\problem11-100.cnf，测试结果如图24，25所示：

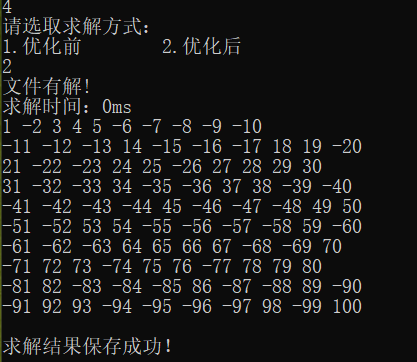
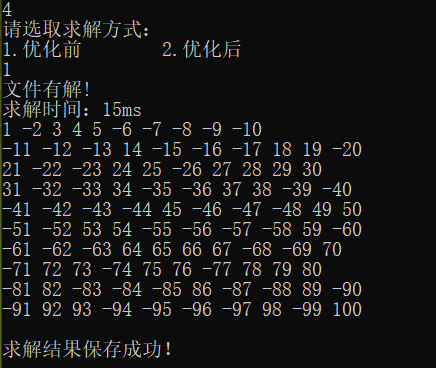


图24 优化前 图25优化后

优化率：[(15-0)/15]\*100%=100%

1. 测试算例6：满足算例\M\bart17.shuffled-231.cnf，测试结果如图26，27所示：

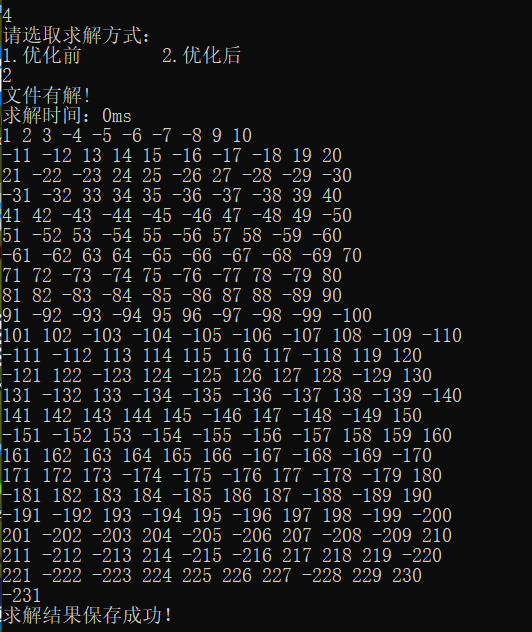
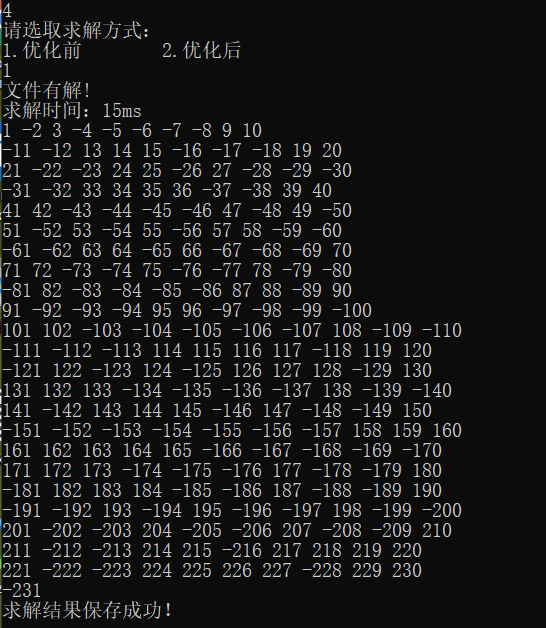


图26 优化前 图27 优化后

优化率：[(15-0)/15]\*100%=100%

1. 测试算例7：满足算例\M\problem-5-200.cnf，测试结果如图28，29所示：

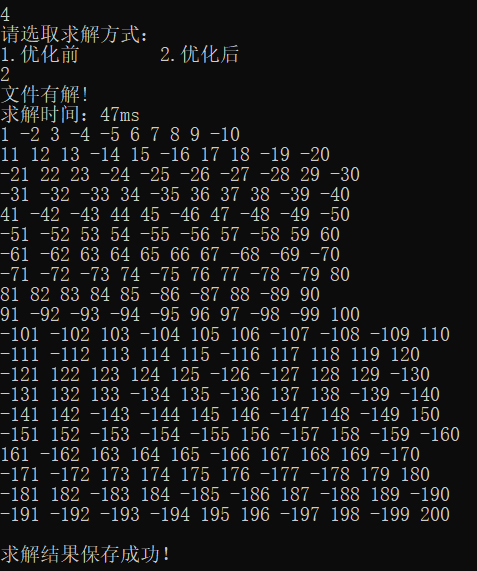
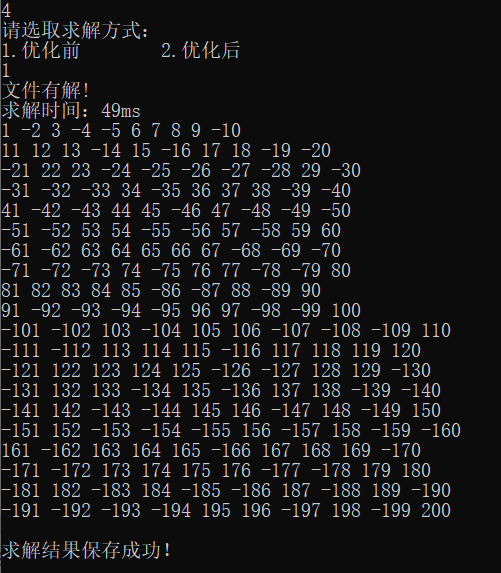


图28 优化前 图29 优化后

优化率：[(49-47)/49]\*100%=4.08%

1. 测试算例8：满足算例\M\problem-12-200.cnf，测试结果如图30，31所示：

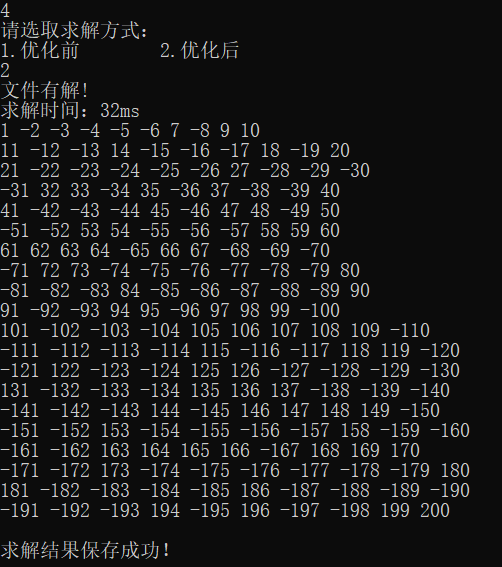
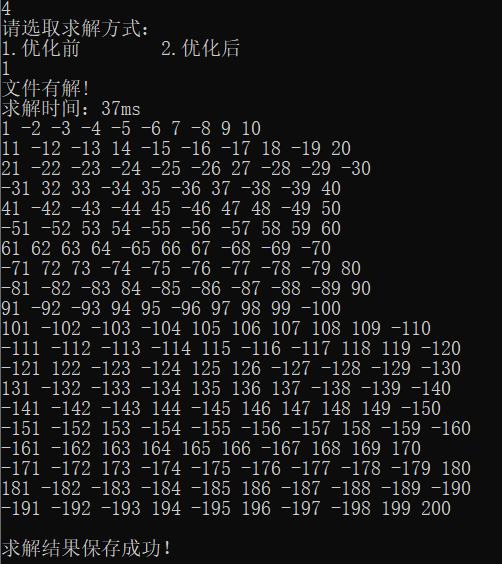


图30 优化前 图31 优化后

优化率：[(37-32)/37]\*100%=13.51%

1. 测试算例9：满足算例\M\sud00001.cnf，测试结果如图32，33所示：

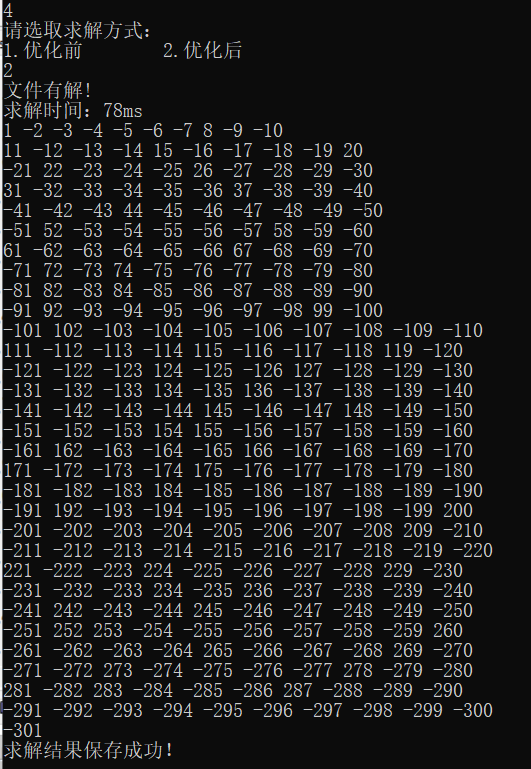
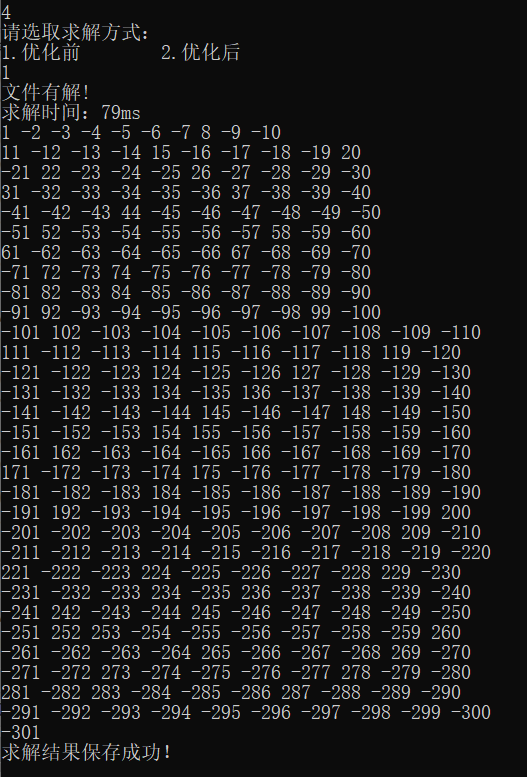


图32 优化前 图33 优化后

优化率：[(79-78)/79]\*100%=1.27%

1. 测试算例10：满足算例\M\sud00009.cnf，测试结果如图34，35所示：

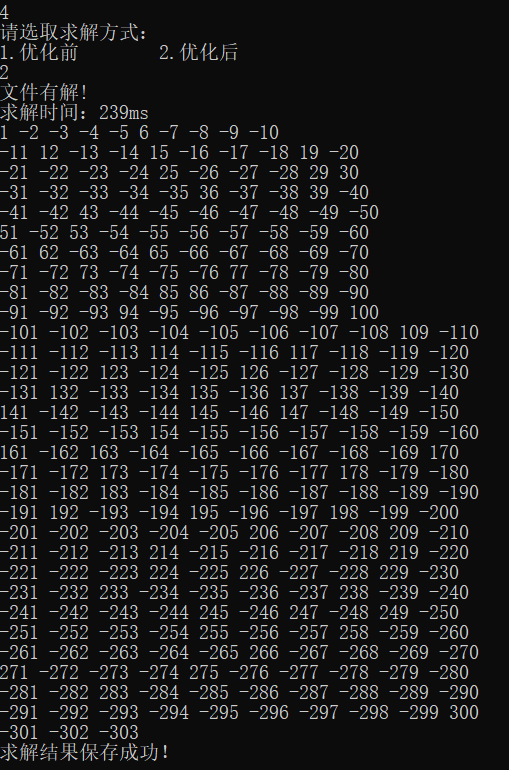
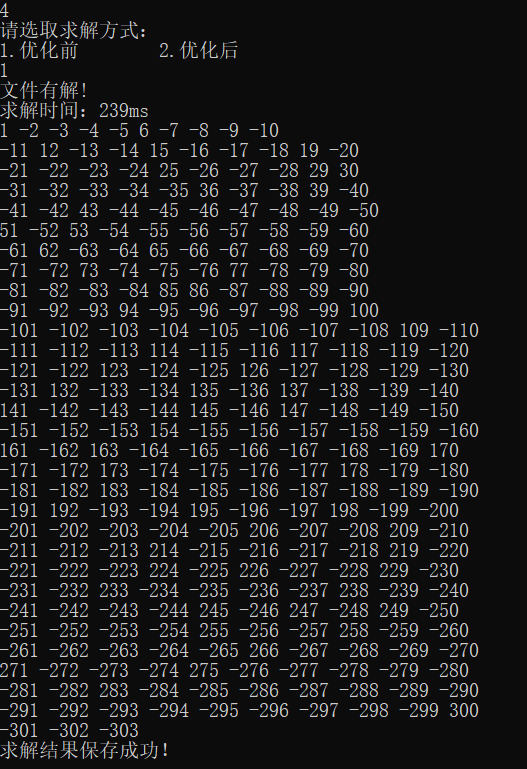


图34 优化前 图35 优化后

优化率：0

1. 测试算例11：满足算例\M\sud00079.cnf，测试结果如图36，37所示：

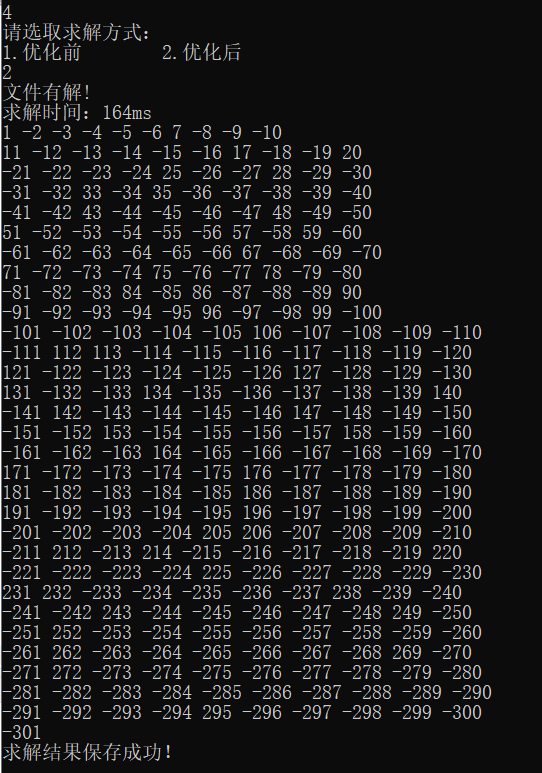
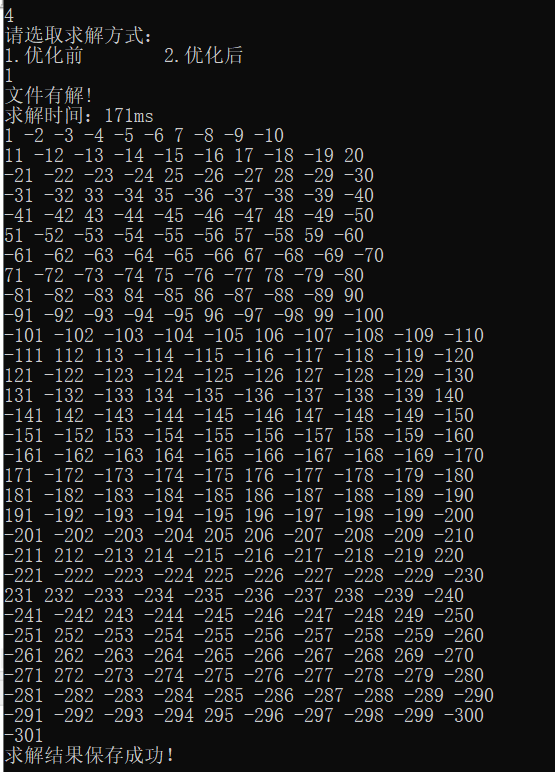


图36 优化前 图37 优化后

优化率：[(171-164)/171]\*100%=4.09%

1. 测试算例12：满足算例\M\sud00861.cnf，测试结果如图38，39所示：

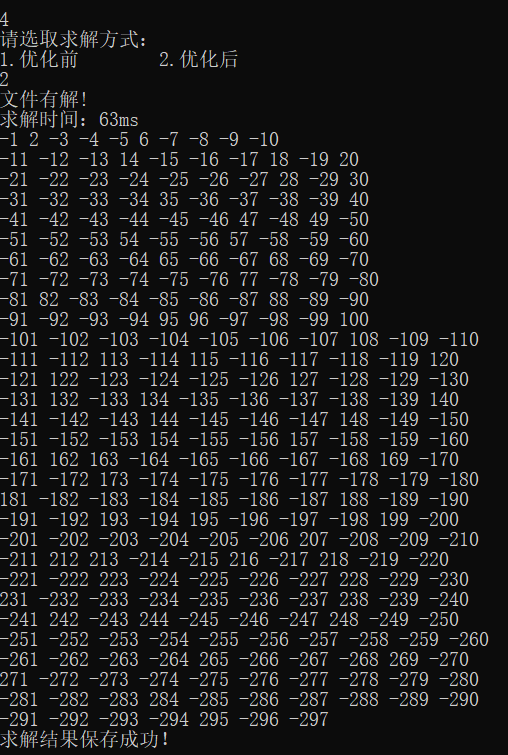
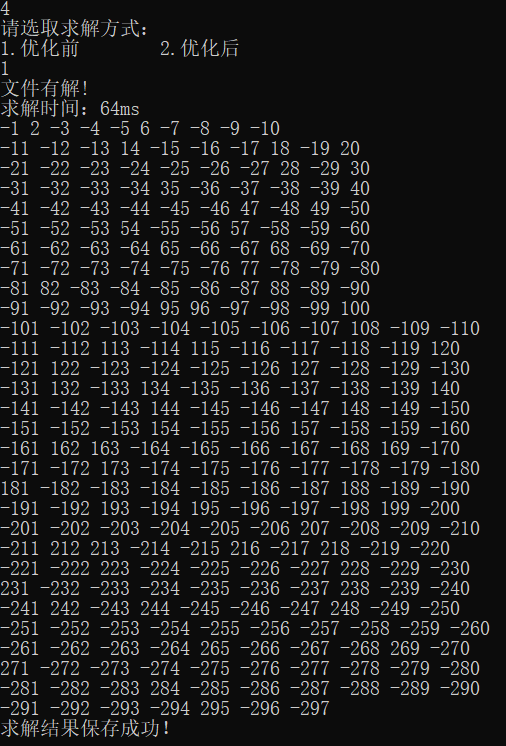


图38 优化前 图39 优化后

优化率：[(64-63)/64]\*100%=1.56%

1. 测试算例13：不满足算例\u-problem10-100.cnf，测试结果如图40，41所示：

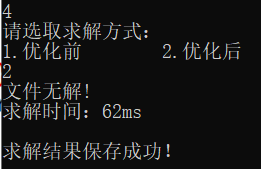
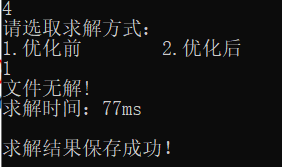


图40 优化前 图41 优化后

优化率：[(77-62)/77]\*100%=19.48%

1. 测试算例14：tst\CBS\_k3\_n100\_m403\_b10\_0.cnf，测试结果如图42，43所示：

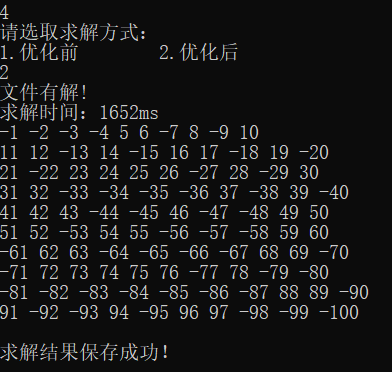
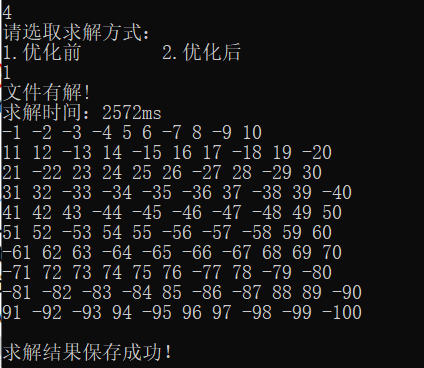


图42 优化前 图43 优化后

优化率：[(2572-1652)/2572]\*100%=35.77%

1. 测试算例15：tst\qg7-09.cnf，测试结果如图44，45所示：

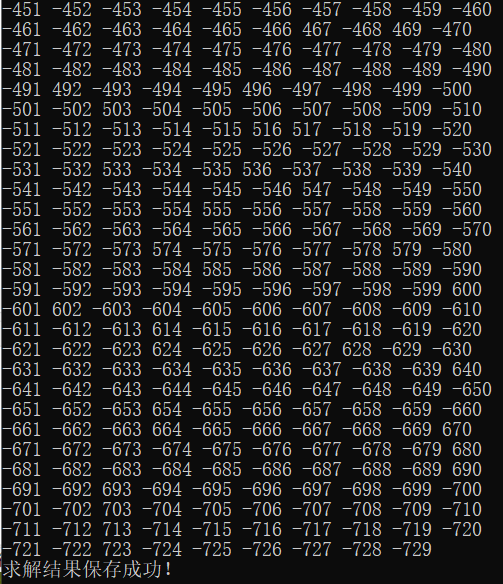
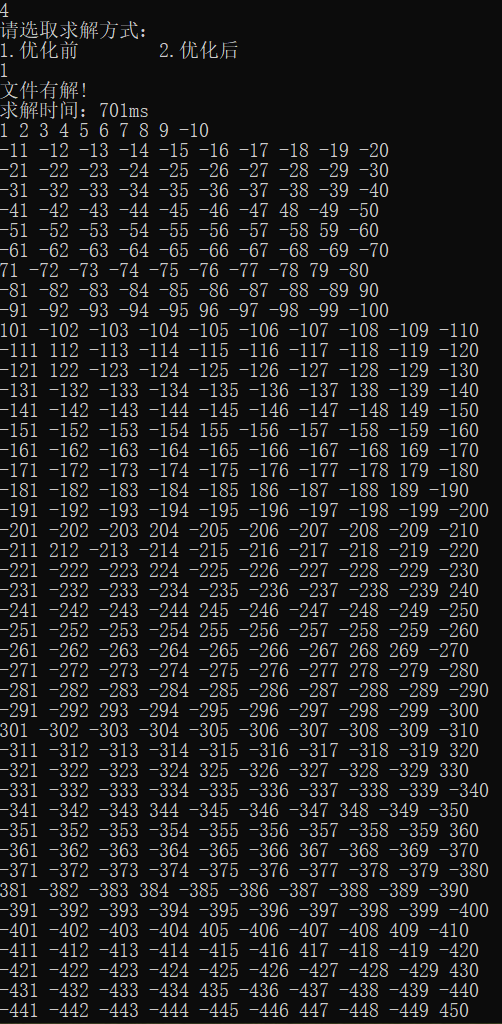


图44 优化前

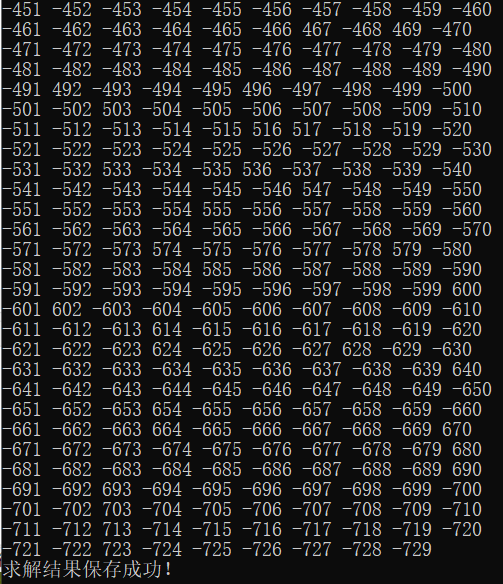
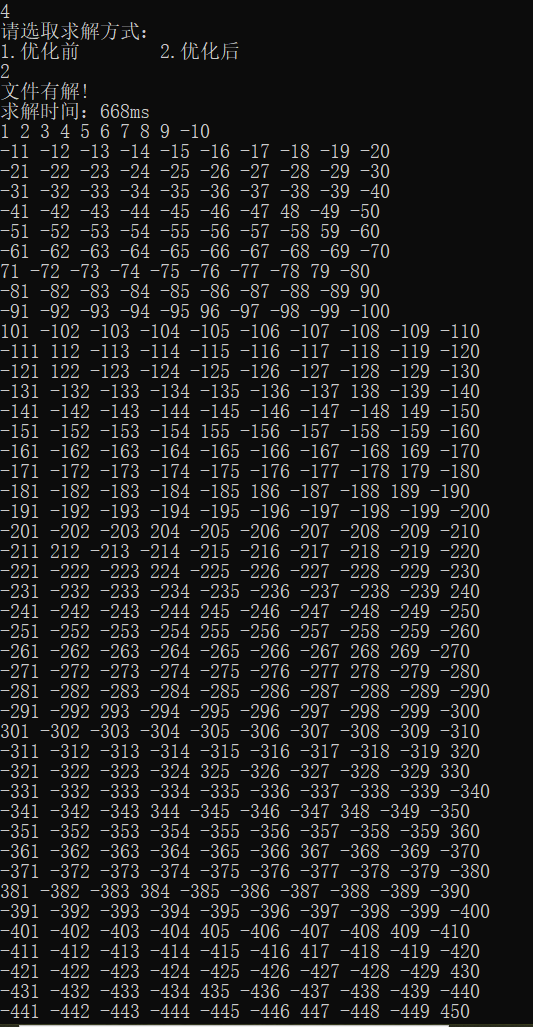


图45 优化后

优化率：[(701-668)/701]\*100%=4.71%

1. 测试算例16：tst\bw\_large.a.cnf，测试结果如图46，47所示：

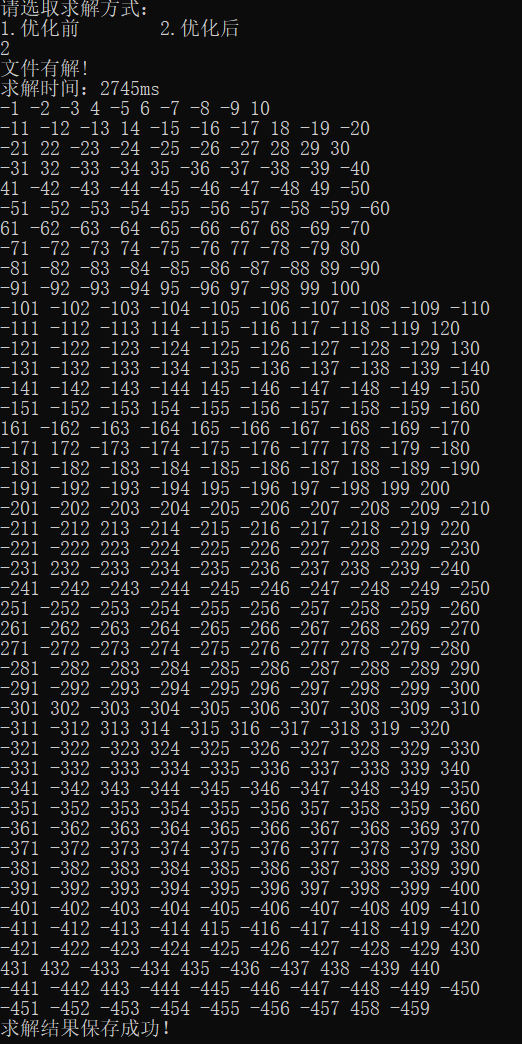
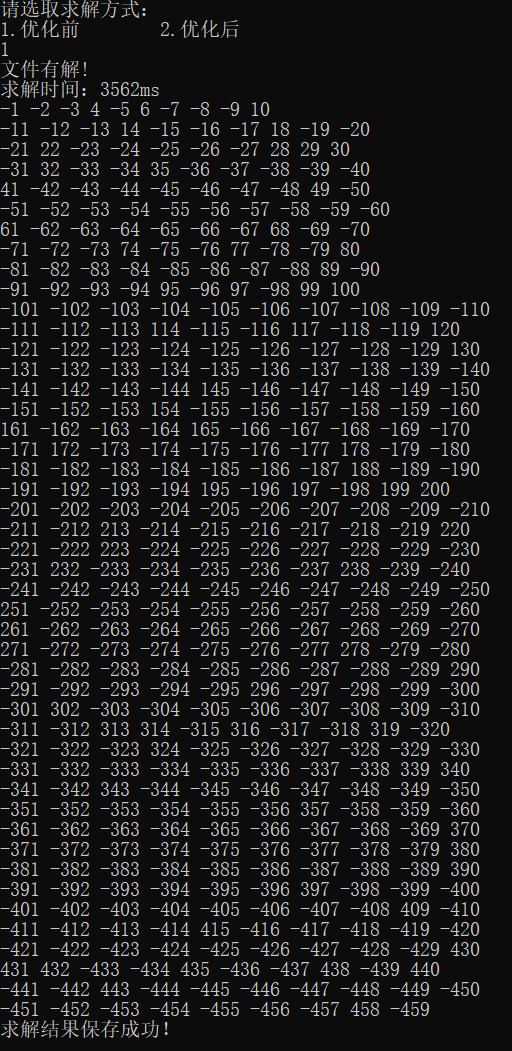


图46 优化前 图47 优化后

优化率：[(3562-2745)/3562]\*100%=22.94%

1. 测试算例17：tst\qg4-08.cnf，测试结果如图48，49所示：

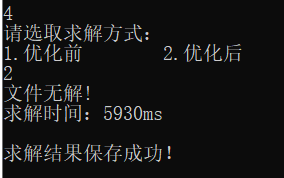
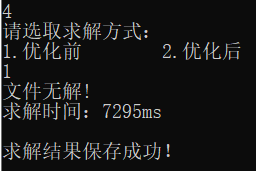


图48 优化前 图49 优化后

优化率：[(7295-5930)/7295]\*100%=18.71%

1. 测试算例18：tst\sw100-1.cnf，测试结果如图50，51所示：

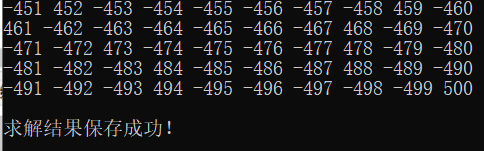
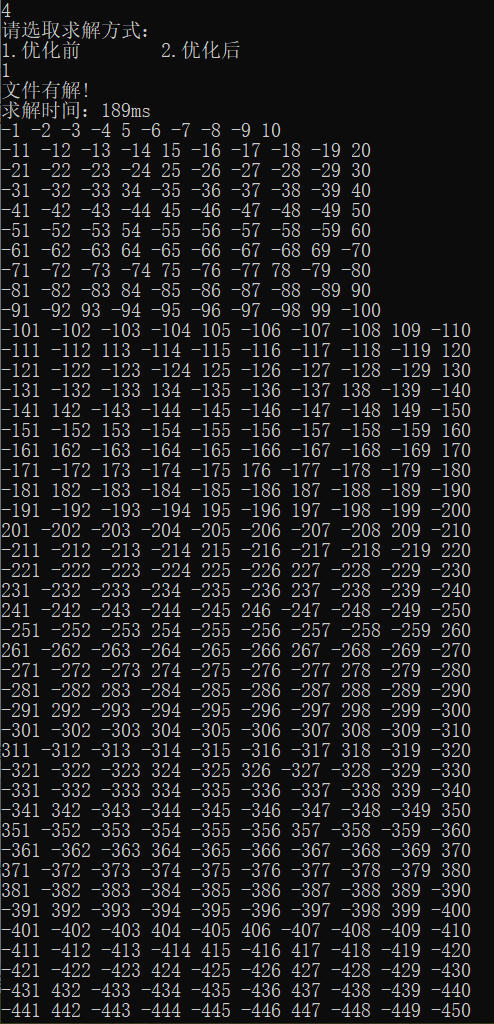


图50 优化前

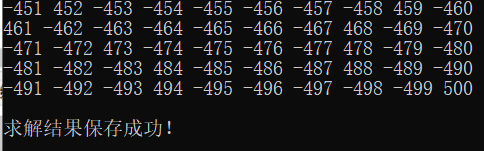
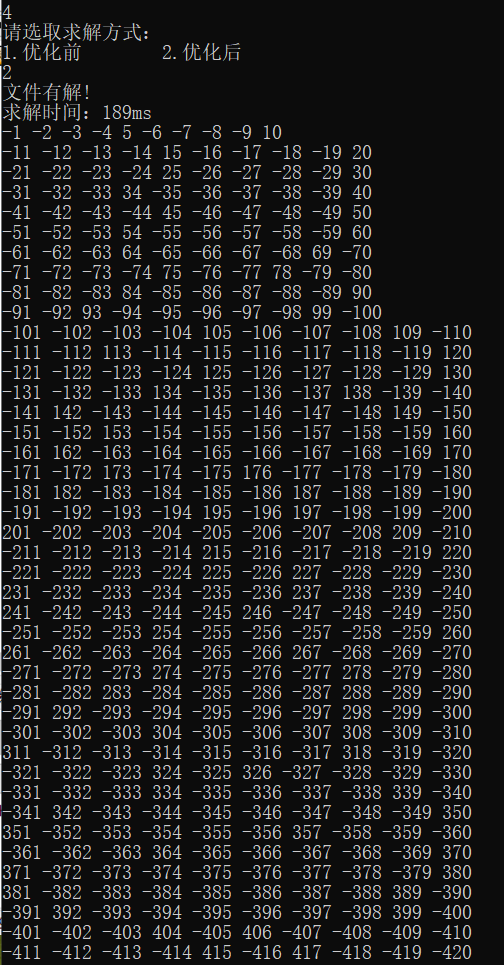


图51 优化后

优化率：0

#### 4.2.4 CNF算例测试总结表格

算例测试总结如下表3所示：

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 | 优化率 |
| 基准算例\功能测试\sat-20.cnf | 20 | 91 | 4.55 | 满足 | 优化前：0ms  优化后：0ms | 0 |
| 基准算例\功能测试\unsat-5cnf-30.cnf | 30 | 420 | 14 | 不满足 | 优化前：220ms  优化后：218ms | 0.91% |
| 基准算例\性能测试\ais10.cnf | 181 | 3151 | 17.41 | 满足 | 优化前：4675ms  优化后：3723ms | 20.36% |
| 基准算例\性能测试\sud00009.cnf | 303 | 2851 | 9.41 | 满足 | 优化前：273ms  优化后：275ms | -0.73% |
| 满足算例\S\problem11-100.cnf | 100 | 600 | 6 | 满足 | 优化前：15ms  优化后：0ms | 100% |
| 满足算例\M\bart17.shuffled-231.cnf | 231 | 1166 | 5.05 | 满足 | 优化前：15ms  优化后：0ms | 100% |
| 满足算例\M\problem-5-200.cnf | 200 | 320 | 1.6 | 满足 | 优化前：49ms  优化后：47ms | 4.08% |
| 满足算例\M\problem-12-200.cnf | 200 | 1200 | 6 | 满足 | 优化前：37ms  优化后：32ms | 13.51% |
| 满足算例\M\sud00001.cnf | 301 | 2780 | 9.24 | 满足 | 优化前：79ms  优化后：78ms | 1.27% |
| 满足算例\M\sud00009.cnf | 303 | 2851 | 9.41 | 满足 | 优化前：239ms  优化后：239ms | 0 |
| 满足算例\M\sud00079.cnf | 301 | 2810 | 9.34 | 满足 | 优化前：171ms  优化后：164ms | 4.09% |
| 满足算例\M\sud00861.cnf | 297 | 2721 | 9.16 | 满足 | 优化前：64ms  优化后：63ms | 1.56% |
| 不满足算例\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% |
| tst\qg4-08.cnf | 512 | 9685 | 18.92 | 不满足 | 优化前：7295ms  优化后：5930ms | 18.71% |
| tst\sw100-1.cnf | 500 | 3100 | 6.2 | 满足 | 优化前：189ms  优化后：189ms | 0 |

#### 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（其实也可以PLAY一半后选择直接求解，因为如果选择直接求解，我会在转化CNF文件之前调用初始化棋盘的函数，这样填充的数值将没有影响），第三步就是Dpll求解出结果或者是验证结果。

1. 数独棋盘的初始化测试，预计结果：输出成功读取信息并展示当前棋盘，如图52所示：

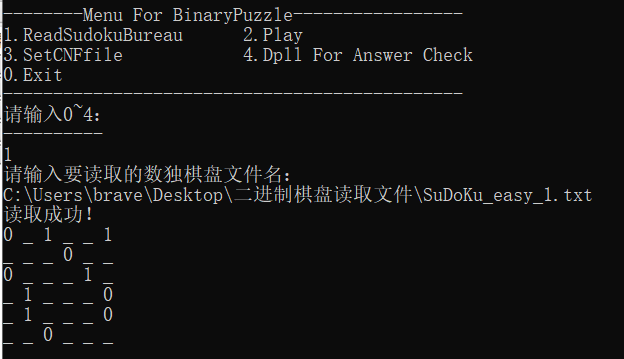


图52 棋盘初始化

1. 第二步选择直接转化为CNF文件Dpll求解，如图52，53所示：

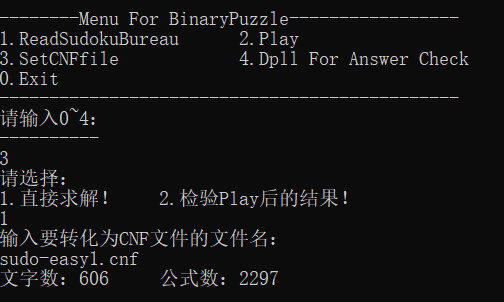


图52 直接转化cnf

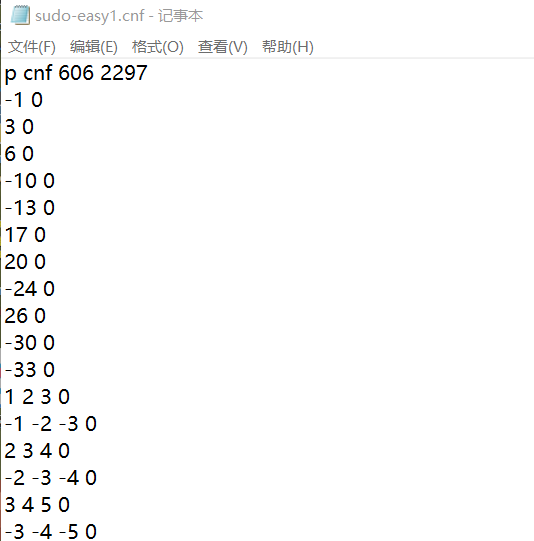


图53 转化后的cnf文件

1. DPLL直接求解出结果并展示，如图54所示：

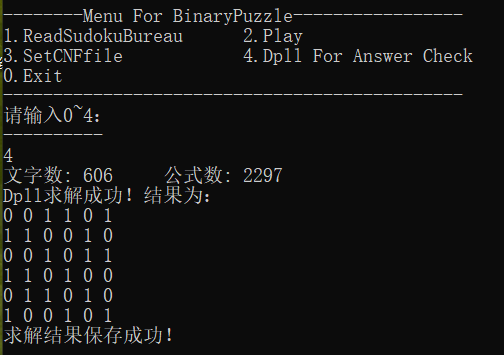


图54 Dpll求解出结果

1. 第二步选择先PLAY，再转化为cnf文件直接验证，如图55所示：

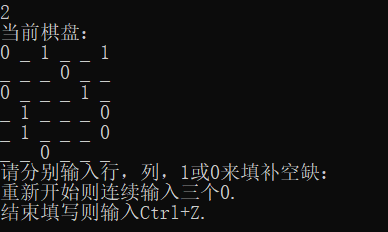


图55 PLAY初始

1. 在PLAY时尝试更改初始棋盘的内容，预计结果提示不可更改信息，如图56所示：

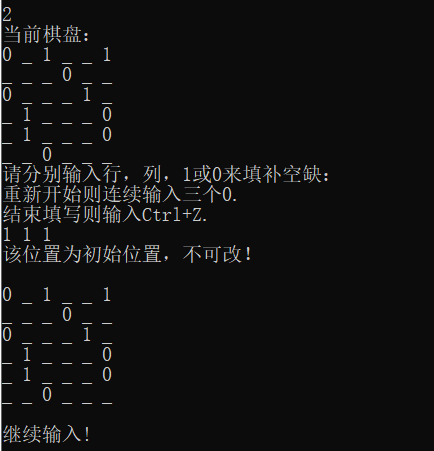


图56 尝试更改初始位置

1. PLAY交互过程展示，如图57所示：

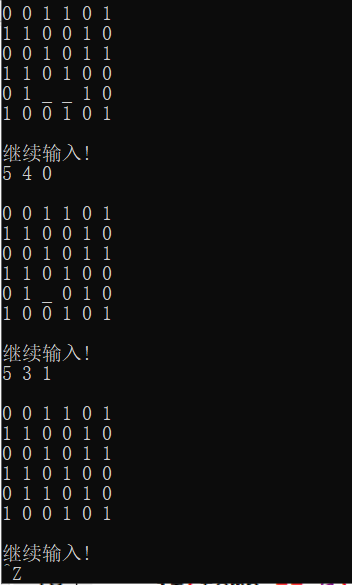


图57 PLAY交互过程

1. PLAY未填满棋盘，预计结果：输出提示信息，如图58所示：

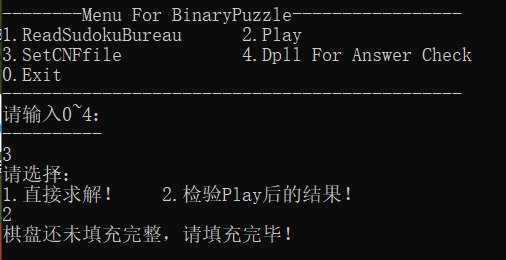


图58 PLAY棋盘未填满

1. Dpll验证PLAY后的求解结果，如图59，60所示：

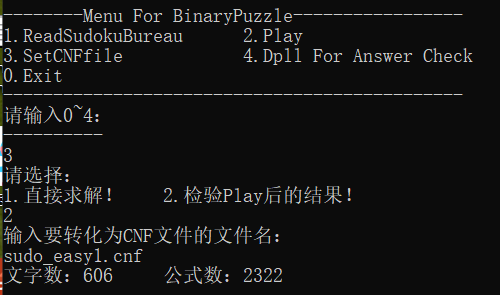


图59 cnf文件转化

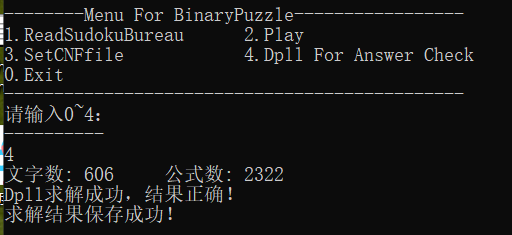


图60 Dpll验证

## 5总结与展望

### 5.1全文总结

对自己的工作做个总结，主要工作如下：

（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的方式上对求解器做了一定优化；

（3）完成了二进制数独游戏的初始化，cnf文件转化，归约为Dpll求解，并实现了游戏的可交互性；

（4）设计了测试方案并完成了多个算例的测试。

### 5.1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）这次实现的SAT求解器较为低级，在测试的时候，最大的算例只是测到了729个变元这个程度，而且还是在这个算例里公式比较简单的情况下，大部分稍微大一点或者复杂一些的算例电脑有的时候会直接跑到重启，由此可见这个SAT求解器的性能不够好。在今后的工作中，希望可以得到更多的编程和项目经验，多学一些优化的策略；

（2）在实现DPLL算法的时候，我最早用回溯的思想，例如将删除的单子句恢复这样的操作，最后发现设计的存储结构会很复杂，求解的时间确实比较快（应该是空间换时间的原因），但是结果会出现某些变元空缺这样的情况，看了很久也没有发现自己的问题在哪里，最后将存储结构改为任务书上建议的之后采取克隆存储结构的办法转化了回溯的问题，在今后的工作中，希望能够接触到更能体现回溯思想的Dpll算法；

（3）在二进制数独模块里，我感觉的交互系统写的十分一般，就是简单的用位置与转化公式来进行棋盘填充，用结构型数组做棋盘展示，在今后的工作中，希望可以通过更多高级语言的学习以及更深入的c语言理解学习到更好的数独游戏交互实现。

## 6体会

在本次课程设计的过程中我收获到了很多编程知识，也认识到了自己的很多不足，比如实现的SAT求解器性能比较低，二进制数独的交互系统设计的也相对简单等等。以下是我按照完成整个课设的时间顺序写的一些体会：

1.万事开头难，在最初开始的时候任务书上的内容体会了几遍才能清晰地理解其中的意思，特别是数独转化为CNF文件约束3提到的什么附加变元，在当时还没有写SAT求解器之前确实让人非常难以理解，最后开始自己上手依次实现cnf文件解析，dpll设计之后才慢慢地明朗起来；

2.在实现基于DPLL的SAT求解器的过程中，在网上找了很多资料来理解回溯等思想。在实现的过程中很多细致的问题就会导致最后的性能有很大的改变，例如在删除单子句里我多加了一个->next导致SAT求解器的求解时间改变很多，最后看了很久才找到问题；在DPLL上走了很多弯路，第一次设计的算法最后在求解的时候总是会漏掉几个变元，找了很久没找到问题就放弃了，在第二次设计的算法里，我准备用先按变元出现频率排序然后优先选取频率大的变元来做优化，最后编译成功后却无法求解，也是没有找到问题在哪，就换了一个优化的方向，最后也是成功实现了SAT求解器，但性能不够理想；

3.在二进制数独的实现过程中，由于时间比较紧张，我就没有去深入学习挖洞法构造棋盘，而是选择了最简单的文件读取。在该部分最难的地方莫过于约束三附加变元和相应公式怎么转化到cnf文件的问题，当时我在草稿纸上列了各部分变元以及附加变元的规律，最早想法是干脆全列出来，反正是确定的公式，最后列着列着就放弃了，想了很久才设计出一个for嵌套循环，设置多个变量的思路，解决了问题；

4.最后就是整合为项目的问题，之前由于没有做项目的经验，以为整合为项目很简单，最后在网上看了教程，自己实际操作一遍才发现有很多的问题。主要就是在刚开始写各个模块的时候，数据结构等东西定义的比较随意，导致整合为项目的时候浪费了很多时间，还有一些类似于重复定义的问题，感觉根本无法理解，也是在网上查资料才慢慢理解和解决这些问题。

5.总而言之，在这次课设实现的过程中收获了非常多的知识，实践出真知的确是真理。

## 参考文献

[1] 曹计昌，卢萍，李开. C语言与程序设计. 电子工业出版社，2013

[2] 严蔚敏等.数据结构（C语言版）.清华大学出版社

[3]Stephen Prata 《C Primer Plus》人民邮电出版社，2016

[4] <https://download.csdn.net/download/zyk1060513882/10166331(CSDN> 基于DPLL算法的完备性SAT求解器研究)

[5] 严蔚敏等. 数据结构题集（C语言版）.清华大学出版社

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

## 附录

源程序：

（1）define.h:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <time.h>

#include <malloc.h>

typedef int status;

#define OK 1

#define ERROR 0

#define Untrue -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

//存储结构

struct Literal{

struct Literal \* next;

int value;//文字值

};

struct Clause{

struct Clause \* next;

struct Literal \* head;

};

//用于保存棋盘

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);

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

/\*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[]);

(2)CNFfileparser.c:

#include "define.h"

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

/\*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){

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", &wordnumber, &formulanumber);//按格式读入文字数与公式数

printf("文字数: %d 公式数: %d\n", wordnumber, formulanumber);

variate = (int \*)calloc(wordnumber + 1, sizeof(int));

for(i = 1; i <= wordnumber; i++) variate[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){

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文件公式\*/

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("\n文件输出成功！\n");

}

/\*CNF文件保存\*/

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 ", wordnumber);

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

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;

}

(3)Dpllsolver.c

#include "define.h"

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

/\*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

variate[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(wordnumber + 1, sizeof(int));//标志数组，初始化为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 <= wordnumber; 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; //表示没有只有一种出现形式的文字

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

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

while(T != NULL){ //删除相应子句

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){

return root->head->value;

}

/\*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){

variate[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 destoryClauses(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(wordnumber+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)) 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(variate[abs(current\_literal->value)] == -1) variate[abs(current\_literal->value)] = current\_literal->value > 0 ? 1 : 0;

current\_literal = current\_literal->next;

}

T = T->next;

}//此时虽然不为空集，但已经满足可解条件，给文字做最后赋真值

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; //代表不满足

if(isAllPureLiteral(root)) return OK; //代表满足

return Untrue;//代表未确定

}

/\*dpll算法\*/

status dpll(struct Clause \* root){

temp = check(root);

if(temp != Untrue){

destoryClauses(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){

destoryClauses(root);

return temp;

}//同上

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

}

if (WAY == 2){

while(1){ //处理只以一种形式出现的文字

temp = check(root);

if(temp != Untrue){

destoryClauses(root);

return temp;

}

if(!removePureLiteral(root)) break;

}

}

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

destoryClauses(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 < 100; 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 <= wordnumber; i++) {

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

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

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

}

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

}

else{

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

fprintf(fp,"v \n");

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

}

printf("求解结果保存成功！\n");

fclose(fp);

}

(4)BinaryPuzzle.c:

#include "define.h"

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

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

/\*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");

}

/\*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Ctrl+Z.\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 {

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

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

printf("该位置为初始位置，不可改！\n");

}

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

}

putchar('\n');

Showqipan(qipan);

printf("\n继续输入!\n");

}

}

/\*检查棋盘是否填充满\*/

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));

else continue;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//约束1：行，列中没有三个连续的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的个数相同

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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;

//处理6行

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);

}

(5)display.c:

#include "define.h"

/\*------------------------主函数------------------------\*/

int main()

{

char filename[100], copy\_filename[100];//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swith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

while (op) {

system("cls");

printf("\n---Menu For Puzzle Or SAT-----\n");

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

printf("0.Exit\n");

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

printf("请输入0~2：\n");

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

scanf("%d", &op);

switch (op) {

case 1:

while (SAT) {

system("cls");

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

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

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

printf("0.Exit\n");

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

printf("请输入0~4：\n");

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

scanf("%d", &SAT);

switch (SAT) {

case 1:

printf("请输入读取的cnf文件名：\n");

scanf("%s", filename);

strcpy(copy\_filename, filename);

struct Clause \* 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();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();getchar();

break;

case 3:

if (flag\_cnf) Traversefile(root);

else printf("文件还未读取！\n");

getchar();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("求解时间：%dms\n", end - start);

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

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

printf("%d ", temp);

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

}

putchar('\n');

writeResult(copy\_filename, 1, end - start);

}

else{

end = clock();

printf("文件无解!\n");

printf("求解时间：%dms\n", end - start);

putchar('\n');

writeResult(copy\_filename, 0, end - start);

}

flag\_cnf = 0;

//destoryClauses(root);

}

else printf("文件还未读取！\n");

getchar();getchar();

break;

case 0:

break;

default:

printf("输入错误！\n");

getchar();getchar();

break;

}

}

getchar();getchar();

break;

case 2:

while (SuDoKu) {

system("cls");

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

printf("1.ReadSudokuBureau 2.Play\n");

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

printf("0.Exit\n");

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

printf("请输入0~4：\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();getchar();

break;

case 2:

if (flag\_sudo) PlaySuDoKu(qipan);

else printf("数独文件还未读取！\n");

getchar();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();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\_setcnf);

if (way\_sudo == 1){

printf("Dpll求解成功！结果为：\n");

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

printf("%d ", variate[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);

//destoryClauses(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();getchar();

break;

case 0:

break;

default:

printf("输入错误！\n");

getchar();getchar();

break;

}

}

getchar();getchar();

break;

case 0:

break;

default:

printf("输入错误！\n");

getchar();getchar();

break;

}//end of switch

}//end of while

printf("欢迎下次使用本系统！\n");

system("pause");

return 0;

}//end of main