

**课程设计报告**

**题目：基于SAT的蜂窝数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： 大数据2201班**

**学 号： U202215566**

**姓 名： 刘师言**

**指导教师： 许贵平 \_**

**报告日期： 2023年9月12日**

**计算机科学与技术学院**

**目录**

**任务书** I

**1引言** 1

1.1 课题背景与意义 1

* + 1. SAT 1
    2. 数独游戏 1

1.2 国内外研究现状 2

1.3课程设计的主要研究工作 3

**2系统需求分析与总体设计** 4

2.1 系统需求分析 4

2.2 系统总体设计 4

**3系统详细设计** 6

3.1有关数据结构的定义 6

3.2主要算法设计 9

3.2.1 CNF文件处理 9

3.2.2 DPLL算法处理 10

3.2.3 蜂窝数独算法处理 16

**4系统实现与测试** 18

4.1系统实现 18

4.1.1 软硬件环境 18

4.1.2 数据类型定义 18

4.1.3 函数声明以及函数调用关系 20

4.2系统测试 22

4.2.1 交互系统展示 22

4.2.2 CNF文件处理模块测试和DPLL算法模块测试 23

4.2.3 CNF算例测试总结表格 50

4.2.4 数独游戏求解及简易游玩模块测试 51

**5总结与展望** 63

5.1全文总结 63

5.2工作展望 63

**6体会** 65

**参考文献** 66

**附录** 67

**任务书**

* **设计内容**

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于DPLL算法实现一个完备SAT求解器，对输入的CNF范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

* **设计要求**

要求具有如下功能：

**（1）输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)

**（2）公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)

**（3）DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)

**（4）时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)

**（5）程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)

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

**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 数独游戏**

数独游戏的历史渊源比较久远，数独是一种源自18世纪末的瑞士，后在美国发展并在日本得以发扬光大的数学智力拼图游戏。

早在数千年前，中国人就发明了[九宫图](https://baike.so.com/doc/4990670-5214413.html):在9个方格中，横行和竖行的数字总和是相同的。"数独"也不是什么新生事物，已经存在了数百年。18世纪，瑞士数学家莱昂哈德·欧勒发明了"拉丁方块"，但并没有受到人们的重视。直到20世纪70年代，美国杂志才以"数字拼图"的名称将它重新推出。日本随后接受并推广了这种游戏，并且将它改名为"数独"，大致的意思是"独个的数字"或"只出现一次的数字"。

现今流行的数独于1984年由日本游戏杂志《パズル通信ニコリ》发表并得了现时的名称。数独本是"独立的数字"的省略，因为每一个方格都填上一个非零的个位数。数独冲出日本成为英国当下的流行游戏，得归功于曾任[香港高等法院](https://baike.so.com/doc/5677194-5889867.html)法官的高乐德(Wayne Gould)。2004年，他在日本旅行的时候，发现杂志上介绍的这款游戏，便带回伦敦向《泰晤士报》推介并获得接纳。英国《每日邮报》也于三日后开始连载，使数独在英国正式掀起热潮。数独不仅是报章增加销量的法宝，脑筋动得快的《泰晤士报》还做起手机族的生意，花4.5英镑就能下载10则数独游戏到手机上玩。渐渐，其他国家和地区受其影响也开始风靡数独。

同类似的填字游戏不同，数独受欢迎的原因之一是它既不需要丰富的百科知识，也不要掌握大量的词汇，这使其能迅速为孩子和初学者所接受。根据游戏开始时的方格中已有的数字和位置，数独难易程度不同，有些复杂的甚至令数学家也不能完成。据著名的动游戏开发商Astraware Ltd.预计，移动数独游戏的版本多达几十种，Palm和Windows Mobile设备版本的数独游戏就各有20种左右。Sudokumo推出的移动数独游戏，能够下载到大多数手机中。这家位于英国的游戏软件公司表示，已经在全球卖出了7500套数独游戏，而且来自用户的兴趣还在增加。

**1.2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；

设计程序要求模块化，程序源代码进行模块化组织。主要模块包括如下：

1. 主控、交互与显示模块(display)
2. CNF解析模块(cnfparser)
3. 核心DPLL模块(solver)
4. 数独模块,包括游戏格局生成、归约、求解(sudoku)
5. 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；
6. 设计问题转化策略将数独问题归约为SAT问题并求解；
7. 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；  
 可以创建随机的蜂窝数独游戏并且具有一定的交互性，可以用来求解和尝试自主求解蜂窝数独游戏。

**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求解器和蜂窝数独游戏，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. 蜂窝数独游戏，能够完成如下功能：

（1） 随机数独游戏的创建（有难度选择）,并可以实现一定程度上的交互；

（2） 读入数独游戏文件（txt），转化为CNF文件DPLL进行求解，再可视化地将结果打印到屏幕上；

系统总体设计流程图如图2.1所示：



图2.1系统程序流程图

**3系统详细设计**

**3.1有关数据结构的定义**

1. 首先是对CNF文件进行读取，就要处理CNF文件中的文字和子句：
2. 对于文字需要用一个结构体Clause来定义：数据项有int型的文字值literal；int型用于标记删除的flag；指向下一个文字的指针nextl以及指向CNF公式邻接链表中下一个文字相同的子句结点的litline，便于后续DPLL的恢复与查找；
3. 对于子句需要用一个结构体Paradigm来定义：数据项有int型表示该子句的文字数的number；int型用于标记删除的flag；指向下一个子句的指针nextc以及指向文字的头指针clause；
4. 用一个结构体Root来定义CNF范式链表头结点类型，用于存储CNF范式信息：数据项有int型用于存储文字数量的lsize；int型用于存储子句数量的psize以及指向第一个子句的指针first；

表3-1对CNF文件读取和处理相关结构体

|  |  |  |
| --- | --- | --- |
| 数据名 | 用途 | 数据项 |
| Clause | 文字 | int：literal，flag；  struct Clause \*：nextl，litline； |
| Paradigm | 子句 | int： number，flag;  struct Paradigm \*nextc;  struct Clause \*clause; |
| Root | 头节点 | int lsize，psize  Paradigm \*first |

关于部分数据结构的图直观表示，如图3.1-1所示：

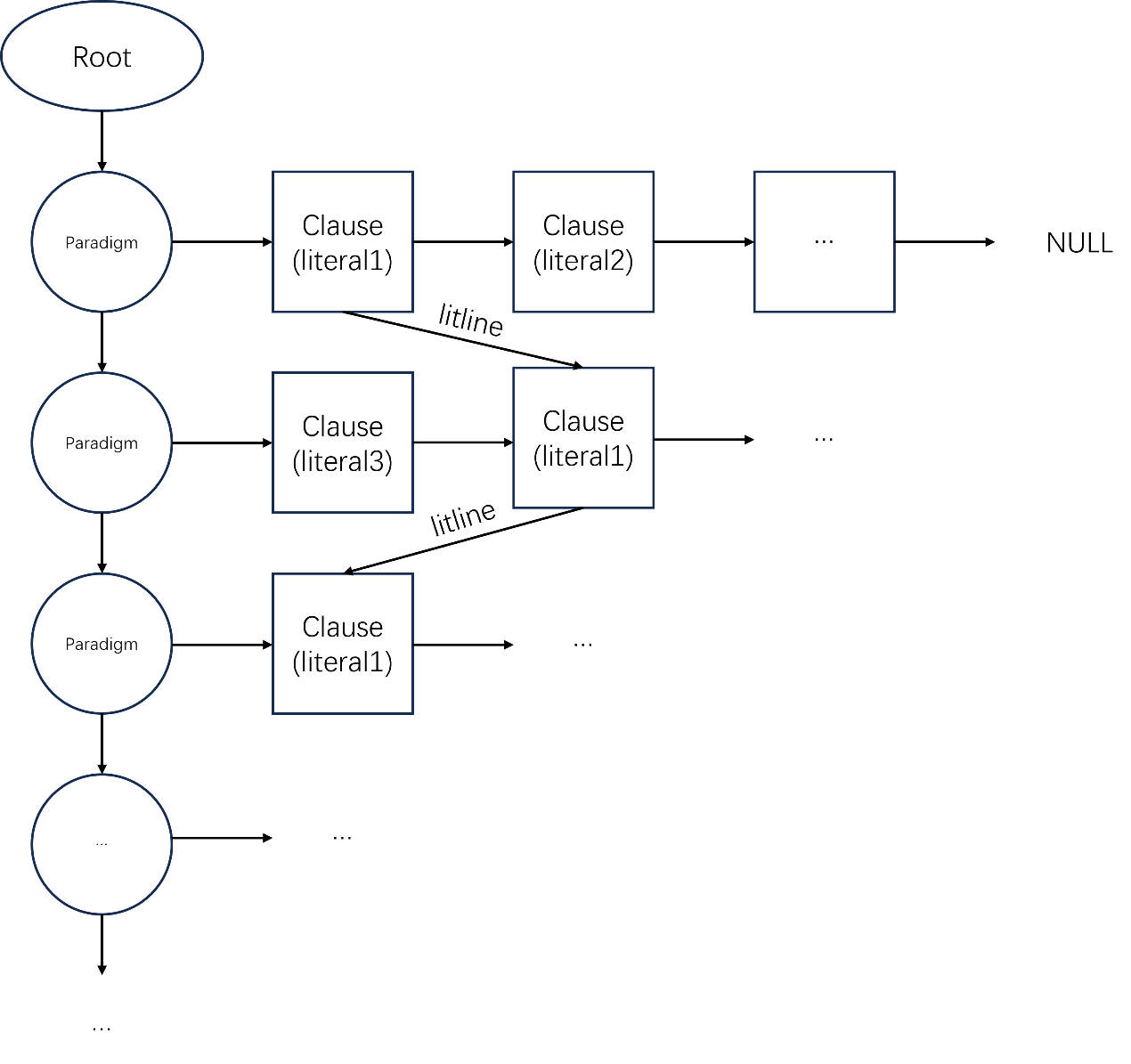


图3.1-1 部分数据结构表示图

1. 其次是DPLL算法中，需要对文字进行查找，就要遍历以及记忆：
2. 定义指向子句链表头结点的指针链表结点结构类型Paraline：数据项有指向子句链表头结点Paradigm的cline；指向下一链表结点的next；（优化）
3. 定义文字相关信息链表结构类型LitTrabverse：数据项有指向含有该正文字或负文字的子句头结点链表的头结点的Tra\_c，指向该正文字或负文字的文字结点的Tra\_l；（优化）
4. 存储变元信息的变元线性表结点结构ArgueValue：int型表示储存变元真值的Value；int型表示变元是否已赋值的IsInit；int型表示变元在所有子句中出现的总次数的Occur\_times，用于分裂政策的选择（优化）；LitTrabverse型存储变元所有正文字相关信息结构的Pos（优化）和存储变元所有负文字相关信息结构的Neg（优化）；

表3-2 DPLL算法处理相关结构体

|  |  |  |
| --- | --- | --- |
| 数据名 | 用途 | 数据项 |
| Paraline | 定义指向子句链表头结点的指针链表结点结构类型 | Paradigm \*cline  struct Paraline \*next; |
| LitTrabverse | 定义文字相关信息链表结构类型 | Paraline \*Tra\_c;  Clause \*Tra\_l; |
| ArgueValue | 定义存储变元信息的变元线性表结点结构类型 | int Value;//变元的真值  int IsInit;//变元是否赋值  int Occur\_times;//变元在所有子句中出现的总次数，用于分裂政策的选择  LitTrabverse Pos;//变元所有正文字的相关信息结构  LitTrabverse Neg;//变元所有负文字的相关信息结构 |

关于总数据结构的图直观表示，如图3.1-2所示：

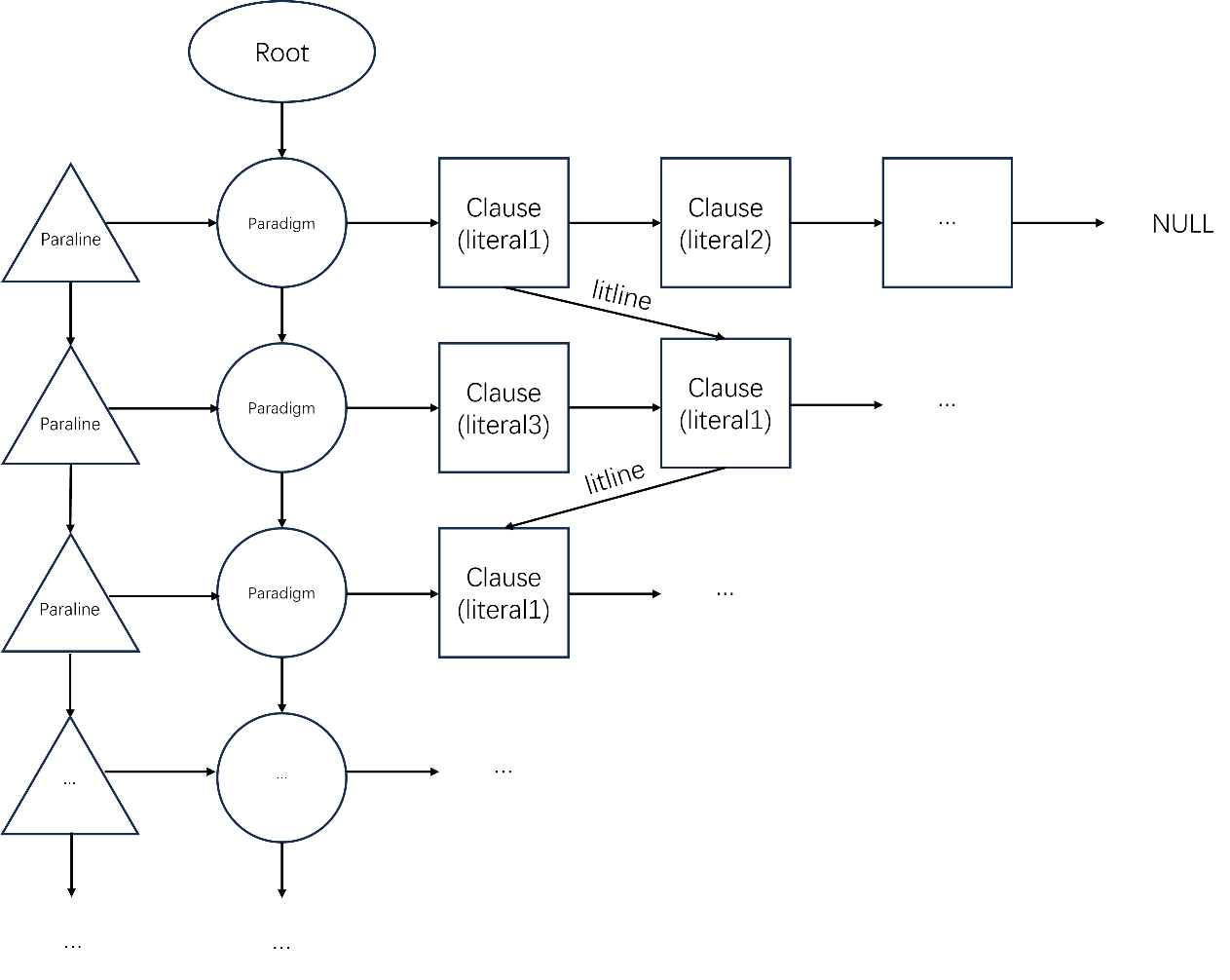


图3.1-2 总数据结构表示图

**3.2 主要算法设计**

**3.2.1 CNF文件处理**

1. 创建子句链表及文字链表

status CreateClause(FILE \*\*fp,Clause \*\*clause,Paradigm \*ClausHead,int first);

参数：(FILE\*\*)open：文件的指针地址

(Clause \*\*)clause:子句的头结点 Clause类型指针地址

(Paradigm \*)ClausHead：子句链表头节点指针

(int )first：创建的子句的第一个文字的值

返回值：i：子句内文字的数量

ERROR：该子句为空子句

OVERFLOW：空间不足溢出

读入一个值后，创建子句节点进行赋值，随后移动继续添加；关键在于不要遗漏正文字和负文字的子句链表添加以及变元真值表的变元出现次数增加，后续的DPLL操作会用到，这也是优化程序的关键步骤；

1. 创建CNF范式邻接链表及变元表

status CreateParadigm(FILE \*\*fp);

参数：(FILE\*\*)open：文件的指针地址

读入CNF文件，首先要区分c和p，c不读入，p开始读入，同时记录文字数和子句数，根据文字数初始化变元真值表，随后循环创建子句以及构建正文字和负文字的子句链表，优化时创建数组来降序排列变元的出现次数，进而分裂原则选择文字进行优化；

1. 销毁所有链表及线性表结构

status DestroyParadigm(Root \*r);

参数：(Root\*)r：CNF邻接链表头结点指针

用while循环销毁创建的子句链表Root，再释放变元正负文字信息链表存储空间；

1. 判断CNF范式中是否还含有单子句

status HasUnitClause(Root \*r);

参数：(Root\*)r：CNF邻接链表头结点指针

返回值：存在单子句：该单子句所含文字

不存在单子句：0

进入子句链表后只要判断子句头节点的number是否为1即可判断该子句是否为单子句，再返回文字值；

1. 判断参数指针c指向的子句头结点对应子句是否为单子句；

status isUnitClause(Paradigm \*p);

参数：(Paradigm\*)p：子句链表头结点指针

返回值：TRUE：是单子句

ERROR：不是单子句

只需判断该子句头节点的number是否为1即可判断是否为单子句；

**3.2.2 Dpll算法处理**

/\*在整个CNF公式中取一个文字（3种变元选择策略）\*/

1. 取每次DPLL处理后公式中出现次数最多（Occur\_Times最大）的文字

status FindLiteral1(Root \*r);

参数：(Root\*)r：CNF邻接链表头结点指针

返回值：公式非空：选中文字的值

公式中没有剩余文字：0

找到变元表正序第一个未被赋真值的变元，num记录其序号，并将flag值置为1进行记录，若未找到未被赋真值的变元，则将flag值置为0；当flag为1时遍历真值表，找到Occur\_times值最大的文字并返回；

1. 取变元表正序正序第一个未赋真值（IsInit=0）的变元的正文字

status FindLiteral2(Root \*r);

参数：(Root\*)r：CNF邻接链表头结点指针

返回值：公式非空：选中文字的值

公式中没有剩余文字：0

遍历真值表中的正文字链表即可；

1. 取原公式中未赋真值的变元中出现次数最多（Occur\_Times最大）的变元正文字

status FindLiteral3(Root \*r);

参数：(Root\*)r：CNF邻接链表头结点指针

返回值：公式非空：选中文字的值

公式中没有剩余文字：0

根据之前降序排序的SortList数组储存原公式的变元次数，遍历查找未赋值，即IsInit为0的文字，再返回该文字；

/\*单子句原则\*/

1. 删除包含文字l的所有子句

status DeleteClause(Root \*r,int l);

参数：(Root\*)r：CNF邻接链表头结点指针

(int)l：真值为1的文字

先判断l的正负值，如果l为正文字情况，搜索l对应变元的正文字信息链表，进入子句链表同时p->flag=abs（l）表明该子句因为l而删除；将该子句的所有文字链表节点flag=abs（l）表明因为l而删除。与此同时子句数，文字数和变元次数都相应减少。负文字情况同理可得；函数运行流程图如图3.2-1所示：

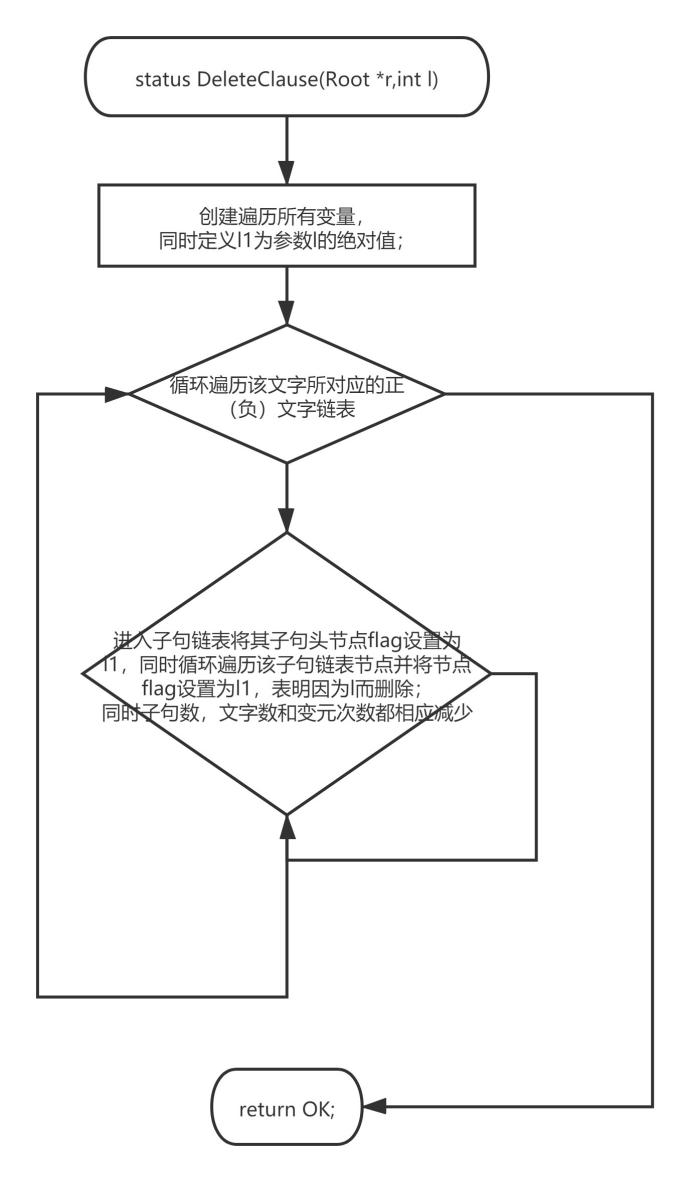


图3.2-1 删除出现了文字l的所有子句

1. 删除文字-l的所有子句链表结点

status DeleteLiteral(Root \*r,int l);

参数：(Root\*)r：CNF邻接链表头结点指针

(int)l：真值为1的文字

返回值：OK：成功删除文字

FALSE：公式无解

先判断l的正负值，如果l为正文字情况，搜索l对应变元的正文字信息链表，进入子句链表；将该子句的找到对应-l文字值的链表节点，表明因为l而删除。如果子句文字数为1，则删除失败返回ERROR，否则链表节点的flag的值置为abs（l）；与此同时该子句文字数和变元次数都相应减少。负文字情况同理可得；函数运行流程图如图3.2-2所示：

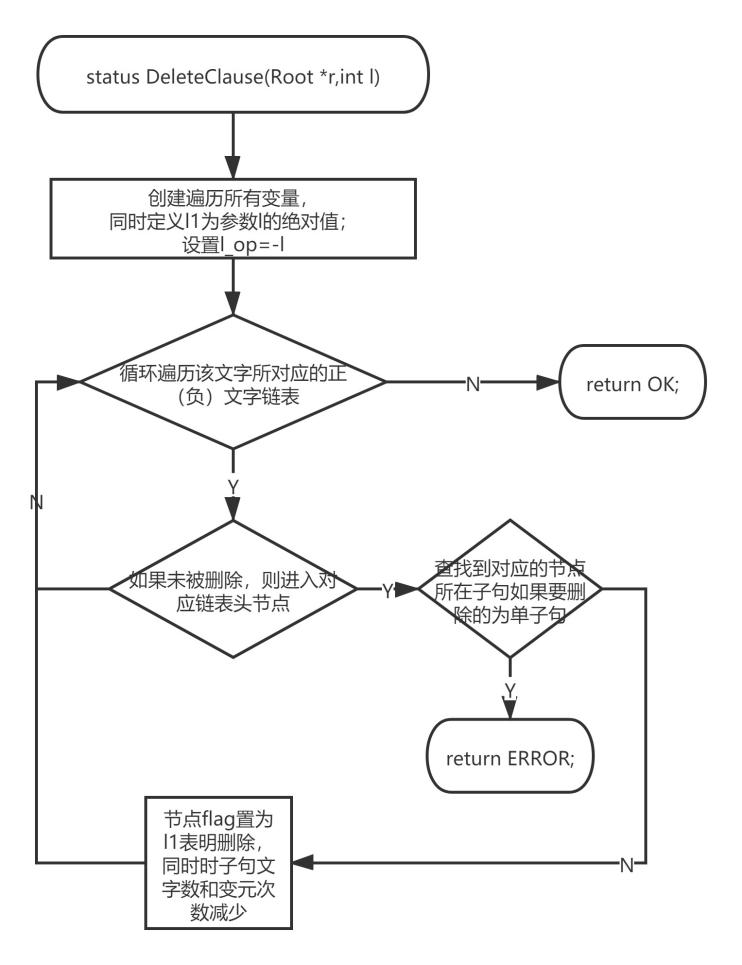


图3.2-2 删除所有文字为-l的子句链表结点

1. 在CNF范式链表表头添加含有文字l的单子句链表结点

status AddClause(Root \*r,int l);

参数：(Root\*)r：CNF邻接链表头结点指针

(int)l：增加的单子句内的唯一文字

返回值：OVERFLOW：空间不足溢出

OK：子句增加成功

用malloc函数分配内存空间之后移动r->first（子句链表头结点）指向；同时在变元表内l对应变元的正（负）文字信息链表表头增加新增结点信息；增加子句数和文字数；

1. 删除CNF范式邻接链表中第一个含有文字l的单子句链表

status RemoveHeadClaus(Root \*r,int l);

参数：(Root\*)r：CNF邻接链表头结点指针

(int)l：要删除的单子句内的唯一文字

返回值：OK：删除成功

要删除的结点信息必在变元表内l对应变元的正（负）文字信息链表表头，移动删除和free即可。此时还需要特判是否在子句链表第一个子句，若是第一个子句，则r->first=p->nextc;否则寻找从表头开始第一个只含文字l的单子句，再移动删除和free即可；

1. 恢复假定文字l真值为1时对CNF范式链表的操作

status Recover(Root \*r,int l);

参数：(Root\*)r：CNF邻接链表头结点指针

(int)l：认为真值为1时出错，需要恢复的文字l

返回值：OK：邻接链表恢复成功

关键在于将原来DeleteClause和DelteLiteral的flag恢复为0即可，若l为正文字，对变元l的正文字信息链表进行搜索，寻找被删除的子句，同时对变元l的负文字信息链表进行搜索，寻找被删除的文字；若l为负文字，对变元abs(l)的负文字信息链表进行搜索，寻找被删除的子句，同时对变元abs(l)的正文字信息链表进行搜索，寻找被删除的文字；

1. 遍历CNF范式链表

void ParadigmTrabverse(Root \*r);

参数：(Root\*)r：CNF邻接链表头结点指针

依次遍历子句，进入子句后遍历子句链表节点输出文字值即可；

1. 保存CNF范式公式的解和求解时间信息

status SaveAnswer(ArgueValue \*ValueList,int solut,int time)

参数：(ArgueValue\*)ValueList：变元表

(int)solut：CNF公式的解，有解为1，无解为0

(int)time: DPLL求解时间；

返回值：ERROR：文件打开失败

OK：解和时间存储成功

首先创建相同文件名但不同文件格式（.res格式）的文件，打开输入s solut（1为有解，0为无解），若solut为1则在v后打印变元真值到文件中，否则不打印，最后打印t 和时间time；

1. SAT流程控制

status SAT(void);

SAT解CNF的流程控制函数；

1. 采用第一种变元选择策略的递归算法DPLL函数（优化）

status DPLL(int num,int times);

参数：(int)num：该次进入函数真值设为1的变元文字值

(int)times：第一次进入DPLL函数为1，其后为2

返回值：FALSE：公式无解

TRUE：公式有解

求解CNF文件时，若初次进入DPLL函数，则不进入单子句循环，直接采用第一种变元选择策略选取l=FindLiteral1，进而进入DPLL（l，2）；

若不是初次进入，则进入while（）循环进行单子句规则，将选中变元l在变元表的真值设置为1（l>0）或0（l<0）,同时IsInit设置为1（已经赋值），若删除不成功则恢复原状态；搜索完所有单子句后退出再找新的文字值设置其真值为1，继续DPLL（l，2）；

若探测DPLL（l，2）返回值不为OK，即表明设l的真值为1时探测求解失败，故l真值只能为0，即其反文字真值为1；若栈容量instacksize==0，则，return FALSE，因为l已是本次DPLL1函数第一个进行探测的变元，所以探测失败，恢复递归进入本次函数前的邻接链表状态；若探测成功，即有解，则return OK；函数运行流程图如图3.2-3所示：

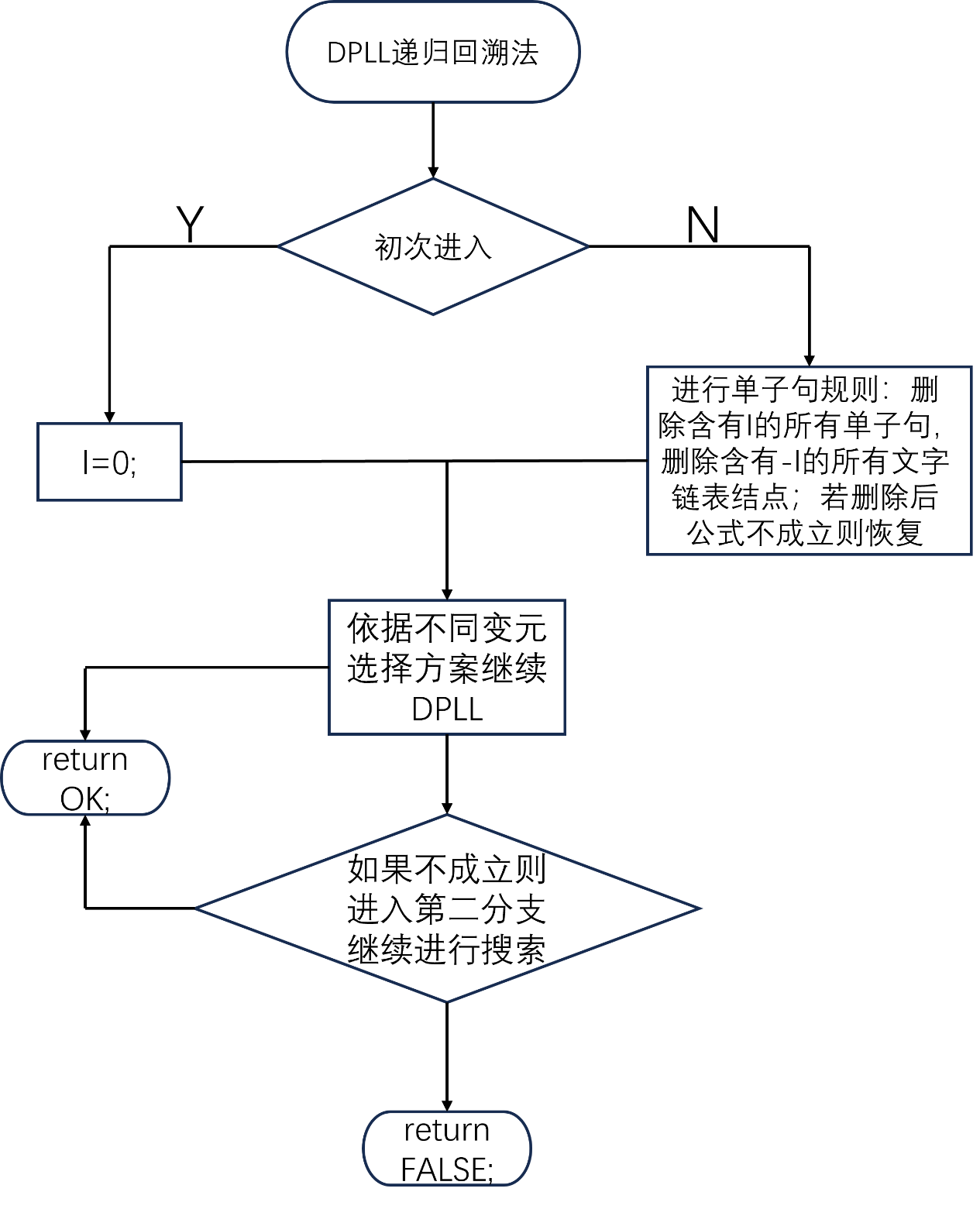


图3.2-3 递归算法DPLL函数

1. 采用第二种变元选择策略的递归算法DPLL函数（优化前）

status DPLL\_BEFORE(int num,int times);

参数：(int)num：该次进入函数真值设为1的变元序号

(int)times：第一次进入DPLL函数为1，其后为2

返回值：FALSE：公式无解

TRUE：公式有解

递归算法DPLL函数与采用第一种变元选择策略的递归算法DPLL函数同理；

1. 求解完成后将所有未赋真值的变元（在求解过程中真值为1，被化简舍去的变元）赋真值为1

void AnswerComplete(void)；

使用while（）循环给所有未赋真值的变元赋值；

1. 检查SAT求解结果是否正确

status AnswerCheck(int solut)；

参数：(int)solut：公式求解结果，有解为1，无解为0

返回值：TRUE：求解正确

FALSE：求解错误

如果solut为1，表明公式有解，则判断每个子句中是否至少有一个节点文字值真值为1，若有则说明子句为真，否则子句中无真值为1的文字，子句真值为0，求解错误；若所有子句真值均为1，求解正确；  
 如果solut为0，表明公式无解，则判断每个子句中是否至少有一个节点文字值真值为1，若有则说明子句为真，否则子句中无真值为1的文字，子句真值为0，求解正确；若所有子句真值均为1，求解错误；

**3.2.3 蜂窝数独算法处理**

1. 数独流程控制

status Sudoku(void);

Sudoku的流程控制台；

1. 预处理数独文件，获取原始数独盘信息并创建数独CNF公式文件

status SudokuPrepare(char\* filename, int op);

参数：(char\*)filename：打开的数独文件名

(int)op：数独题序号（即数独在文件中的行数）

返回值：ERROR：文件打开失败

OK：数独预处理完成

打开指定的数独文件，从其中选取随机或指定序号的数独题，获取数独原始盘信息并将已知的数独数字按顺序存储在table数组中，用于后续数独的归约操作；

1. 对数独进行归约，创建数独问题转化为SAT问题后的CNF文件

status SudokuCreateFile(void);

返回值：ERROR：文件打开失败

OK：数独归约完成，成功创建相应CNF文件

先创建.cnf文件，初始化填入共有549个变元，61个数独空格每个格对应9个变元，填入1～9中某一值则对应变元为真，其他值对应变元为假，共13965个子句。先将table中已知数字按坐标和数字转换为对应的1～549号变元形式并写入用于数独盘归约的CNF文件中，再根据已有蜂窝数独的规则依次对数独的横行、左斜行、右斜行进行归约，满足的约束条件为每个横行、左斜行和右斜行数字不重复，且数字均连续，同时每个小格可以填入1～9任何值。其中需要注意的是，在对数独每行数字的连续性进行归约时，先得到的是DNF公式，需通过循环将其转换为相应CNF公式后再写入到CNF文件中；

1. 将变元文字信息转换成数独数字，得到数独终盘信息

void SudokuComplete(void);

求解数独盘CNF文件完毕后，将结果中的赋真值为1的文字依据坐标关系和数字转换为对应的数独数字并按顺序存储到数组table中，得到数独终盘的所有数字；

1. 绘制出数独终盘图像

void SudokuPrint(int \*tab);

参数：(int\*)tab：数独盘的名称

根据给定的数独盘名称在控制台中绘制出对应的蜂窝数独格局及内容；

1. 基于挖洞法创造一个新的随机数独题

status SudokuCreate(int num);

参数：(int)num：挖洞的数量

返回值：ERROR：文件打开失败

OK：挖洞完成，成功生成新随机数独

先从已有数独题库中随机挑选一道数独题，将其求解后得到对应的数独终盘，以该数独终盘为基础开始随机挖洞来创造一个新的数独题。

在挖够足够数量的洞前，使用while（）循环挖洞，每次挖洞后将数独盘对应位置的数字依次改为除原数字以外的其他各数字，并进行DPLL求解，若求解成功，说明本次挖洞后数独的解不唯一，本次挖洞失败，回溯到挖洞前状态进行重新挖洞；若均求解失败，说明挖洞后数独盘只有唯一解，本次挖洞成功，则继续挖下一个洞。当挖完足够多的洞后，新数独创建完成，得到的新数独盘原始信息存储在数组table中。

**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：12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz

机带RAM：16.0 GB

系统类型：64 位操作系统, 基于 x64 的处理器

1. 软件环境：

Windows11下Microsoft Visual Studio Community 2022 (64 位) 17.2.6

**4.1.2数据类型定义**

1. 宏定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define INCREASEMENT 100

1. 数据类型定义

typedef int status;

1. 全局变量

char filename[1000];//文件名

Root\* r;//CNF范式链表

ArgueValue\* ValueList;//变元表

int\* SortList;//变元出现次数排序数组，便于后续DPLL

int\* LiteralList;//记录DPLL中真值赋为1的文字

long Listsize;//记录LiteralList大小

int sum\_op;//记录总选择

int table0[62];//记录原始数独盘

int table[62];//记录数独盘

char tableself[20000];//记录用户输入数独盘

int count;//记录原数独盘中非0数字个数

1. 存储结构

/\*定义子句链表结点结构类型\*/

typedef struct Clause{

int literal;//记录子句中的文字

int flag;//标记该文字是否被删除，未删除时值为0，否则值为使之删除的变元序号以便于恢复

struct Clause \*nextl;//指向该子句中下一文字

struct Clause \*litline;//指向CNF范式链表中下一个具有相同文字的子句结点，用于回复与方便查找

}Clause;

/\*定义子句链表头结点结构类型\*/

typedef struct Paradigm{

int number;//子句中的文字个数，用于寻找单子句

int flag;//标记该子句是否被删除，未删除时值为0，否则值为使之删除的变元序号以便于恢复

struct Paradigm \*nextc;//指向CNF范式链表中下一子句头结点

struct Clause \*clause;//子句头指针

}Paradigm;

/\*定义CNF范式链表头结点类型\*/

typedef struct Root{

int lsize;//记录CNF范式链表中的文字数量

int psize;//记录CNF范式链表中的子句数量

Paradigm \*first;//指向第一个子句的指针

}Root;

/\*DPLL用到的两个结构Paraline和LitTrabverse，便于更快寻找正文字和负文字 \*/

/\*定义指向子句链表头结点的指针链表结点结构类型\*/

typedef struct Paraline{

Paradigm \*cline;//指向子句链表头结点

struct Paraline \*next;//指向下一链表结点

} Paraline;

/\*定义文字相关信息链表结构类型\*/

typedef struct LitTrabverse{

Paraline \*Tra\_c;//指向含有该文字的子句头结点链表的头结点

Clause \*Tra\_l;//指向该文字的文字结点

}LitTrabverse;

/\*定义存储变元信息的变元线性表结点结构类型\*/

typedef struct ArgueValue{

int Value;//变元的真值

int IsInit;//变元是否已被赋值

int Occur\_times;//变元在所有子句中出现的总次数

LitTrabverse Pos;//变元所有正文字的相关信息结构

LitTrabverse Neg;//变元所有负文字的相关信息结构

} ArgueValue;

**4.1.3函数声明以及函数调用关系**

1. CNF文件处理

/\*创建CNF范式链表及变元表\*/

status CreateParadigm(FILE \*\*fp);

/\*创建子句链表及文字链表\*/

int CreateClause(FILE \*\*fp,Clause \*\*clause,Paradigm \*ClausHead,int first);

/\*销毁CNF范式链表结构\*/

status DestroyParadigm(Root \*r);

/\*判断CNF范式链表中是否含有单子句\*/

int HasUnitClause(Root \*r);

/\*判断子句链表是否为单子句链表\*/

status isUnitClause(Paradigm \*p);

/\*遍历CNF范式邻接链表\*/

void ParadigmTrabverse(Root \*r);

/\*删除包含文字l（真值为1）的所有子句\*/

status DeleteClause(Root \*r,int l);

/\*在CNF范式链表表头添加含文字l的单子句链表结点\*/

status AddClause(Root \*r,int l);

/\*删除CNF范式链表中第一个含有文字l的单子句链表\*/

status RemoveHeadClaus(Root \*r,int l);

/\*删除文字-l（l为真值为1的文字）的所有子句链表结点\*/

status DeleteLiteral(Root \*r,int l);

/\*恢复假定文字l真值为1时对CNF范式链表的操作\*/

status Recover(Root \*r,int l);

/\*保存CNF范式公式的解和求解时间信息\*/

status SaveAnswer(ArgueValue \*ValueList,int solut,int time);

1. DPLL算法

/\*在CNF范式链表中取一个文字，用于DPLL\*/

/\*取每次DPLL处理后公式中出现次数最多的文字\*/

status FindLiteral1(Root \*r);

/\*取变元表中第一个未赋真值的变元的文字\*/

status FindLiteral2(Root \*r);

/\*取原公式中未赋真值的变元中出现次数最多的变元正文字

status FindLiteral3(Root \*r);

/\*DPLL流程控制\*/

status SAT(void);

/\*采用第一种变元选择策略的递归算法DPLL函数\*/

status DPLL1(int num,int times);

/\*采用第二种变元选择策略的递归算法DPLL函数\*/

status DPLL2(int num,int times);

/\*完善SAT求解结果\*/

void AnswerComplete(void);

/\*检查SAT求解结果是否正确\*/

status AnswerCheck(int solut);

函数功能联合：读取CNF文件，创建对应变元真值表和CNF范式链表后，选取任意一种选择变元方法，进入对应DPLL的函数进行求解。

1. Sudoku

/\*数独流程控制\*/

status Sudoku(void);

/\*对数独进行归约，创建数独问题转化为SAT问题后的CNF文件\*/

status SudokuCreateFile(void);/

/\*预处理数独文件，获取原始数独盘信息并创建数独CNF公式文件\*/

status SudokuPrepare(char\* filename, int op);

/\*将变元文字信息转换成数独数字，得到数独终盘信息\*/

void SudokuComplete(void);

/\*绘制出数独终盘图像\*/

void SudokuPrint(int \*tab);

/\*采用挖洞法创造一个新的随机数独题\*/

status SudokuCreate(int num);

**4.2系统测试**

**4.2.1交互系统展示**

主交互系统界面如图4.2.1-1所示，SAT交互系统界面如图4.2.1-2所示，数独游戏交互系统界面如图4.2.1-3，4.2.1-4和4.2.1-5所示：

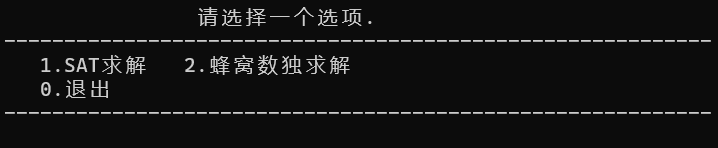


图4.2.1-1 主交互系统界面

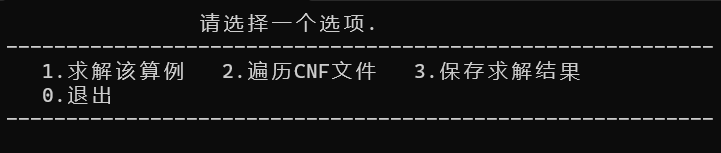


图4.2.1-2 SAT交互系统界面

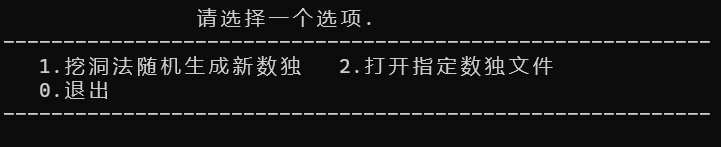


图4.2.1-3 数独游戏交互系统界面（1）

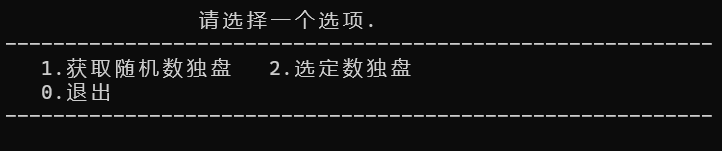


图4.2.1-4 数独游戏交互系统界面（2）

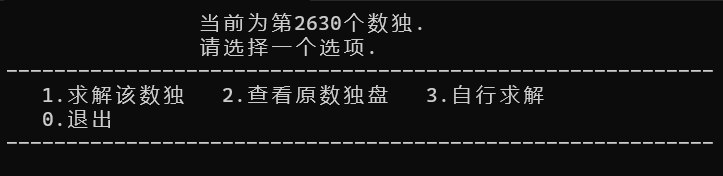


图4.2.1-5 数独游戏交互系统界面（3）

**4.2.2 CNF文件处理模块测试和DPLL算法模块测试**

过程简要介绍：在主交互系统界面选择1.SAT求解，并输入想要求解的CNF文件名；进入SAT求解界面；选择1.求解该算例，有优化算法，优化前算法两个选项，求解出来后会打印出真值；求解完成后可以选择遍历CNF文件或者保存求解结果到与CNF文件同名.res文件中；

1. 测试算例1：SAT测试备选算例\基准算例\功能测试\sat-20.cnf；先读入所要求解的CNF文件名（如图4.2.2-1所示），使用优化算法和优化前算法求解测试算例，结果分别如图4.2.2-2，图4.2.2-3所示，遍历测试算例CNF文件，保存SAT的求解结果，遍历结果和保存结果分别如图4.2.2-4，图4.2.2-5所示（接下来测试算例CNF文件的遍历结果将省略）：

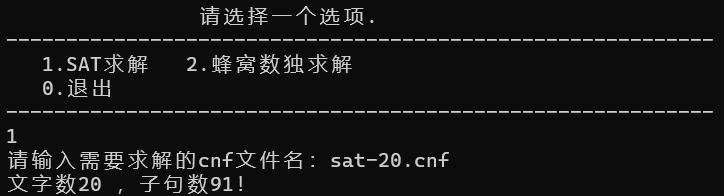


图4.2.2-1 读入所要求解的CNF文件名

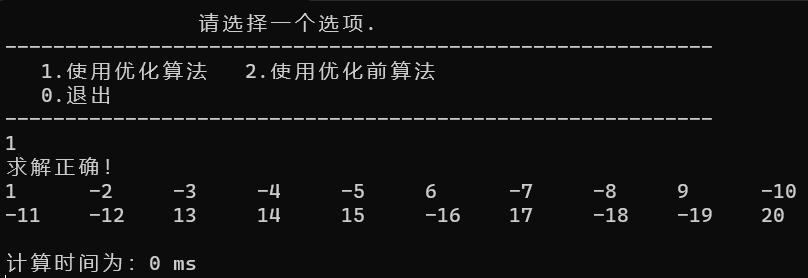


图4.2.2-2 使用优化算法的SAT求解

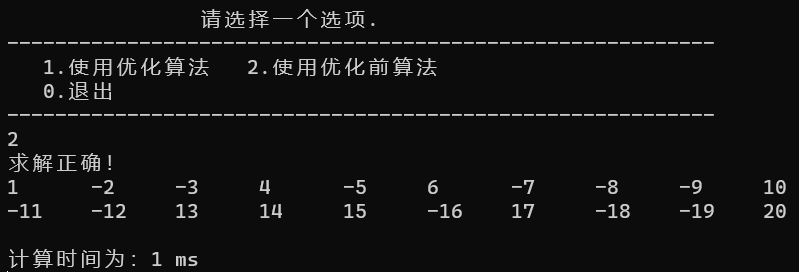


图4.2.2-3 使用优化前算法的SAT求解

优化率：；

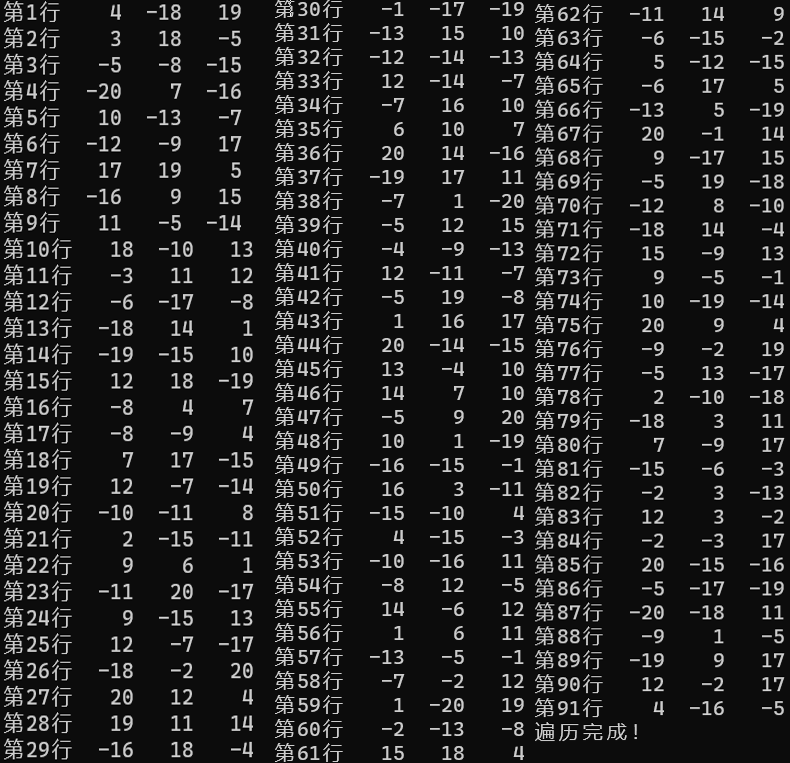


图4.2.2-4 CNF文件遍历结果

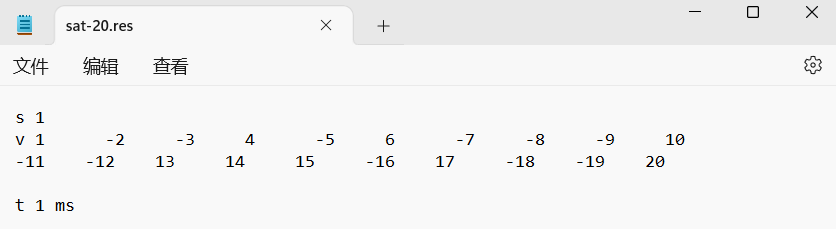


图4.2.2-5 SAT求解保存结果

1. 测试算例2：SAT测试备选算例\其他可供选择的算例\tst\tst\_v10\_c100.cnf；测试结果如图所示（接下来测试算例SAT求解保存结果将省略）：

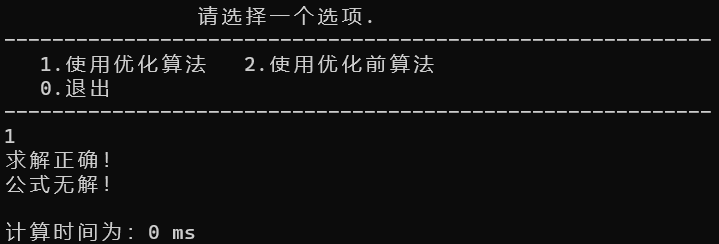


图4.2.2-6 使用优化算法的SAT求解

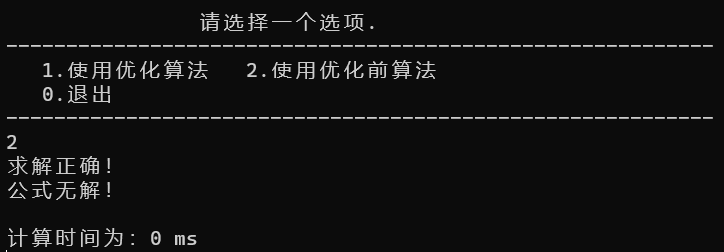


图4.2.2-7 使用优化前算法的SAT求解

优化率：0；

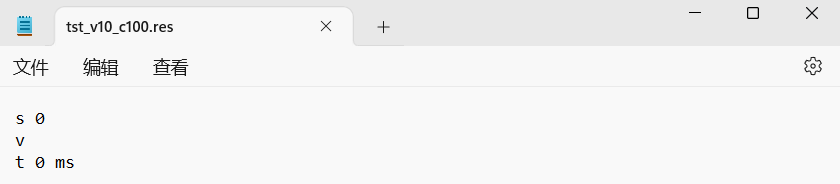


图4.2.2-8 SAT求解保存结果

1. 测试算例3：SAT测试备选算例\其它可供选择使用的算例\ais\ais6.cnf；测试结果如图所示：

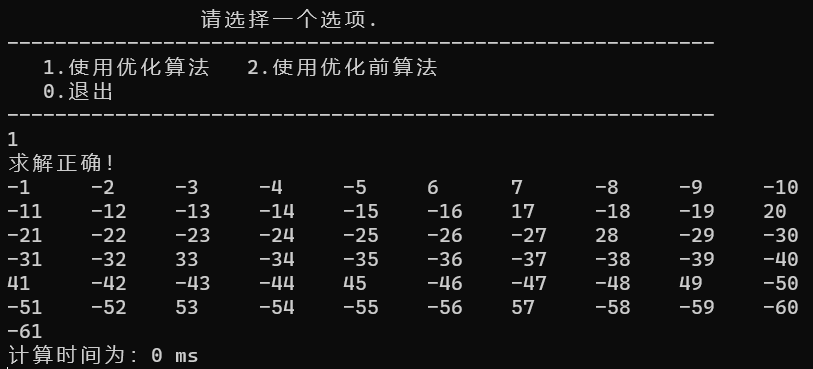


图4.2.2-9 使用优化算法的SAT求解

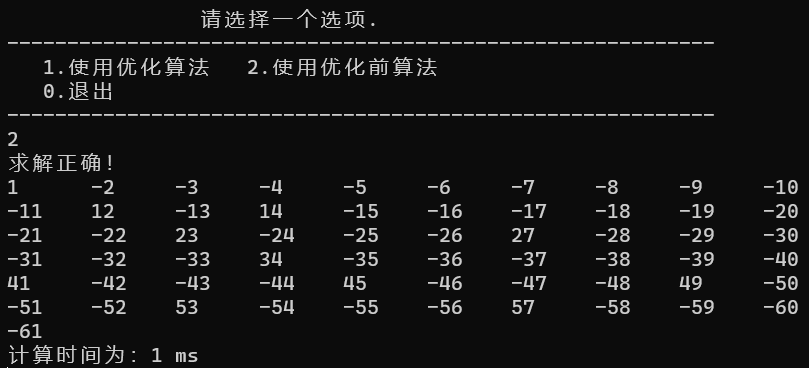


图4.2.2-10 使用优化前算法的SAT求解

优化率：

1. 测试算例4：SAT测试备选算例\其他可供选择的算例\tst\flat30-1.cnf；测试结果如图所示：

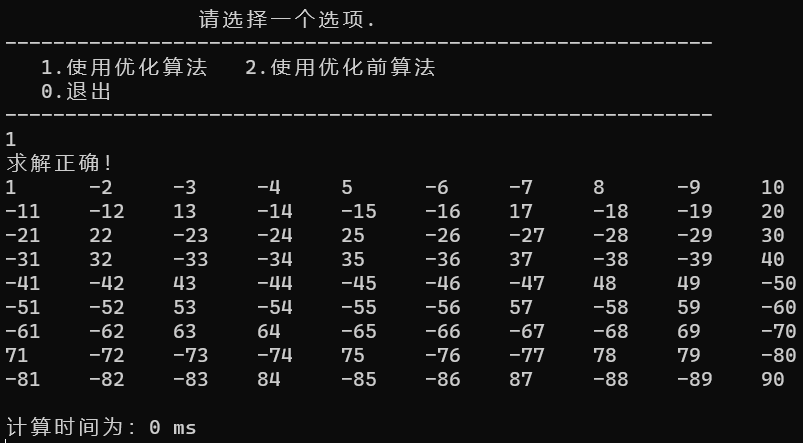


图4.2.2-11 使用优化算法的SAT求解

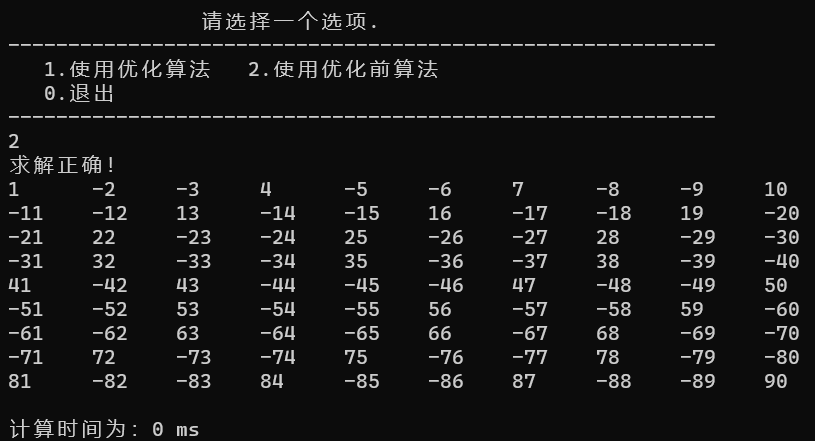


图4.2.2-12 使用优化前算法的SAT求解

优化率：0；

1. 测试算例5：SAT测试备选算例\其他可供选择的算例\tst\flat30-99.cnf；测试结果如图所示：

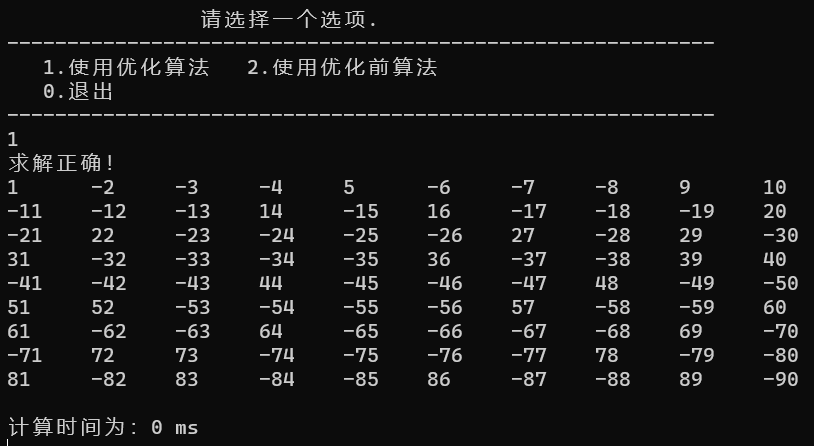


图4.2.2-13 使用优化算法的SAT求解

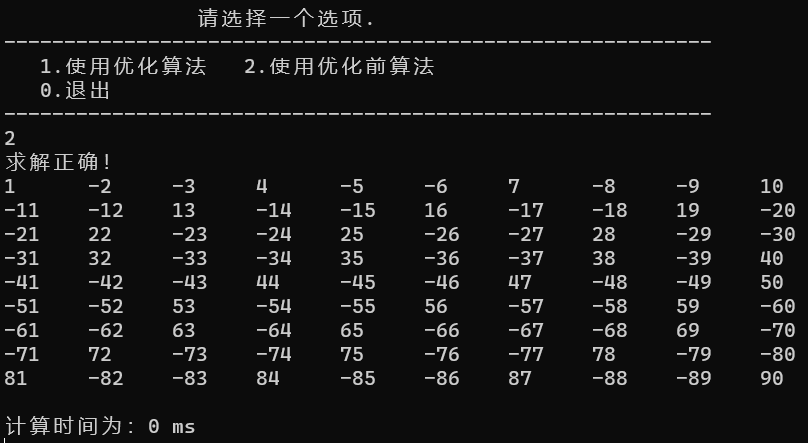


图4.2.2-14 使用优化前算法的SAT求解

优化率：0；

1. 测试算例6：SAT测试备选算例\其他可供选择的算例\tst\tst\_v100\_c400.cnf；测试结果如图所示：

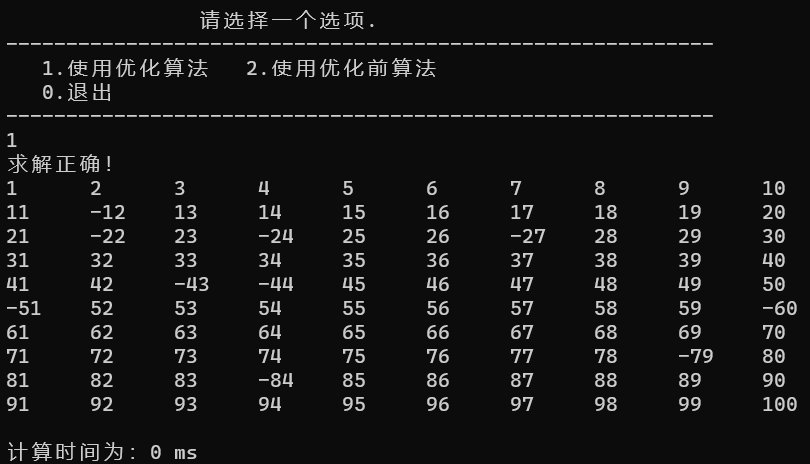


图4.2.2-15 使用优化算法的SAT求解

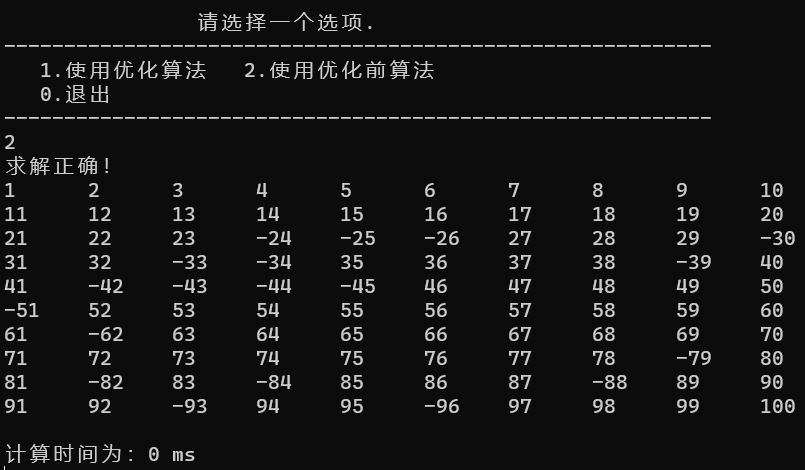


图4.2.2-16 使用优化前算法的SAT求解

优化率：0；

(由于测试算例过于简单，几乎看不出优化前后在求解用时上的差别，接下来使用几组稍微复杂的算例进行测试)

1. 测试算例7：SAT测试备选算例\其它可供选择使用的算例\tst\tst\_v100\_c425.cnf；测试结果如图所示：

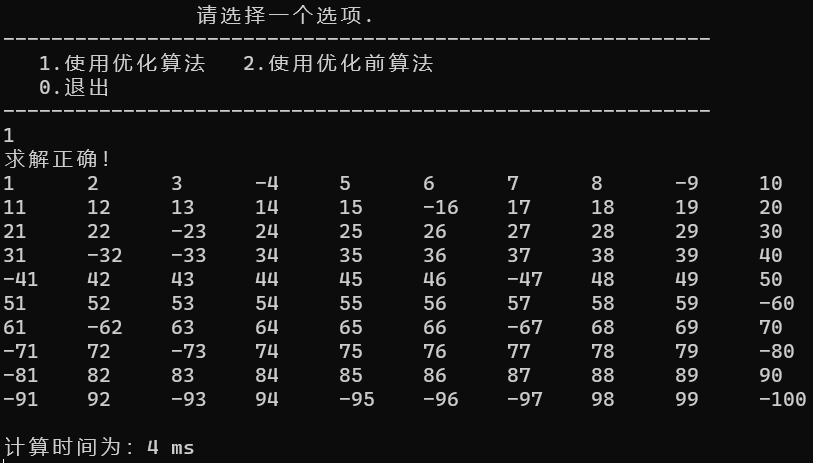


图4.2.2-17 使用优化算法的SAT求解

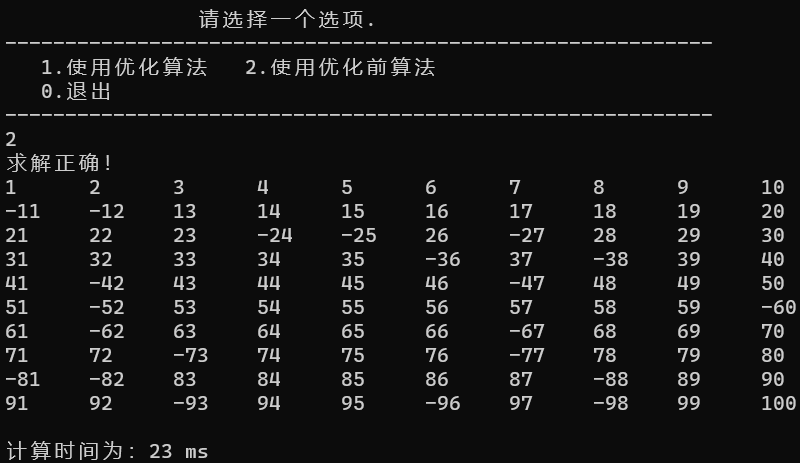


图4.2.2-18 使用优化前算法的SAT求解

优化率：

1. 测试算例8：SAT测试备选算例\其它可供选择使用的算例\ais\ais8.cnf；测试结果如图所示：

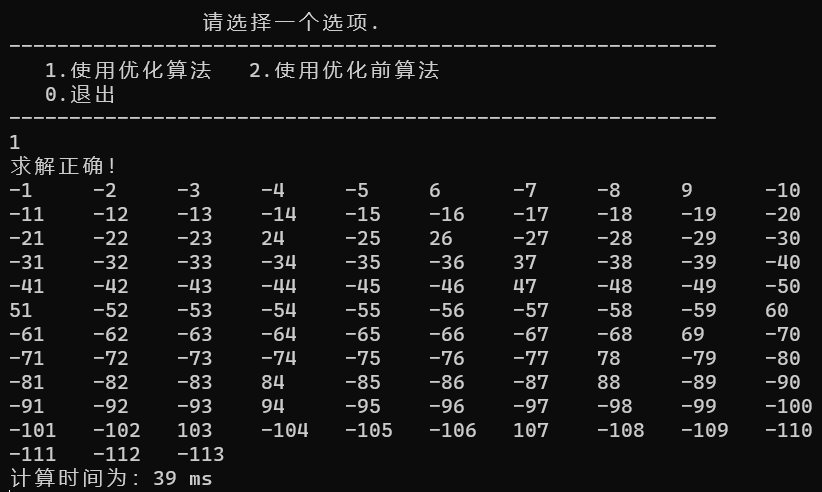


图4.2.2-19 使用优化算法的SAT求解

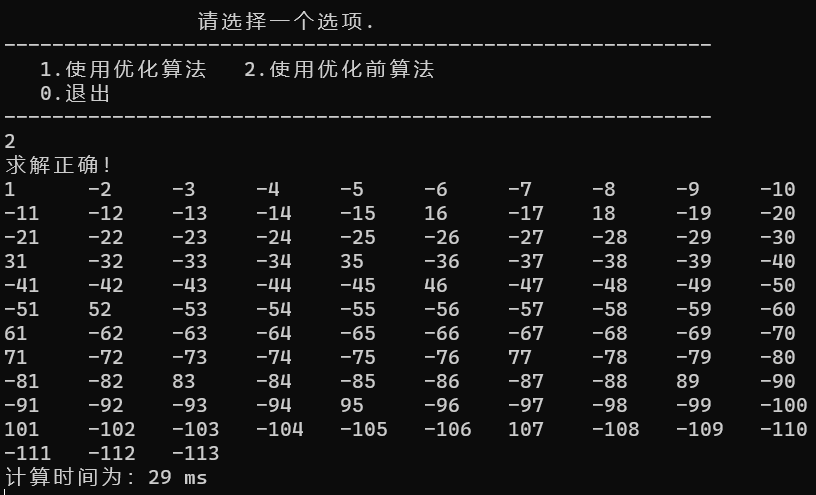


图4.2.2-20 使用优化前算法的SAT求解

优化率：

(由于算例具有一定的特殊性，同一算法对不同算例的求解效果不完全相同，对于优化前后算法部分算例亦可能出现负优化的情况)

1. 测试算例9：SAT测试备选算例\基准算例\性能算例\ais10.cnf；测试结果如图所示：

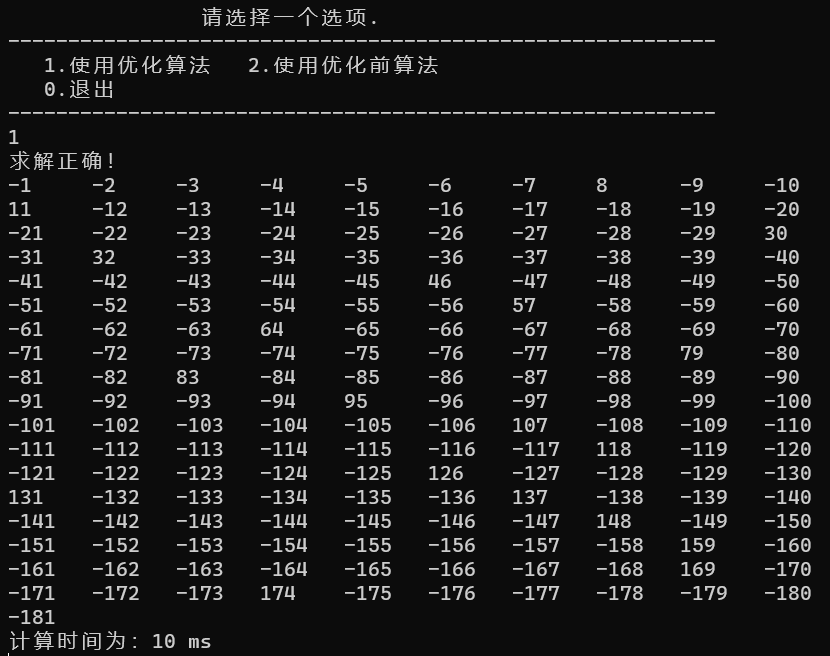


图4.2.2-21 使用优化算法的SAT求解

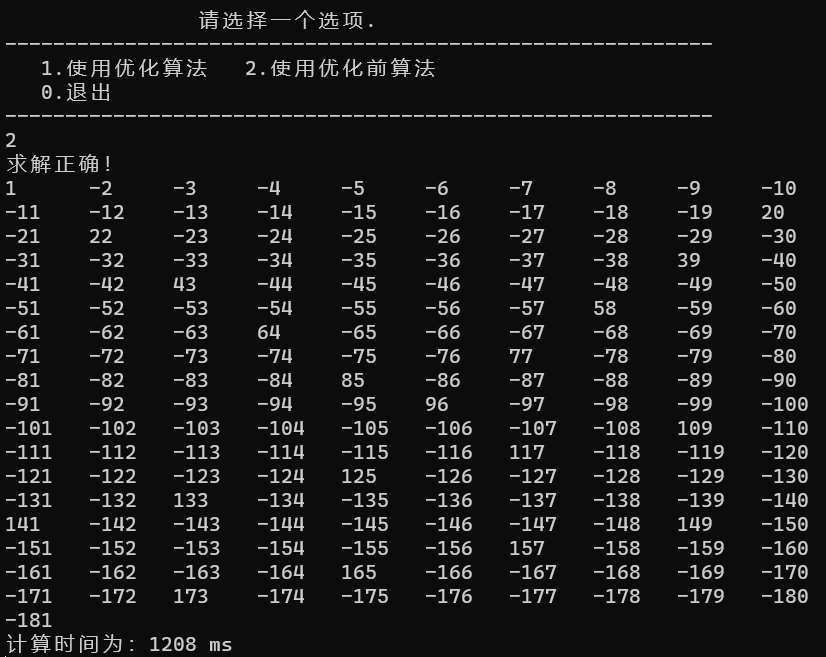


图4.2.2-22 使用优化前算法的SAT求解

优化率：

1. 测试算例10：SAT测试备选算例\其它可供选择使用的算例\tst\tst\_v200\_c220.cnf；测试结果如图所示：

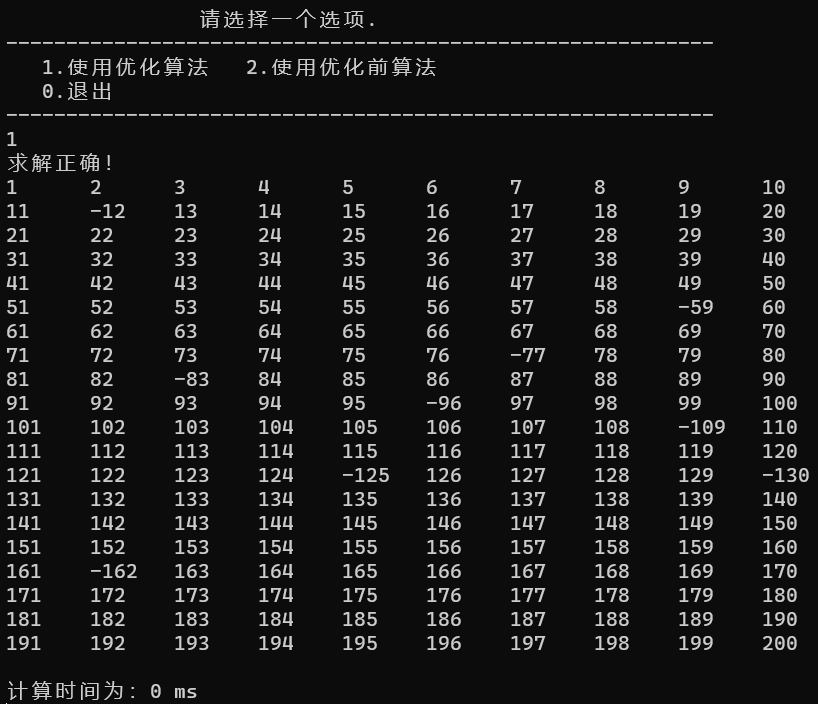


图4.2.2-23 使用优化算法的SAT求解

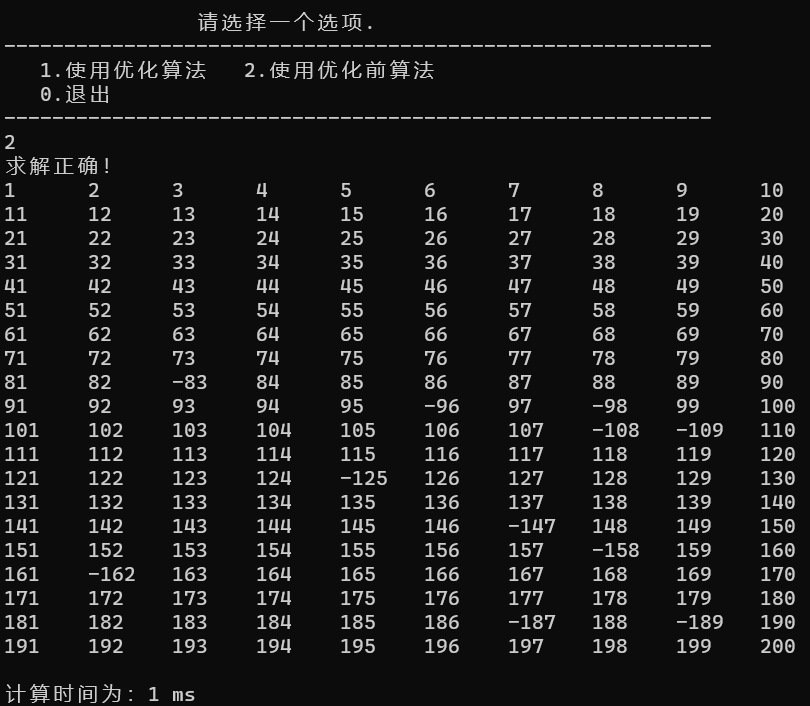


图4.2.2-24 使用优化前算法的SAT求解

优化率：

1. 测试算例11：SAT测试备选算例\满足算例\M\sud00012.cnf；测试结果如图所示：

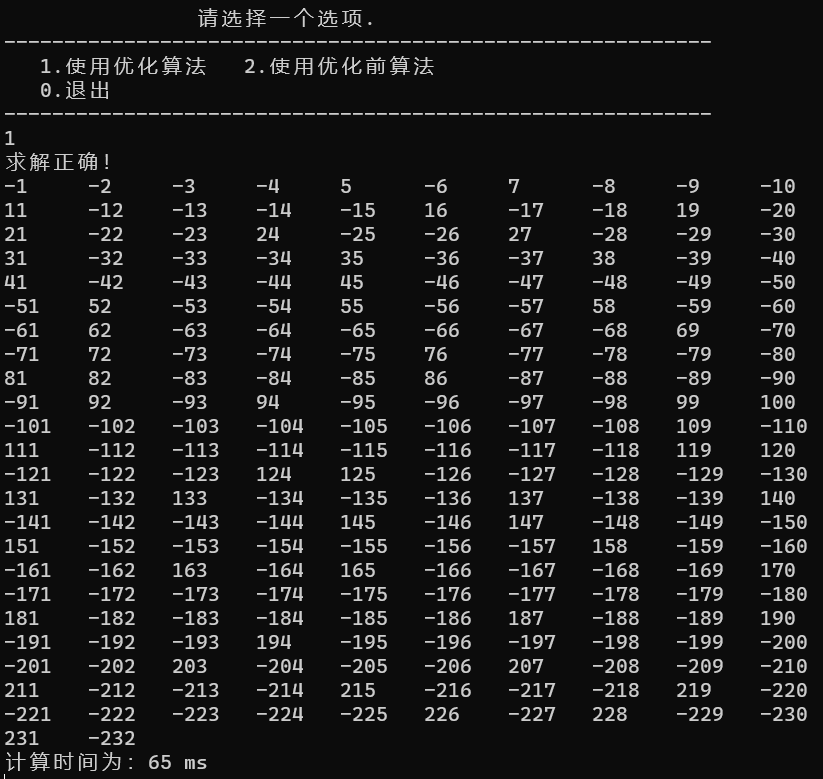


图4.2.2-25 使用优化算法的SAT求解

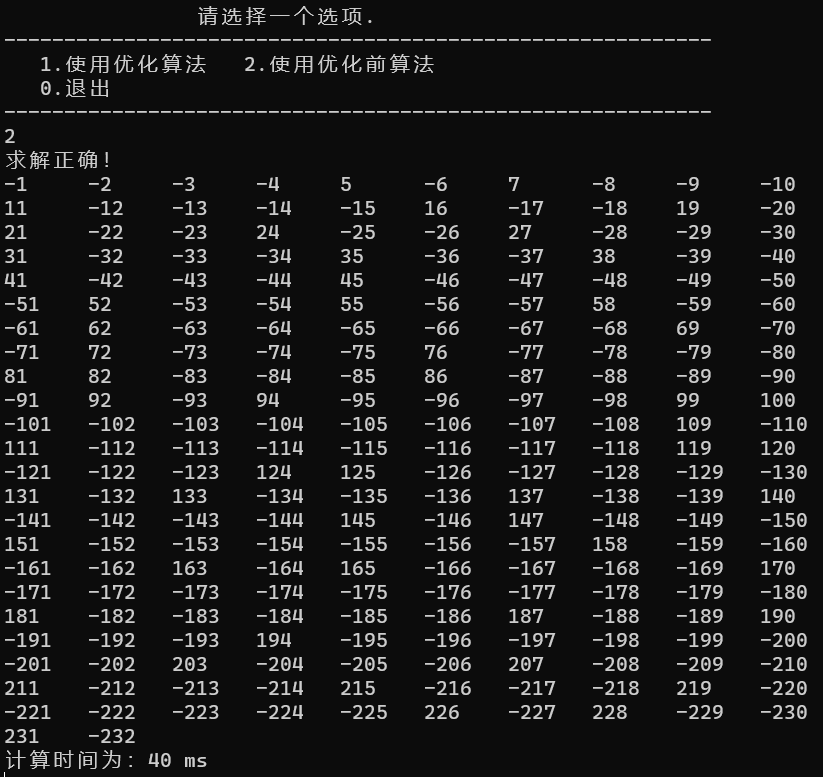


图4.2.2-26 使用优化前算法的SAT求解

优化率：

1. 测试算例12：SAT测试备选算例\其他可供选择的算例\ais\ais12.cnf；测试结果如图所示：

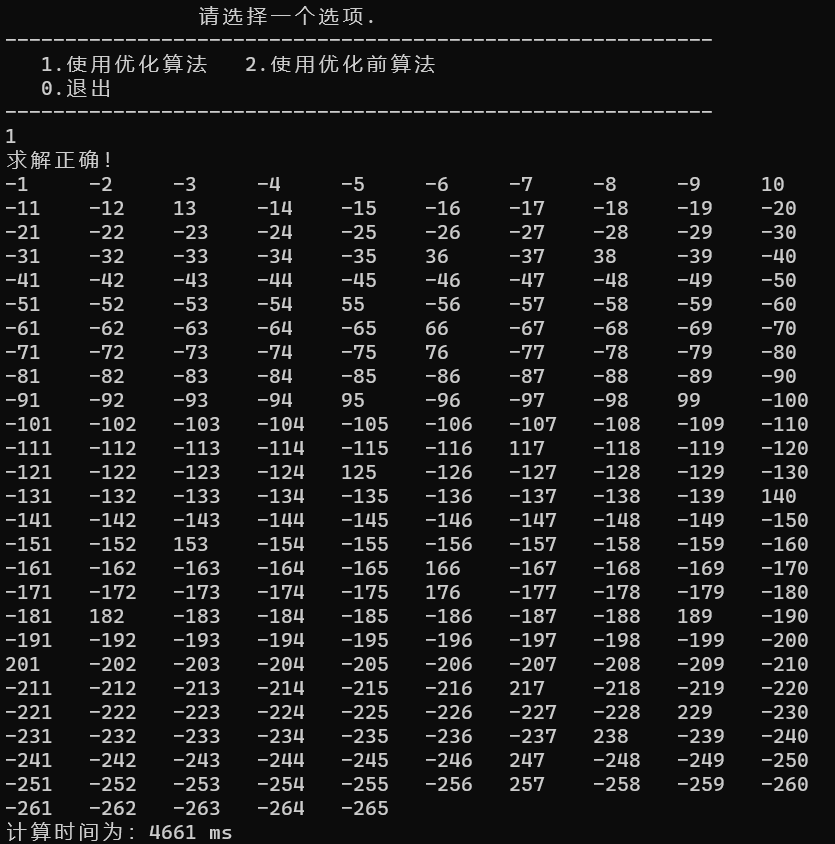


图4.2.2-27 使用优化算法的SAT求解

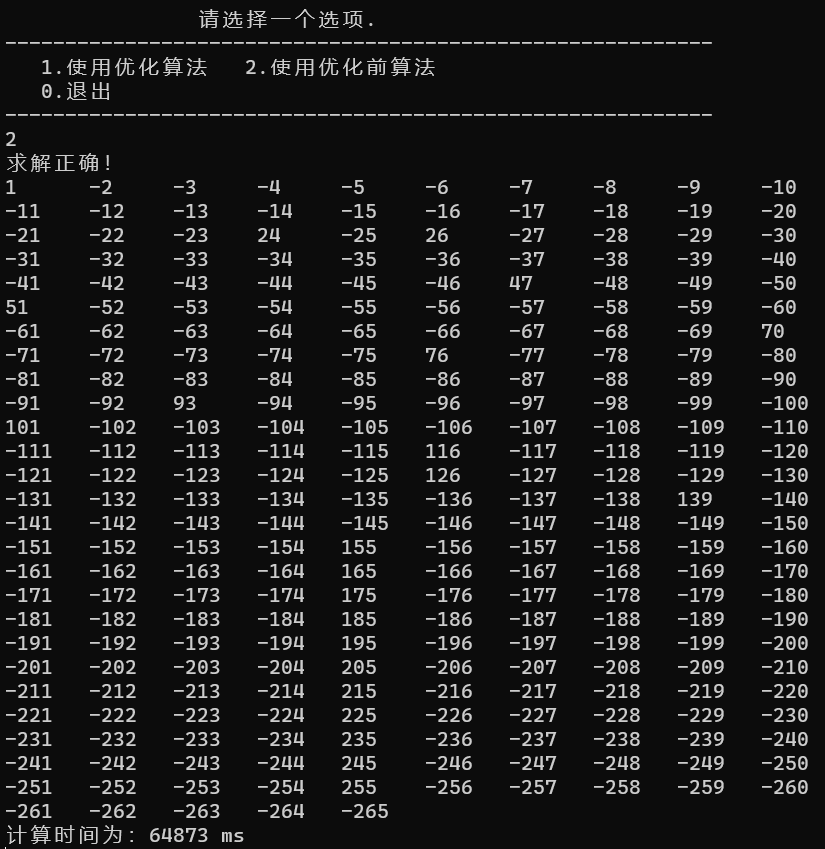


图4.2.2-28 使用优化前算法的SAT求解

优化率：

1. 测试算例13：SAT测试备选算例\满足算例\M\sud00001.cnf；测试结果如图所示：

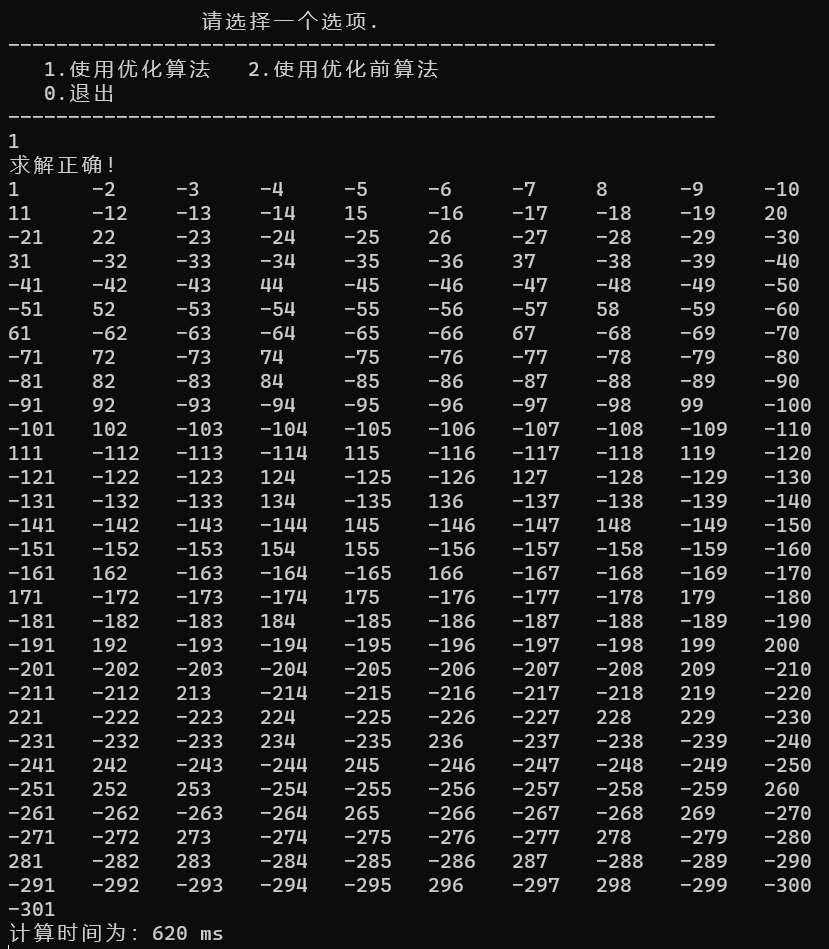


图4.2.2-29 使用优化算法的SAT求解

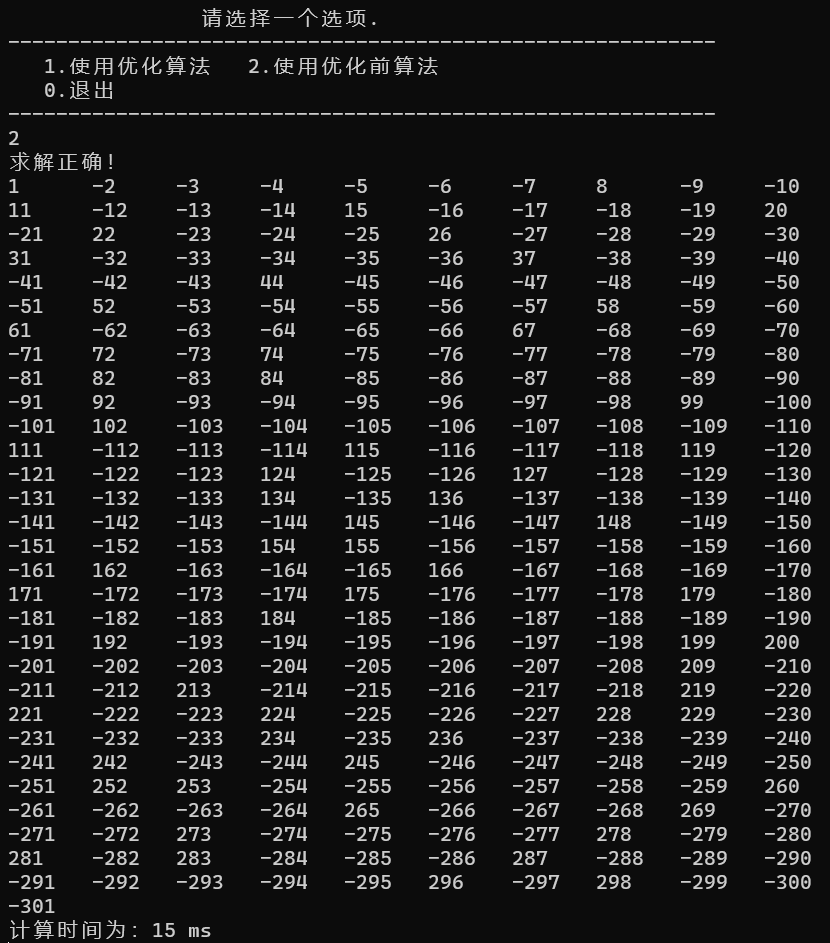


图4.2.2-30 使用优化前算法的SAT求解

优化率：

1. 测试算例14：SAT测试备选算例\其他可供选择的算例\tst\sw100-1.cnf；测试结果如图所示：

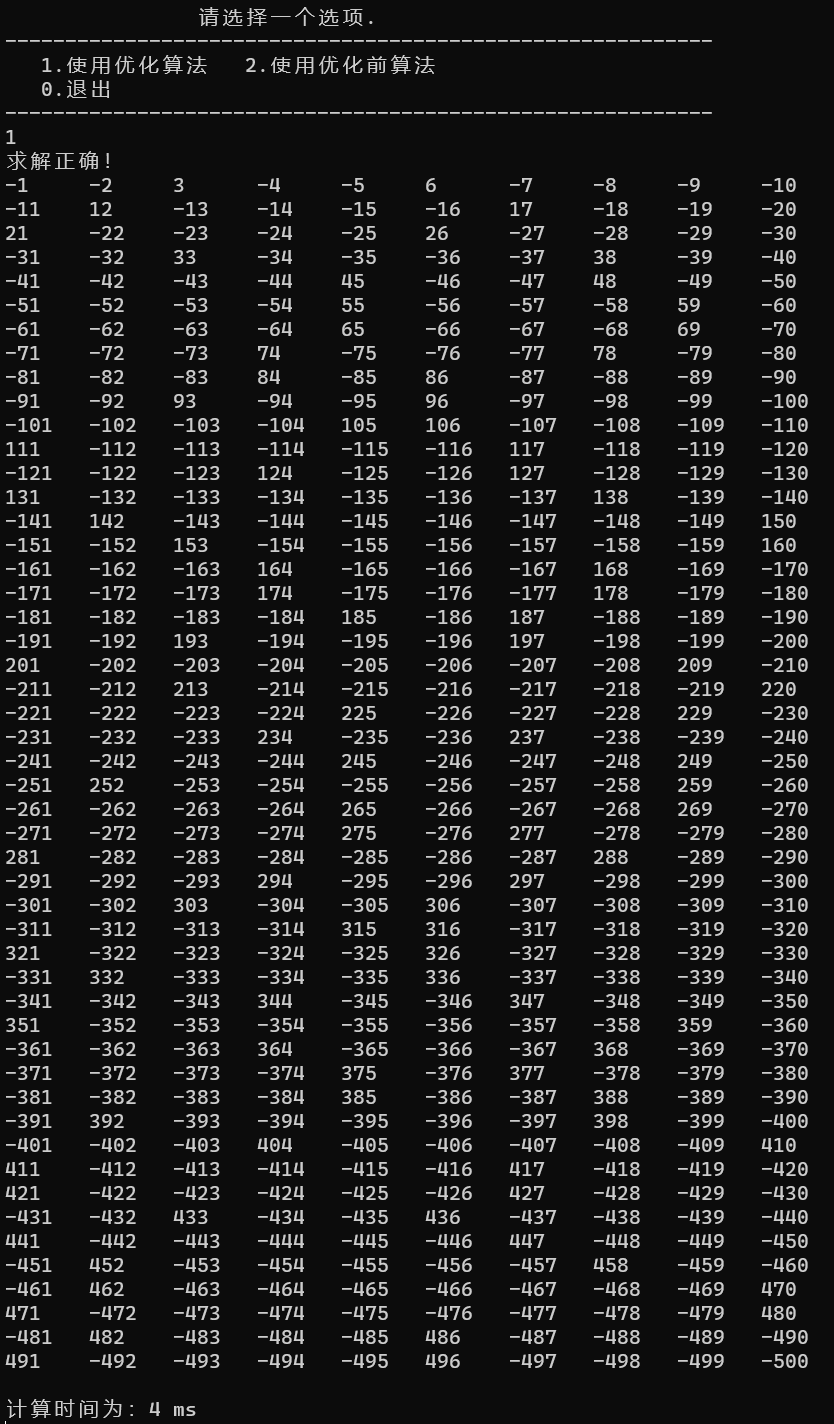


图4.2.2-31 使用优化算法的SAT求解

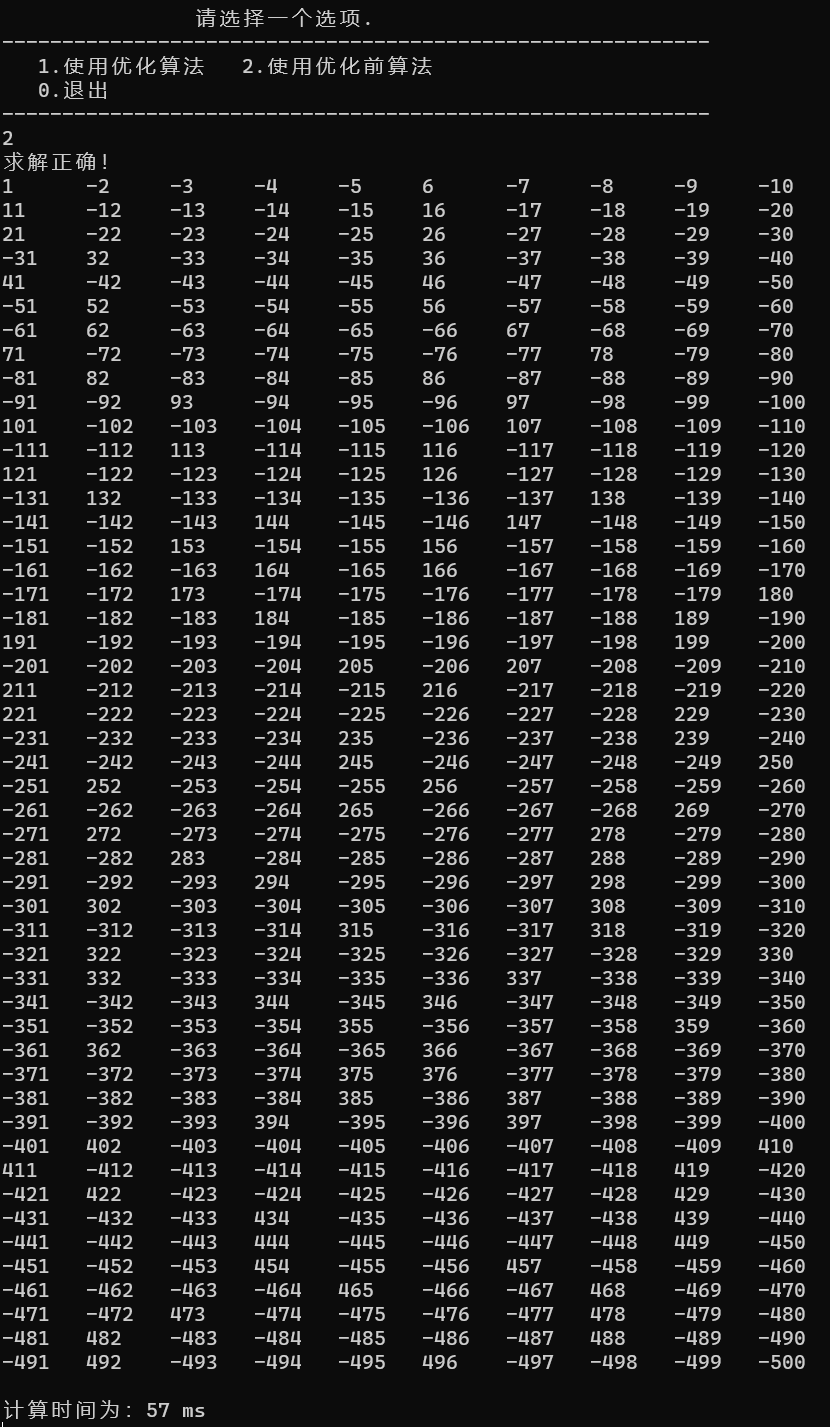


图4.2.2-32 使用优化前算法的SAT求解

优化率：

1. 测试算例15：SAT测试备选算例\其它可供选择使用的算例\tst\sw100-70.cnf；测试结果如图所示：

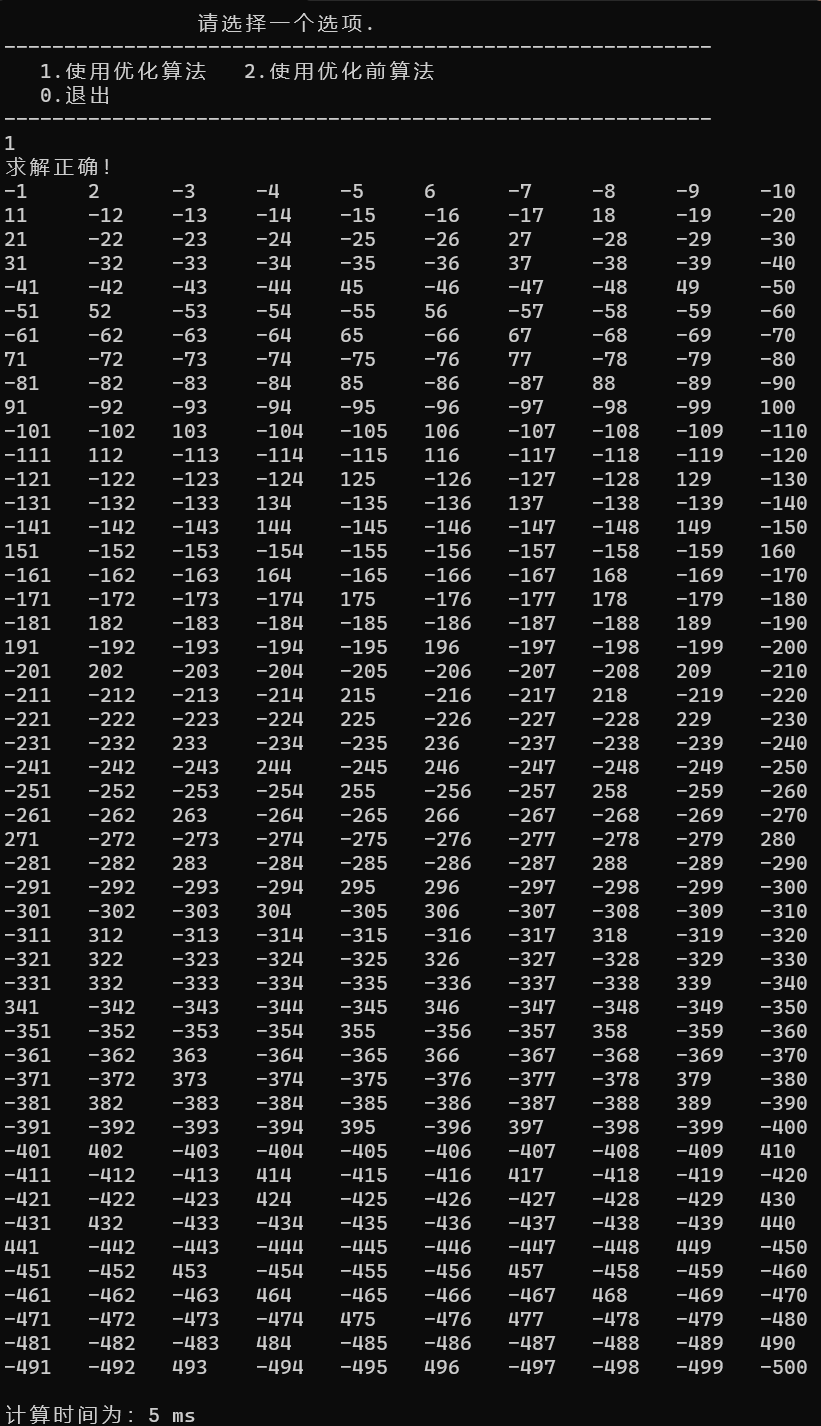


图4.2.2-33 使用优化算法的SAT求解

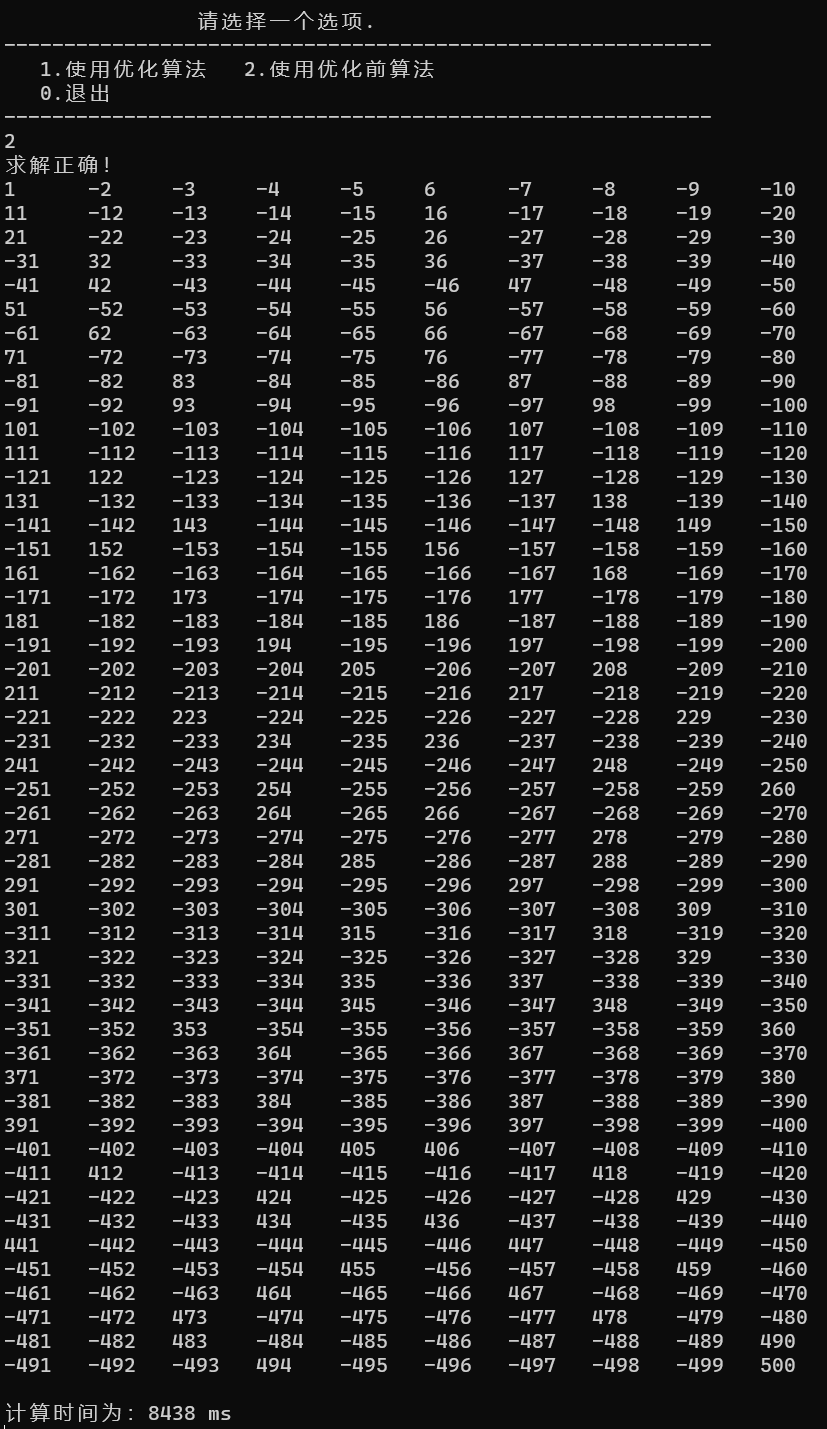


图4.2.2-34 使用优化前算法的SAT求解

优化率：

1. 测试算例16：SAT测试备选算例\其它可供选择使用的算例\tst\qg7-09.cnf；测试结果如图所示：

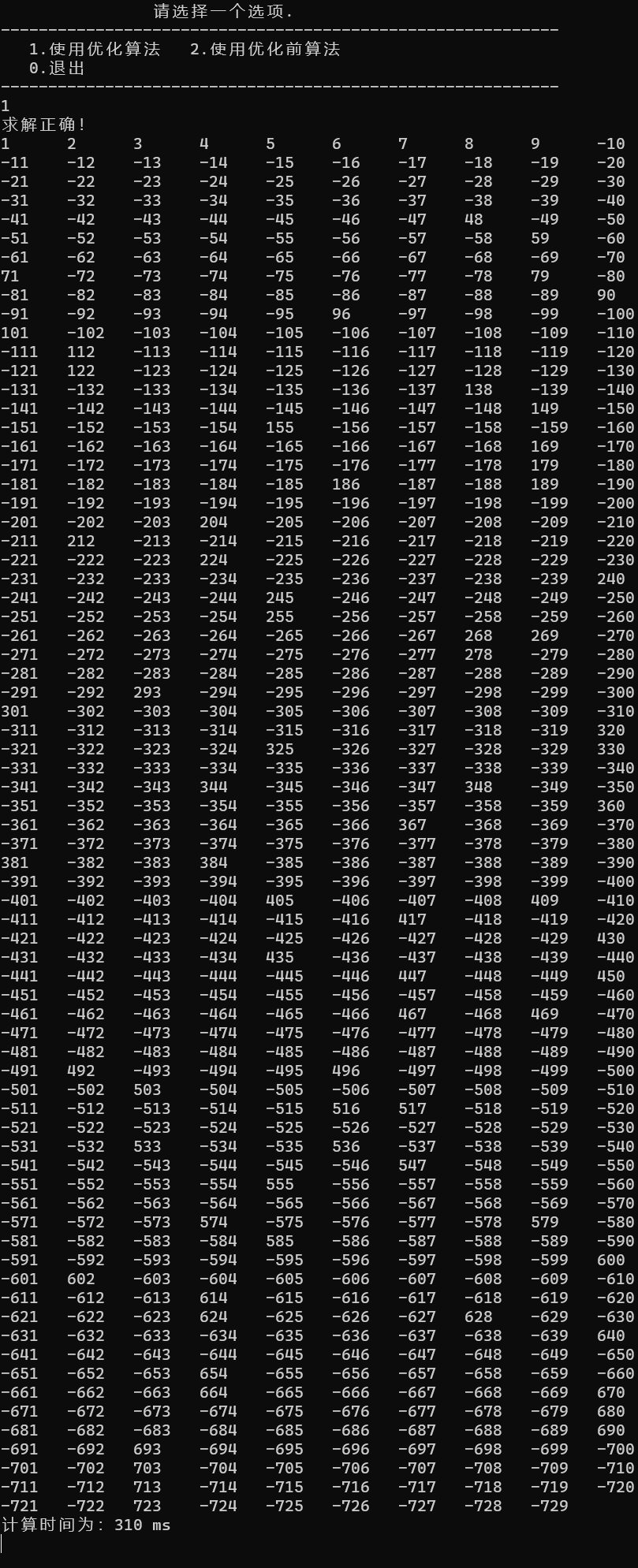


图4.2.2-35使用优化算法的SAT求解

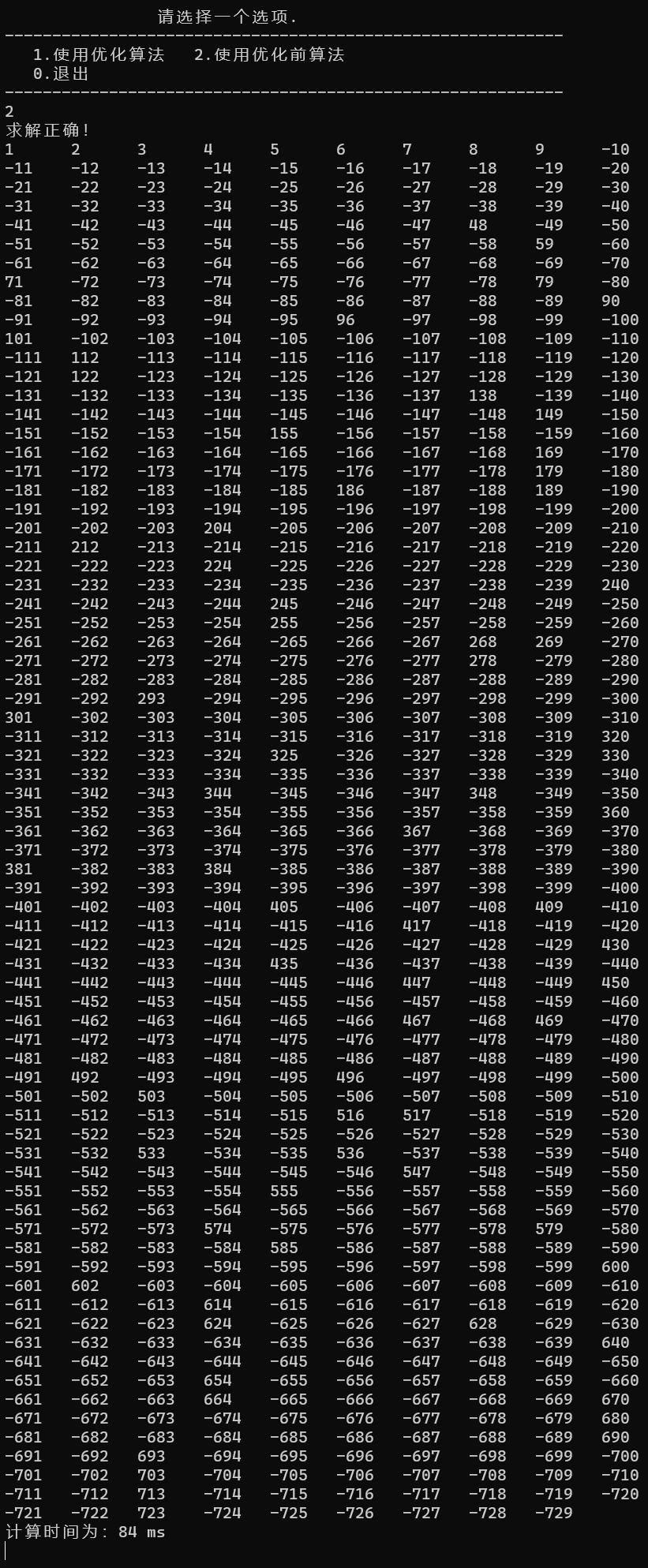


图4.2.2-36 使用优化前算法的SAT求解

优化率：

1. 测试算例17：SAT测试备选算例\不满足算例\u-5cnf\_3500\_3500\_30f1.shuffled-30.cnf；测试结果如图所示：

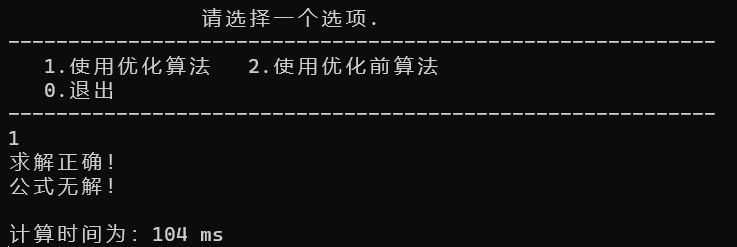


图4.2.2-37 使用优化算法的SAT求解

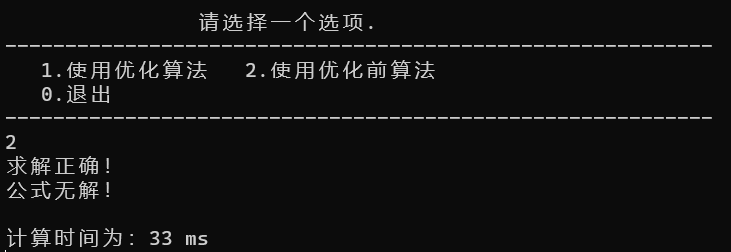


图4.2.2-38 使用优化前算法的SAT求解

优化率：

1. 测试算例18：SAT测试备选算例\其它可供选择使用的算例\tst\qg4-08.cnf；测试结果如图所示：

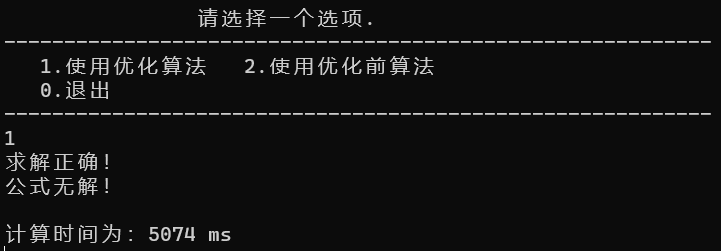


图4.2.2-39 使用优化算法的SAT求解

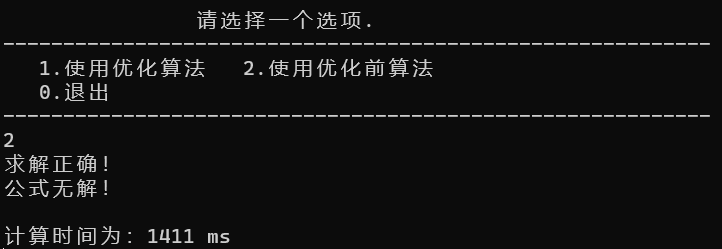


图4.2.2-40 使用优化前算法的SAT求解

优化率：

**4.2.3 CNF算例测试总结表格**

算例测试总结如下表4-1所示：

**表4-1 CNF算例测试总结表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| SAT测试备选算例\基准算例\功能测试\sat-20.cnf | 20 | 91 | 0.22 | 满足 | 优化后：0ms  优化前：1ms | 100.0% |
| SAT测试备选算例\其他可供选择的算例\tst\tst\_v10\_c100.cnf | 10 | 100 | 0.10 | 不满足 | 优化后：0ms  优化前：0ms | 0 |
| SAT测试备选算例\其它可供选择使用的算例\ais\ais6.cnf | 61 | 581 | 0.10 | 满足 | 优化后：0ms  优化前：1ms | 100.0% |
| SAT测试备选算例\其他可供选择的算例\tst\flat30-1.cnf | 90 | 300 | 0.30 | 满足 | 优化后：0ms  优化前：0ms | 0 |
| SAT测试备选算例\其他可供选择的算例\tst\flat30-99.cnf | 90 | 300 | 0.30 | 满足 | 优化后：0ms  优化前：0ms | 0 |
| SAT测试备选算例\其他可供选择的算例\tst\tst\_v100\_c400.cnf | 100 | 400 | 0.25 | 满足 | 优化后：0ms  优化前：0ms | 0 |
| SAT测试备选算例\其它可供选择使用的算例\tst\tst\_v100\_c425.cnf | 100 | 425 | 0.24 | 满足 | 优化后：4ms  优化前：23ms | 82.6% |
| SAT测试备选算例\其它可供选择使用的算例\ais\ais8.cnf | 113 | 1520 | 0.07 | 满足 | 优化后：39ms  优化前：29ms | -34.5% |
| SAT测试备选算例\基准算例\性能算例\ais10.cnf | 181 | 3151 | 0.06 | 满足 | 优化后：10ms  优化前：1208ms | 99.2% |
| SAT测试备选算例\其它可供选择使用的算例\tst\tst\_v200\_c220.cnf | 200 | 200 | 1.00 | 满足 | 优化后：0ms  优化前：1ms | 100.0% |
| SAT测试备选算例\满足算例\M\sud00012.cnf | 232 | 1901 | 0.12 | 满足 | 优化后：65ms  优化前：40ms | -62.5% |
| SAT测试备选算例\其他可供选择的算例\ais\ais12.cnf | 265 | 5666 | 0.05 | 满足 | 优化后：4661ms  优化前：64873ms | 92.8% |
| SAT测试备选算例\满足算例\M\sud00001.cnf | 301 | 2780 | 0.11 | 满足 | 优化后：620ms  优化前：15ms | -4033.3% |
| SAT测试备选算例\其他可供选择的算例\tst\sw100-1.cnf | 500 | 3100 | 0.16 | 满足 | 优化后：4ms  优化前：57ms | 93.0% |
| SAT测试备选算例\其它可供选择使用的算例\tst\sw100-70.cnf | 500 | 3100 | 0.16 | 满足 | 优化后：5ms  优化前：8438ms | 99.9% |
| SAT测试备选算例\其它可供选择使用的算例\tst\qg7-09.cnf | 729 | 22060 | 0.03 | 不满足 | 优化后：310ms  优化前：84ms | -269.0% |
| SAT测试备选算例\不满足算例\u-5cnf\_3500\_3500\_30f1.shuffled-30.cnf | 30 | 420 | 0.07 | 不满足 | 优化后：104ms  优化前：33ms | -215.2% |
| SAT测试备选算例\其它可供选择使用的算例\tst\qg4-08.cnf | 512 | 9685 | 0.05 | 不满足 | 优化后：5074ms  优化前：1411ms | -259.6% |

**4.2.4蜂窝数独游戏生成与求解测试**

数独部分简要介绍：数独部分分为两个功能，一是基与挖洞法随机生成数独游戏盘，并且通过控制挖洞数量可选择生成不同难度的数独盘；二是读入指定数独文件（.txt格式），随机选取或自行选定数独题序号来获取已知数独盘；将生成或读取的原始数独盘进行归约，转化为CNF文件,再DPLL求解CNF文件，最终将求解结果转换为数独数字打印到屏幕上显示。

1. 基于挖洞法随机生成新数独，可基于挖洞数量选择难度,生成完毕后可对数独盘进行查看和求解，如图所示：

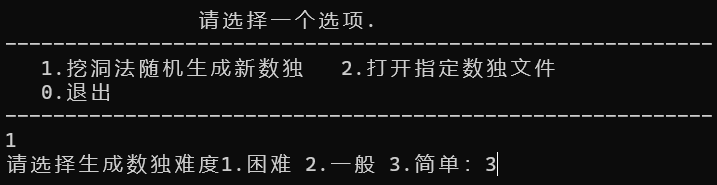


图4.2.4-1 选择随机生成数独的难度

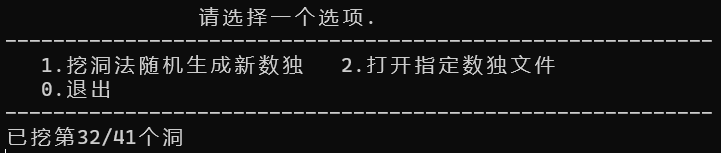


图4.2.4-2 数独挖洞过程

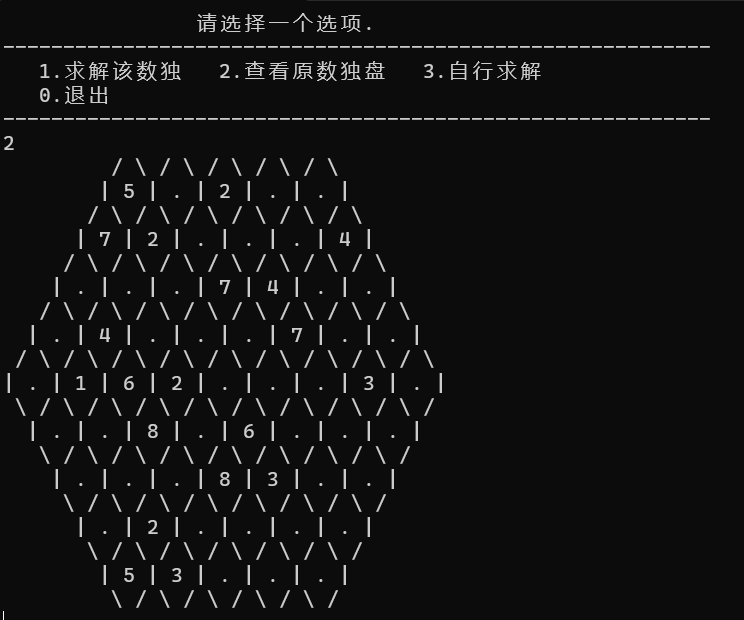


图4.2.4-3 查看随机生成的新数独原盘

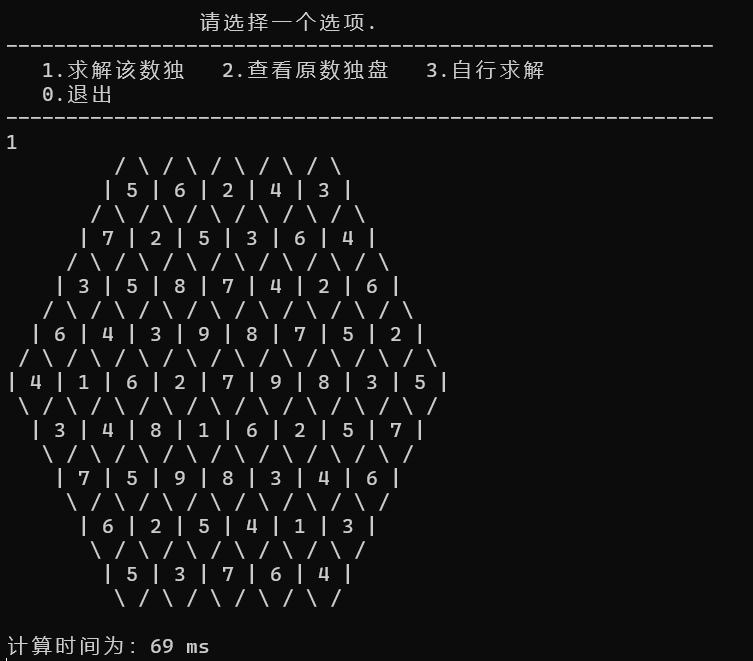


图4.2.4-4 对随机生成的新数独盘进行求解

接下来选择生成不同难度的数独盘再次进行测试，如图所示：

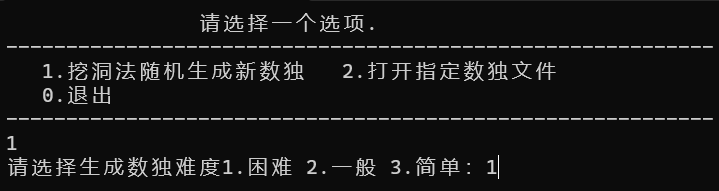


图4.2.4-5 选择随机生成数独的难度

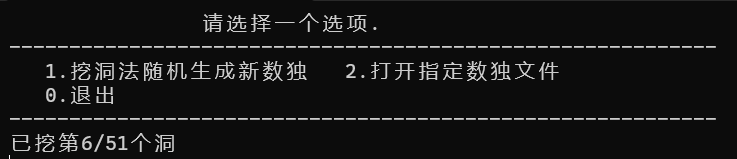


图4.2.4-6 数独挖洞过程

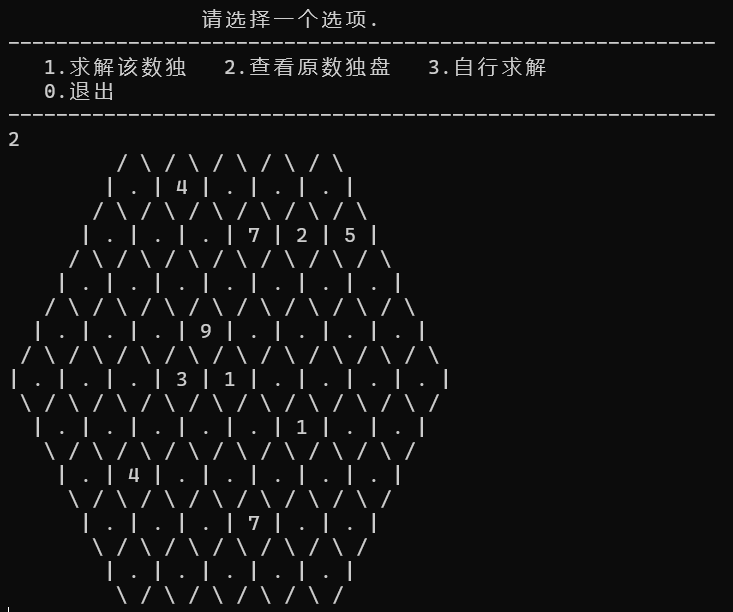


图4.2.4-7 查看随机生成的新数独原盘

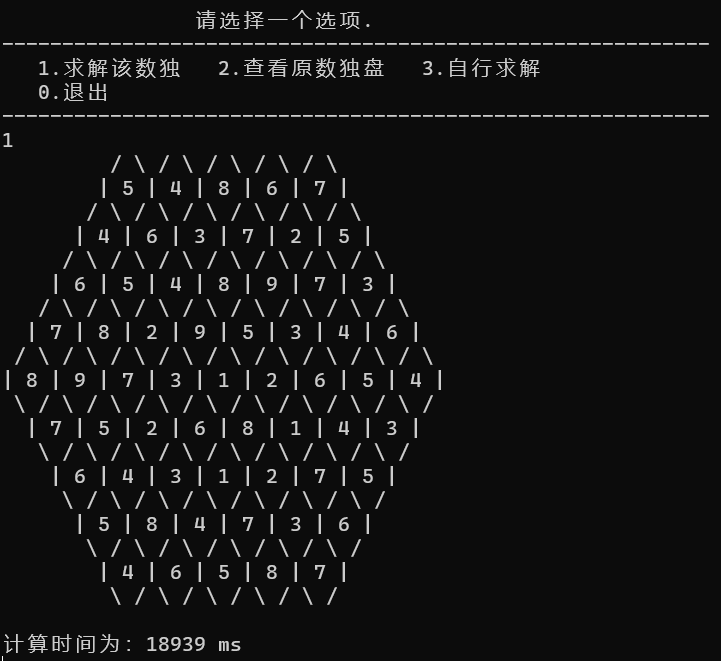


图4.2.4-8 对随机生成的新数独盘进行求解

1. 读入指定数独文件（.txt格式），随机选取或自行选定数独题序号来获取已知数独盘，然后可对数独盘进行查看和求解，也可以自行输入解答并与正确答案进行比对，具有一定的互动性，如图所示：

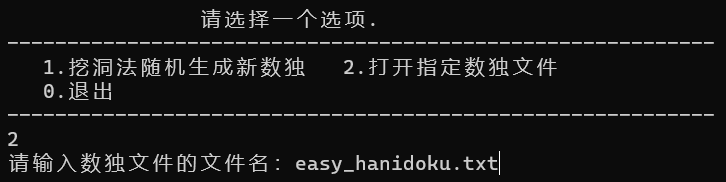


图4.2.4-9 读入指定数独文件

数独文件内容格式要求是：每行为一道数独题，以HQV1G开头，后面有61个字符，按照数独盘上的顺序未知位置填入0，已知位置填入1～9，以回车作为结尾，例如：

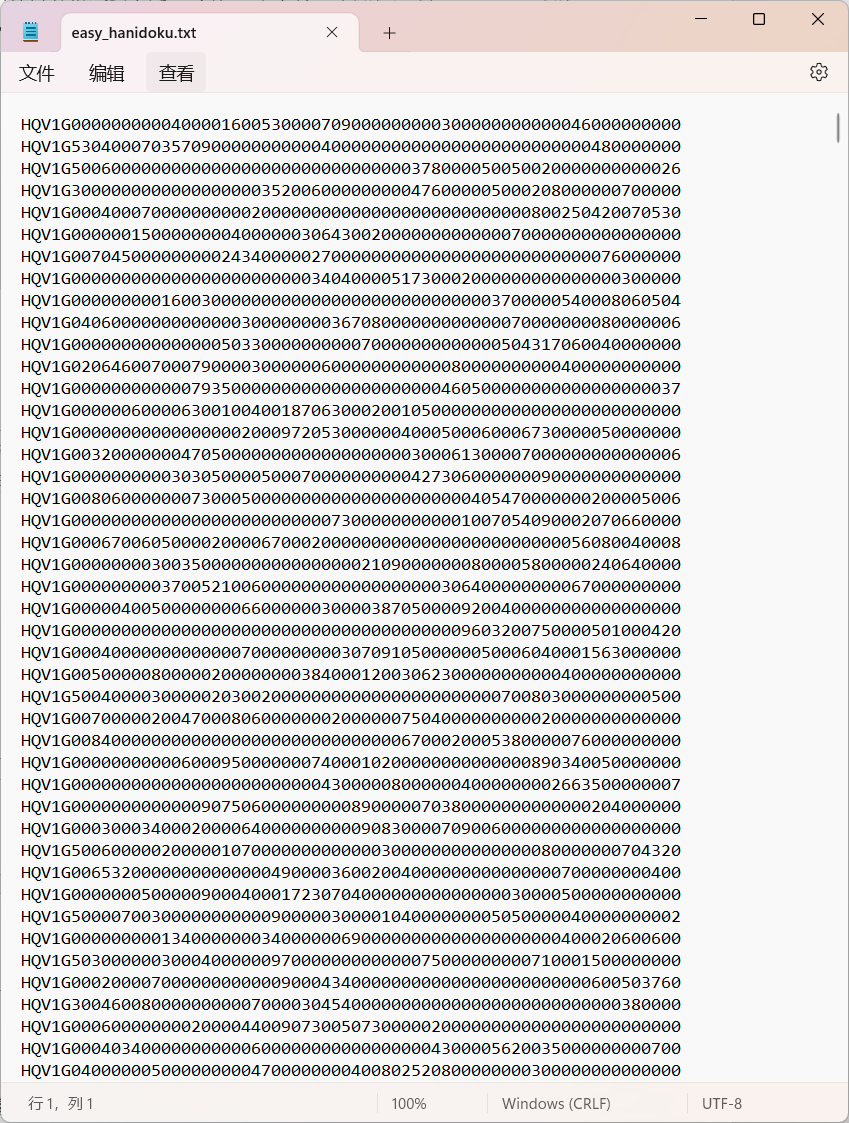


图4.2.4-10数独文件示例

从指定数独文件中获取随机数独盘，如图所示：

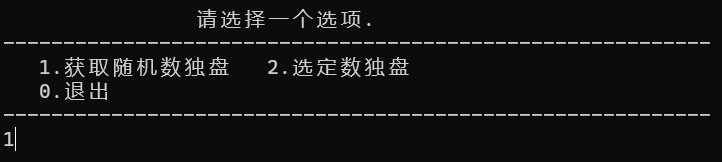


图4.2.4-11从指定数独文件中获取随机数独盘

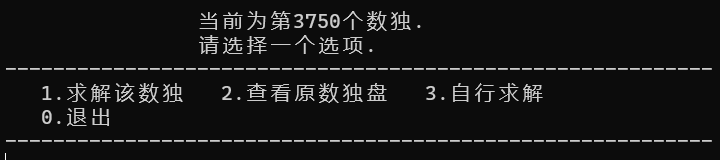


图4.2.4-12从指定数独文件中获取随机数独盘

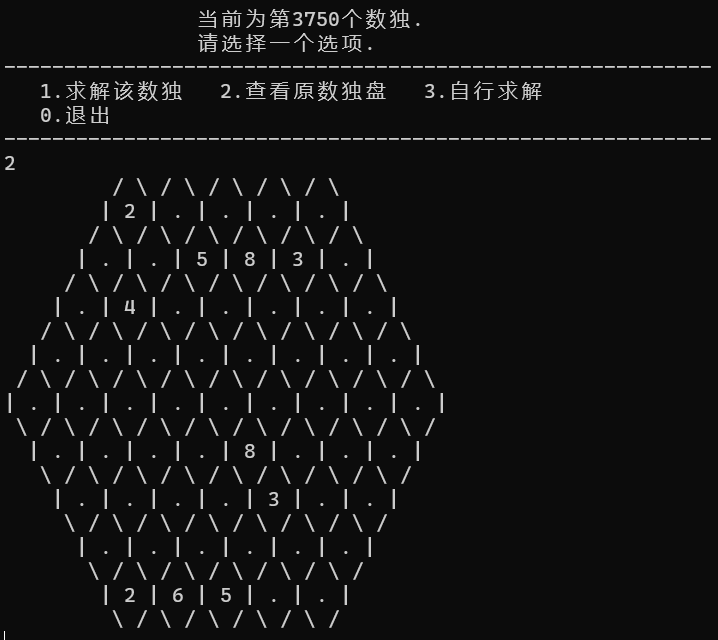


图4.2.4-13 查看数独原盘

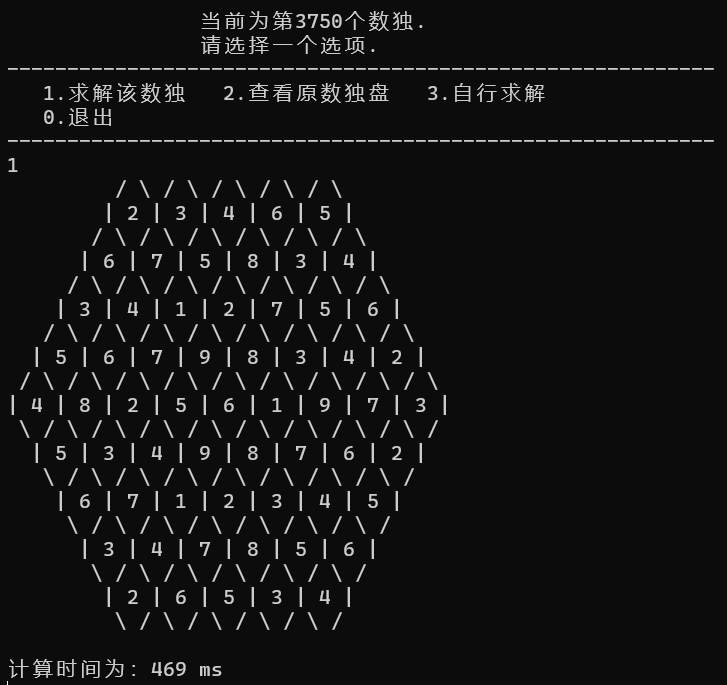


图4.2.4-14 对选定的数独盘进行求解

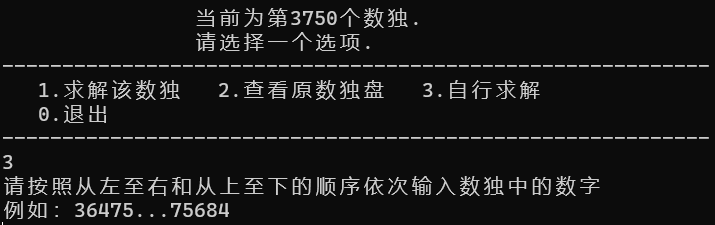


图4.2.4-15 尝试自行解答数独

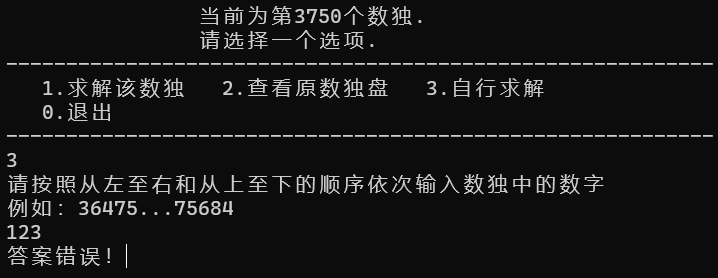


图4.2.4-16 自行解答并与标准答案进行比对

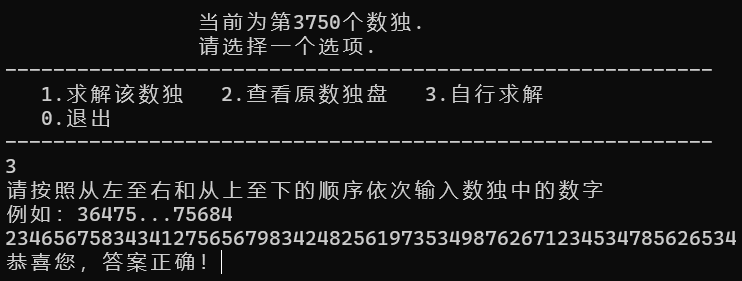


图4.2.4-17 自行解答并与标准答案进行比对

从指定文件中自行选定数独盘，如图所示：

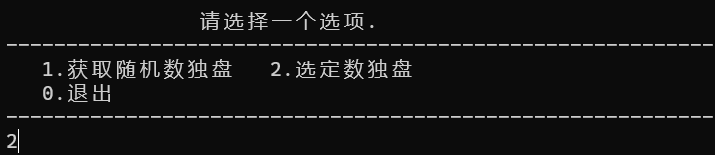


图4.2.4-18 从指定文件中自行选定数独盘

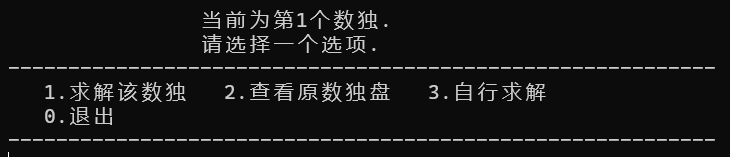


图4.2.4-19 从指定文件中自行选定数独盘

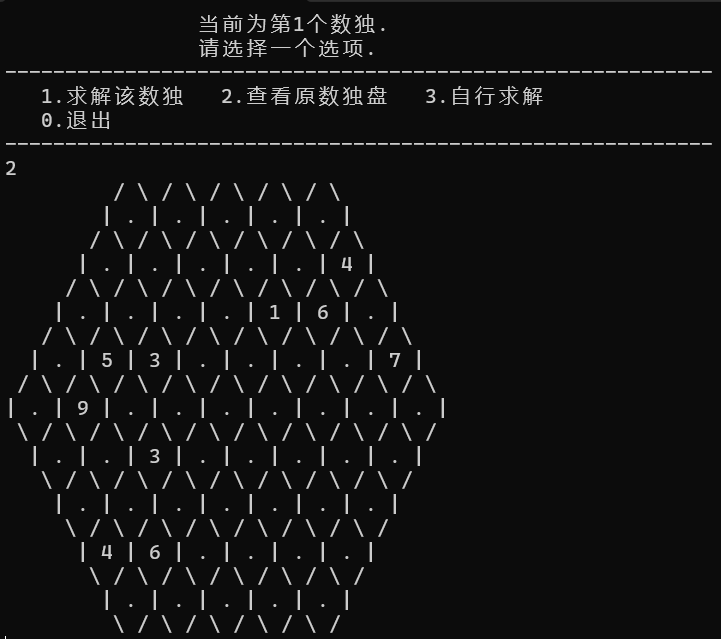


图4.2.4-20 查看数独原盘

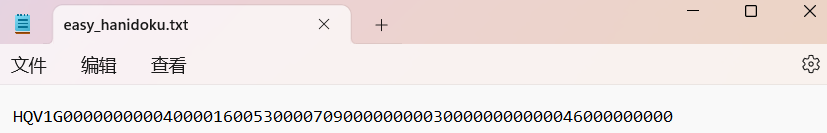


图4.2.4-21数独原盘在数独文件中的对应行

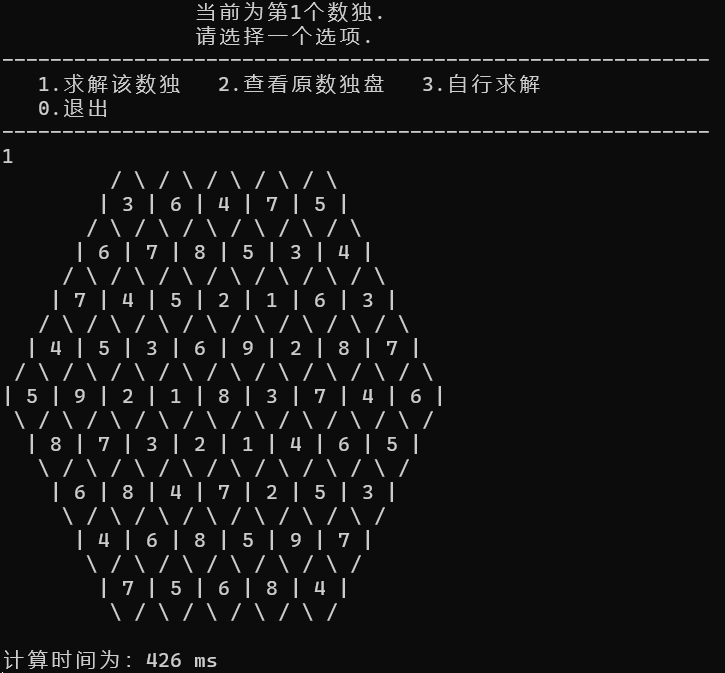


图4.2.4-22 对选定的数独盘进行求解

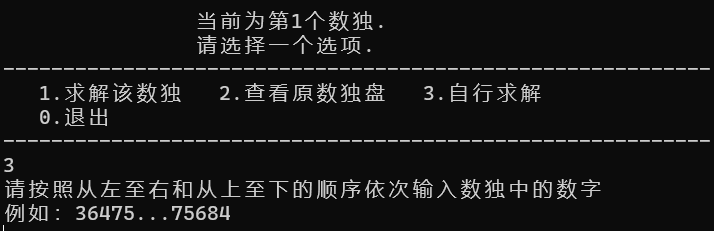


图4.2.4-23 尝试自行解答数独

**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. 在变元选择策略上对SAT求解器做了一定的优化；
3. 完成了数独游戏的初始化，CNF文件的转化，基于挖洞法的随机数独生成，数独终盘的绘制等，并实现了游戏一定的可交互性；
4. 设计了测试方案并完成了多个算例的测试；
5. 完善了各种部分的衔接，构建一个简易系统。

**5.2工作展望**

在今后的研究中，围绕着如下几个方面开展工作：

1. 由于用了较多的数据结构体去构建，设计出的SAT求解器较为低级。测试过程中，面对较多变元的算例时，偶尔可以求解但时间效率往往非常低下。起初我仅仅用链表储存文字和子句，后续优化了才加入变元真值表和正文字、负文字信息链表来便于更快的查找到要删除的文字节点，即便如此，计算效率还是有些不尽如人意。在今后的学习和工作生活中，希望可以收获更多的编程和项目经验；
2. 在DPLL算法实现中，最主要的步骤是变元选择的策略。我选择了递归回溯方案，因为设置了较复杂的数据结构体，所以回溯时都要考虑到。而变元选择策略也需要根据变元数和子句数之间的关系进行灵活调整。我暂未发现如何能用一个函数归纳总结如何确定最优变元选择策略。在今后的学习工作中，希望可以了解学习更多关于DPLL算法的编程知识；
3. 在数独游戏模块，由于蜂窝数独与普通数独相比具有更多变的结构特点和更丰富的约束条件，因此在归约时我需要仔细考虑蜂窝数独的独特造型，依据横行和斜行所具有的不同特点分别编写相应的代码对数独变元进行约束。由于我对DNF公式转换为CNF公式的方法掌握不够熟练，在归约时产生了一部分冗余语句，一定程度上降低了后续DPLL求解的工作效率，希望通过日后的学习能更好地对数独归约的实现进行不断改善。基于挖洞法生成随机数独的方法更是经历了多次失败之后才慢慢摸索出来。在今后的学习生活中，希望能通过更多高级语言的学习以及更加深入的C语言理解，学习到如何更好完成数独游戏等实际问题的实现。

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，也认识到了自己的许多不足，比如实现的SAT求解器性能较低，数独的归约逻辑也相对冗繁等等。以下是我按照完成整个课设的时间顺序写的一些心得体会：

1. 万事开头难。在最开始的时候需要阅读很多次任务书才能较为清晰地感受到实现程序的大致方法，比如数独棋盘创建CNF文件的多个约束，将变元语义编码转换为自然顺序编码的方式等等。当自己慢慢上手摸索时，经历了多次失败才逐渐开始懂得如何去构建SAT求解器，实现DPLL递归函数和构建数独系统；
2. 在实现基于DPLL的SAT求解器的过程中，我上网找了很多资料来理解递归、回溯等算法思想。由于设计了较复杂的结构体，所以回溯的时候都要考虑到。在此过程中，经历了多次失败和屡次调试后我才明白问题出在哪，需要耐心细致地对程序进行Debug。最终实现了DPLL算法，但是变元的选择策略还未太掌握清楚。
3. 在数独游戏的实现过程中，查阅了很多资料才知道挖洞法的具体创建方法，即对已知数独终盘进行挖洞，第一次挖洞后数独盘的解一定是唯一的，所以不用考虑唯一性。之后每次挖洞需要DPLL且只能求解出唯一解，否则挖洞失败。读入txt文件将原数独盘数字由语义编码转换为自然顺序编码后用数组储存，再由数组创建CNF文件，对蜂窝数独每个横行、左斜行和右斜行添加多个约束后，再进行DPLL求解。整个过程一一罗列后逐步实现，需要足够的耐心和细致的观察思考；
4. 最后就是将各部分整合为项目的问题，由于从来没有做课程设计的经验，以为项目整合并非难事，然而在网上看了教程，自己实际操作一遍后才发现有很多的问题。主要问题是在刚开始写各个模块的时候，变量、数据结构等定义得比较随意，导致项目整合的时候进行了诸多调整，浪费了许多时间。还有一些类似于重复定义的问题，起初无法解决，也是通过上网查阅相关资料才慢慢理解和完善这些问题。
5. 总而言之，在实现这次课程设计的过程中我收获了非常多的编程知识和实操经验，懂得了实践出真知的真理。代码之路需要充足的细致与耐心，路漫漫而其修远兮，吾将上下而求索。

**参考文献**

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

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

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

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

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

Twodoku： https://en.grandgames.net/multisudoku/twodoku

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

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

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

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

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

[10] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191

**附录**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define INCREASEMENT 100

typedef int status;

//定义子句链表结点结构类型

typedef struct Clause {

int literal;//记录子句中的文字

int flag;//标记该文字是否被删除，未删除时值为0，否则值为使之删除的变元序号以便于恢复

struct Clause\* nextl;//指向该子句中下一文字

struct Clause\* litline;//指向CNF范式链表中下一个具有相同文字的子句结点

}Clause;

//定义子句链表头结点结构类型

typedef struct Paradigm {

int number;//子句中的文字个数，用于寻找单子句

int flag;//标记该子句是否被删除，未删除时值为0，否则值为使之删除的变元序号以便于恢复

struct Paradigm\* nextc;//指向CNF范式链表中下一子句头结点

struct Clause\* clause;//子句头指针

}Paradigm;

//定义CNF范式链表头结点结构类型

typedef struct Root {

int lsize;//CNF范式链表中的文字数量

int psize;//CNF范式链表中的子句数量

Paradigm\* first;//指向第一个子句的指针

}Root;

//定义指向子句链表头结点的指针链表结点结构类型

typedef struct Paraline {

Paradigm\* cline;//指向子句链表头结点

struct Paraline\* next;//指向下一链表结点

}Paraline;

//定义文字相关信息链表结构类型

typedef struct LitTrabverse {

Paraline\* Tra\_c;//指向含有该文字的子句头结点链表的头结点

Clause\* Tra\_l;//指向该文字的文字结点

}LitTrabverse;

//定义存储变元信息的变元链表结点结构类型

typedef struct ArgueValue {

int Value;//变元的真值

int IsInit;//变元是否已被赋值

int Occur\_times;//变元在所有子句中出现的总次数

LitTrabverse Pos;//变元所有正文字的相关信息结构

LitTrabverse Neg;//变元所有负文字的相关信息结构

}ArgueValue;

//全局变量

char filename[1000];//文件名

Root\* r;//CNF范式链表

ArgueValue\* ValueList;//变元表

int\* SortList;//变元出现次数排序数组，便于后续DPLL

int\* LiteralList;//记录DPLL中真值赋为1的文字

long Listsize;//记录LiteralList大小

int sum\_op;

int table0[62];//记录原始数独盘

int table[62];//记录数独盘

char tableself[20000];//记录用户输入数独盘

int count;//记录原数独盘中非0数字个数

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

status CreateParadigm(FILE\*\* fp);//创建CNF范式链表及变元表

status CreateClause(FILE\*\* fp, Clause\*\* clause, Paradigm\* ClausHead, int first);//创建子句链表及文字链表

status DestroyParadigm(Root\* r);//销毁CNF范式链表结构

status HasUnitClause(Root\* r);//判断CNF范式链表中是否含有单子句

status isUnitClause(Paradigm\* p);//判断子句链表是否为单子句链表

//在CNF范式链表中取一个文字，用于DPLL

status FindLiteral1(Root\* r);//取每次DPLL处理后公式中出现次数最多的文字

status FindLiteral2(Root\* r);//取变元表中第一个未赋真值的变元的正文字

status FindLiteral3(Root \* r);//取原公式中未赋真值的变元中出现次数最多的变元正文字

status DeleteClause(Root\* r, int l);//删除包含文字l（真值为1）的所有子句

status DeleteLiteral(Root\* r, int l);//删除文字-l（l为真值为1的文字）的所有子句链表结点

status AddClause(Root\* r, int l);//在CNF范式链表表头添加含文字l的单子句链表结点

status RemoveHeadClause(Root\* r, int l);//删除CNF范式链表中第一个含有文字l的单子句链表

status Recover(Root\* r, int l);//恢复假定文字l真值为1时对CNF范式链表的操作

void ParadigmTrabverse(Root \*r);//遍历CNF范式链表

status SaveAnswer(ArgueValue\* ValueList, int solut, int time);//保存CNF范式公式的解和求解时间信息

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

status DPLL(int num,int times);//优化后的DPLL函数（采用第一种变元选择策略）

status DPLL\_BEFORE(int num,int times);//优化前的DPLL函数（采用第二种变元选择策略）

void AnswerComplete(void);//求解完成后将所有未赋真值的变元（在求解过程中真值为1，被化简舍去的变元）赋真值为1

status AnswerCheck(int solut);//检查SAT求解结果是否正确

status SAT(void);//SAT流程控制

/\*--------------------------蜂窝数独--------------------------\*/

status SudokuCreateFile(void);//对数独进行归约，创建数独问题转化为SAT问题后的CNF文件

status SudokuPrepare(char\* filename, int op);//预处理数独文件，获取原始数独盘信息并创建数独CNF公式文件

status Sudoku(void);//数独流程控制

void SudokuComplete(void);//将变元文字信息转换成数独数字，得到数独终盘信息

void SudokuPrint(int \*tab);//绘制出数独终盘图像

status SudokuCreate(int num);//采用挖洞法创造一个新的随机数独题

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "SAT.h"

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

//创建CNF范式链表及变元表

status CreateParadigm(FILE\*\* fp) {

char readfile[200];//记录在文件中读到的内容

int n, i, j;

Paradigm\* p, \* q;

r = (Root\*)malloc(sizeof(Root));//创建CNF链表表头结点

r->first = NULL;

while (fscanf(\*fp, "%s", readfile) != EOF) {//循环读文件

if (strcmp(readfile, "p") == 0)//读到字符“p”即开始读取文件

break;

}

while (fscanf(\*fp, "%s", readfile) != EOF) {

if(strcmp(readfile,"cnf")==0){//读到字符串“cnf”即开始读取并记录变元数和子句总数

fscanf(\*fp, "%d", &n);

r->lsize = n;//变元数存入r->lsize

fscanf(\*fp, "%d", &n);

r->psize = n;//子句总数存入r->psize

break;

}

}

if (sum\_op == 1) printf("文字数%d ，子句数%d！\n", r->lsize, r->psize);

//创建变元表

ValueList = (ArgueValue\*)malloc((r->lsize + 1) \* sizeof(ArgueValue));

if (ValueList == NULL) {

return OVERFLOW;//变元表内存分配失败

}

for (i = 0; i <= r->lsize; i++) {//初始化所有变元

ValueList[i].IsInit = 0;

ValueList[i].Occur\_times = 0;

ValueList[i].Pos.Tra\_c = NULL;

ValueList[i].Pos.Tra\_l = NULL;

ValueList[i].Neg.Tra\_c = NULL;

ValueList[i].Neg.Tra\_l = NULL;

}

if (r->psize != 0) {

//创建第一个子句头结点链表结点

if (fscanf(\*fp, "%d", &n) != EOF && n != 0) {

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

if (p == NULL) {

return OVERFLOW;

}

r->first = p;//创建头结点

q = p;

p->number = CreateClause(fp, &p->clause, p, n);//创建其对应子句链表

p->flag = 0;

p->nextc = NULL;

//创建完整CNF链表

i = 2;//跳过第一个子句头结点链表结点

while (i <= r->psize) {

fscanf(\*fp, "%d", &n);

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

if (p == NULL) {

return OVERFLOW;

}

p->number = CreateClause(fp, &p->clause, p, n);//创建其对应子句链表,CreateClause函数中读取并存储每个文字相关信息

p->flag = 0;

p->nextc = NULL;

q->nextc = p;//将链表中两个相邻结点连接起来

q = q->nextc;

i++;

}

}

}

//创建栈stack记录DPLL过程中设为1的文字及随之产生的单子句文字(后续用DPLL)

LiteralList = (int\*)malloc(200000 \* sizeof(int));

LiteralList[0] = 0;

Listsize = 0;

SortList = (int\*)malloc((r->lsize) \* sizeof(int));

//将变元按出现次数由多至少按顺序排序

for (i = 0; i < r->lsize; i++)

{

SortList[i] = i + 1;

}

//进行冒泡排序

for (i = 0; i < r->lsize; i++) {

for (j = i + 1; j <= r->lsize; j++) {

if (ValueList[SortList[i]].Occur\_times < ValueList[SortList[j - 1]].Occur\_times) {

int temp = SortList[i];

SortList[i] = SortList[j - 1];

SortList[j - 1] = temp;

}

}

}

fclose(\*fp);

return OK;

}

//创建子句链表和文字链表

int CreateClause(FILE\*\* fp, Clause\*\* clause, Paradigm\* ClausHead, int first) {

Clause\* p, \* q;

Paraline\* pline;

int n, l;

int i = 0;//记录每个子句中文字的数量

if (first == 0) {

return ERROR;//若没有文字的子句则报错

}

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

if (p == NULL) {

return OVERFLOW;

}

p->literal = first;

p->flag = 0;

p->nextl = NULL;

p->litline = NULL;

q = p;

\*clause = p;

i++;

l = abs(first);

ValueList[l].Occur\_times++;

if(first>0){//文字值为正

//用于DPLL时寻找包含相同文字的不同字句及删除后的恢复

p->litline = ValueList[l].Pos.Tra\_l;

ValueList[l].Pos.Tra\_l = p;

pline = (Paraline\*)malloc(sizeof(Paraline));

pline->cline = ClausHead;

pline->next = ValueList[l].Pos.Tra\_c;

ValueList[l].Pos.Tra\_c = pline;

}

else {

p->litline = ValueList[l].Neg.Tra\_l;

ValueList[l].Neg.Tra\_l = p;

pline = (Paraline\*)malloc(sizeof(Paraline));

pline->cline = ClausHead;

pline->next = ValueList[l].Neg.Tra\_c;

ValueList[l].Neg.Tra\_c = pline;

}

fscanf(\*fp, "%d", &n);

while (n != 0) {

//创建文字链表结点

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

if (p == NULL) {

return OVERFLOW;

}

p->literal = n;

p->flag = 0;

p->nextl = NULL;

q->nextl = p;//连接两个相邻文字结点

q = q->nextl;

i++;

l = abs(n);

ValueList[l].Occur\_times++;

if (n > 0) {

p->litline = ValueList[l].Pos.Tra\_l;

ValueList[l].Pos.Tra\_l = p;

pline = (Paraline\*)malloc(sizeof(Paraline));

pline->cline = ClausHead;

pline->next = ValueList[l].Pos.Tra\_c;

ValueList[l].Pos.Tra\_c = pline;

}

else {

p->litline = ValueList[l].Neg.Tra\_l;

ValueList[l].Neg.Tra\_l = p;

pline = (Paraline\*)malloc(sizeof(Paraline));

pline->cline = ClausHead;

pline->next = ValueList[l].Neg.Tra\_c;

ValueList[l].Neg.Tra\_c = pline;

}

fscanf(\*fp, "%d", &n);

}

return i;

}

//销毁CNF范式链表

status DestroyParadigm(Root\* r) {

Paradigm\* p;

Clause\* c;

Paraline\* pline;

int i;

if (r->first != NULL) {

p = r->first;

//销毁CNF范式链表

while (p != NULL) {

if (p->clause != NULL) {

c = p->clause;

//销毁单个子句链表

while (c != NULL) {

p->clause = c->nextl;

free(c);//释放存储文字的单个子句链表结点空间

c = p->clause;

}

}

r->first = p->nextc;

free(p);//释放子句链表头结点存储空间

p = r->first;

}

}

//释放变元正负文字信息链表存储空间

for (i = 0; i <= r->lsize; i++) {

for (pline = ValueList[i].Pos.Tra\_c; pline != NULL; pline = ValueList[i].Pos.Tra\_c) {

ValueList[i].Pos.Tra\_c = pline->next;

free(pline);

}

ValueList[i].Pos.Tra\_l = NULL;

for (pline = ValueList[i].Neg.Tra\_c; pline != NULL; pline = ValueList[i].Neg.Tra\_c) {

ValueList[i].Neg.Tra\_c = pline->next;

free(pline);

}

ValueList[i].Neg.Tra\_l = NULL;

}

free(SortList);

free(r);

return OK;

}

//判断CNF范式链表中是否含有单子句

//若存在则返回单子句的文字值，不存在则返回0

status HasUnitClause(Root\* r) {

Paradigm\* p;

Clause\* c;

for (p = r->first;p != NULL;p = p->nextc) {

if (p->flag == 0 && p->number == 1) {//子句没有被删除，且文字数为1

for ( c = p->clause; c!=NULL ; c=c->nextl){

if (c->flag == 0) {//文字也没有被删除

return c->literal;//返回文字的值

}

}

}

}

return 0;//不存在单子句，返回0

}

//判断子句链表是否为单子句链表

status isUnitClause(Paradigm\* p) {

if (p->number == 1) {//子句链表含文字数为1

return TRUE;

}

else {

return FALSE;

}

}

//取每次DPLL处理后公式中出现次数最多的文字

//若公式非空，则返回该文字的值，反之则返回0

status FindLiteral1(Root\* r) {

int i, num = 0, flag = 0;

for (i = 1; i <= r->lsize; i++) {

if (ValueList[i].IsInit == 0) {

num = i;//找到变元表正序第一个未被赋真值的变元，num记录其序号

flag = 1;//表示变元表内存在未被赋真值的变元

break;

}

}

if (flag == 1) {

for (i++; i <= r->lsize; i++) {

if (ValueList[i].IsInit == 0) {

if (ValueList[i].Occur\_times > ValueList[num].Occur\_times)

num = i;

}

}

}

return num;

}

//取变元表中第一个未赋真值的变元的正文字

//若变元表中无未赋真值的变元，则返回0

status FindLiteral2(Root\* r) {

int i = 0;

for (i = 1;i <= r->lsize;i++) {

if (ValueList[i].IsInit == 0) {

return i;

}

}

return 0;

}

//取原公式中未赋真值的变元中出现次数最多的变元正文字

int FindLiteral3(Root\* r) {

int i = 0;

for (i = 0; i < r->lsize; i++) {

if (ValueList[SortList[i]].IsInit == 0) {

return SortList[i];

}

}

return 0;

}

//删除含文字l（真值为1）的所有子句

status DeleteClause(Root\* r, int l) {

Paradigm\* p;

Clause\* c;

Paraline\* pline;

//搜索正（负）文字l对应变元的正（负）文字信息链表

if (l > 0) {//l为正文字

for (pline = ValueList[l].Pos.Tra\_c;pline;pline = pline->next) {

p = pline->cline;

if(p->flag==0){//子句头结点所在子句未被删除

p->flag = l;//将子句头结点的flag值标记为l

r->psize--;

for(c=p->clause;c!=NULL;c=c->nextl){//依次将子句中所有未被删除的文字的flag值标记为l

if (c->flag == 0) {

c ->flag = l;

ValueList[abs(c->literal)].Occur\_times--;//被删除文字的出现次数减1

p->number--;

}

}

}

}

}

else {//l为负文字，同上

l = abs(l);

for (pline = ValueList[l].Neg.Tra\_c;pline;pline = pline->next) {

p = pline->cline;

if (p->flag == 0) {//子句头结点所在子句未被删除

p->flag = l;//将子句头结点的flag值标记为l

r->psize--;

for (c = p->clause;c != NULL;c = c->nextl) {//依次将子句中所有未被删除的文字的flag值标记为l

if (c->flag == 0) {

c->flag = l;

ValueList[abs(c->literal)].Occur\_times--;//被删除文字的出现次数减1

p->number--;

}

}

}

}

}

return OK;

}

//删除文字-l（l为真值为1的文字）的所有子句链表结点

//若公式无解则返回FALSE

status DeleteLiteral(Root\* r, int l) {

Paradigm\* p;

Clause\* c;

Paraline\* pline;

if(l>0){//l为正文字，则-l为负文字，需要在负文字信息链表中搜索

for (pline = ValueList[l].Neg.Tra\_c;pline != NULL;pline = pline->next) {

p = pline->cline;

if (p->flag==0){//子句头结点所在未被删除

for (c = p->clause;c != NULL;c = c->nextl) {

if (c->literal == -l && c->flag==0) {//结点文字为-l且文字未被删除

if (p->number==1){//该子句为单子句，真值为0，公式无解

return FALSE;

}

c->flag = l;//将文字结点的flag值标记为l

ValueList[abs(c->literal)].Occur\_times--;

p->number--;

}

}

}

}

}

else{//l为负文字，则-l为正文字，需要在正文字信息链表中搜索

for (pline = ValueList[abs(l)].Pos.Tra\_c;pline != NULL;pline = pline->next) {

p = pline->cline;

if (p->flag == 0) {//子句头结点所在未被删除

for (c = p->clause;c != NULL;c = c->nextl) {

if (c->literal == -l && c->flag == 0) {//结点文字为-l且文字未被删除

if (p->number == 1) {//该子句为单子句，真值为0，公式无解

return FALSE;

}

c->flag = abs(l);//将文字结点的flag值标记为l

ValueList[abs(c->literal)].Occur\_times--;

p->number--;

}

}

}

}

}

return OK;

}

//在CNF范式链表表头添加含文字l的单子句链表结点

status AddClause(Root\* r, int l) {

Paradigm\* p;

Clause\* c;

Paraline\* pline;

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

if (p == NULL) {

return OVERFLOW;

}

p->nextc = r->first;

c = (Clause\*)malloc(sizeof(Clause));

if (c == NULL) {

return OVERFLOW;

}

c->literal = l;

c->flag = 0;

c->nextl = NULL;

if(l>0){//l为正文字，在变元表内l对应变元的正文字信息链表表头增加新结点信息

c->litline = ValueList[l].Pos.Tra\_l;

ValueList[l].Pos.Tra\_l = c;

pline = (Paraline\*)malloc(sizeof(Paraline));

pline->cline = p;

pline->next = ValueList[l].Pos.Tra\_c;

ValueList[l].Pos.Tra\_c = pline;

}

else{//l为负文字，在变元表内l对应变元的负文字信息链表表头增加新结点信息

l = abs(l);

c->litline = ValueList[l].Neg.Tra\_l;

ValueList[l].Neg.Tra\_l = c;

pline = (Paraline\*)malloc(sizeof(Paraline));

pline->cline = p;

pline->next = ValueList[l].Neg.Tra\_c;

ValueList[l].Neg.Tra\_c = pline;

}

p->clause = c;

p->number = 1;

p->flag = 0;

r->first = p;

r->psize++;

return OK;

}

//删除CNF范式链表中第一个含有文字l的单子句链表

status RemoveHeadClause(Root\* r, int l) {

Paradigm\* p, \* q;

Clause\* c;

Paraline\* pline;

if(l>0){//l为正文字，要删除的结点信息在变元表内l对应变元的正文字信息链表表头

pline = ValueList[l].Pos.Tra\_c;

p = pline->cline;

ValueList[l].Pos.Tra\_c = pline->next;

ValueList[l].Pos.Tra\_l = ValueList[l].Pos.Tra\_l->litline;

}

else {//l为负文字，要删除的结点信息在变元表内l对应变元的负文字信息链表表头

l = abs(l);

pline = ValueList[l].Neg.Tra\_c;

p = pline->cline;

ValueList[l].Neg.Tra\_c = pline->next;

ValueList[l].Neg.Tra\_l = ValueList[l].Neg.Tra\_l->litline;

}

if (r->first == p) {

r->first = p->nextc;

}

else {

for (q = r->first;q != NULL;q = q->nextc) {//寻找表头开始第一个含文字l的单子句

if (q->nextc == p) {

q->nextc = p->nextc;

break;

}

}

}

c = p->clause;

//释放该单子句结点所有信息

free(c);

free(p);

free(pline);

r->psize--;

return OK;

}

//恢复假定文字l真值为1时对CNF范式链表的操作

status Recover(Root\* r, int l) {

Paradigm\* p;

Clause\* c;

Paraline\* pline;

if (l > 0) {

//l为正文字，对变元表内l对应变元的正文字信息链表进行搜索，找到被删除的子句及文字

for (pline = ValueList[l].Pos.Tra\_c;pline != NULL;pline = pline->next) {

p = pline->cline;

if (p->flag==l){//找到因l被删除的子句

p->flag = 0;

r->psize++;

for(c=p->clause;c;c=c->nextl){//找到因l被删除的文字

if (c->flag == l) {

c->flag = 0;

p->number++;

}

}

}

}

//l为正文字，对变元表内l对应变元的负文字信息链表进行搜索，找到被删除文字

for (pline = ValueList[l].Neg.Tra\_c;pline != NULL;pline = pline->next) {

p = pline->cline;

for (c = p->clause;c != NULL;c = c->nextl) {

if (c->flag == l) {//找到因l被删除的文字

c->flag = 0;

p->number++;

}

}

}

}

else {

l = abs(l);

//l为负文字，对变元表内l对应变元的负文字信息链表进行搜索，找到被删除的子句及文字

for (pline = ValueList[l].Neg.Tra\_c;pline != NULL;pline = pline->next) {

p = pline->cline;

if (p->flag == l) {//找到因l被删除的子句

p->flag = 0;

r->psize++;

for (c = p->clause;c;c = c->nextl) {//找到因l被删除的文字

if (c->flag == l) {

c->flag = 0;

p->number++;

}

}

}

}

//l为负文字，对变元表内l对应变元的正文字信息链表进行搜索，找到被删除文字

for (pline = ValueList[l].Pos.Tra\_c;pline != NULL;pline = pline->next) {

p = pline->cline;

for (c = p->clause;c != NULL;c = c->nextl) {

if (c->flag == l) {//找到因l被删除的文字

c->flag = 0;

p->number++;

}

}

}

}

return OK;

}

//遍历CNF范式链表

void ParadigmTrabverse(Root\* r) {

Paradigm\* p;

Clause\* c;

int i=1;

for (p = r->first;p != NULL;p = p->nextc) {

printf("第%d行 ", i++);

for (c = p->clause;c != NULL;c = c->nextl) {

printf("%4d ", c->literal);

}

printf("\n");

}

printf("遍历完成！\n");

}

//保存CNF范式公式的解和求解时间信息

status SaveAnswer(ArgueValue\* ValueList, int solut, int time) {

int i;

FILE\* saveinfo;

char filenameinfo[1000];

for (i = 0;filename[i] != '.';i++) {

filenameinfo[i] = filename[i];

}

//为新文件添加文件后缀名

filenameinfo[i++] = '.';

filenameinfo[i++] = 'r';

filenameinfo[i++] = 'e';

filenameinfo[i++] = 's';

filenameinfo[i++] = '\0';

saveinfo = fopen(filenameinfo, "wb");

if (saveinfo == NULL) {

printf("文件保存失败！\n");

return ERROR;

}

fprintf(saveinfo, "s %d\n", solut);

fprintf(saveinfo, "v ");

if (solut==1){//CNF公式有解

for (i = 1;i <= r->lsize;i++) {

if (ValueList[i].IsInit == 1){//变元已被赋值

if (ValueList[i].Value == 1) {//变元真值为1

fprintf(saveinfo, "%-7d", i);

}

else{//变元真值为0

fprintf(saveinfo, "%-7d", -i);//打印负文字

}

if (i % 10 == 0) {

fprintf(saveinfo, "\n");//每十个数换一次行

}

}

}

}

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

fclose(saveinfo);

return OK;

}

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

//优化后的DPLL函数（采用第一种变元选择策略）

//num：进入DPLL时真值设为1的变元

//op：普通求解cnf文件为1，生成数独终盘为2

//times：第一次进入DPLL为1，反之为2

//公式有解则返回TRUE，反之返回FALSE

cnt = 0;

status DPLL(int num,int times) {

//printf("进入DPLL %d", ++cnt);

int numabs = abs(num);

int l = num;

if (times == 1) {

l = 0;

}

Paradigm\* p = r->first;

Clause\* c;

Paraline\* pline;

//单子句策略

while (l != 0 ){

LiteralList[++Listsize] = l;//记录真值赋为1的文字

DeleteClause(r, l);//删除包含num的所有子句

if (DeleteLiteral(r,l)==FALSE){//删除子句中所有num的反文字

//探测后公式无解，恢复到递归进入本次函数前的状态

for (; LiteralList[Listsize] != num; Listsize--) {

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

}

//已恢复num之后的文字，再来恢复num

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

Listsize--;

return FALSE;

}

//探测后公式仍成立，对num进行赋值

if (l > 0) {

ValueList[l].Value = 1;

ValueList[l].IsInit = 1;

}

else {

ValueList[abs(l)].Value = 0;

ValueList[abs(l)].IsInit = 1;

}

if(FindLiteral1(r)==0){//公式内无剩余未赋值文字，求解成功

return TRUE;

}

l = HasUnitClause(r);//寻找公式中的单子句文字，若无则跳出循环，并重新对num赋值

}

l = FindLiteral1(r);

if (l == 0) {

return TRUE;

}

if (DPLL(l,2)){//进入下一次循环

return TRUE;

}

//探测后公式无解，说明num的真值不为1，则num真值为0，其反文字真值为1

l = -l;

if (DPLL(l,2)) {//进入下一次循环

return TRUE;

}

else{//探测后公式仍无解，同上，恢复到递归进入本次函数前的状态

if (Listsize == 0) {

return FALSE;

}

for (; LiteralList[Listsize] != num; Listsize--) {

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

}

//已恢复num之后的文字，再来恢复num

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

Listsize--;

return FALSE;

}

}

//优化前的DPLL函数（采用第二种变元选择策略）

status DPLL\_BEFORE(int num,int times) {

int numabs = abs(num);

int l = num;

if (times == 1) {

l = 0;

}

Paradigm\* p = r->first;

Clause\* c;

Paraline\* pline;

//单子句策略

while (l != 0) {

LiteralList[++Listsize] = l;//记录真值赋为1的文字

DeleteClause(r, l);//删除包含num的所有子句

if (DeleteLiteral(r, l) == FALSE) {//删除子句中所有num的反文字

//探测后公式无解，恢复到递归进入本次函数前的状态

for (; LiteralList[Listsize] != num; Listsize--) {

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

}

//已恢复num之后的文字，再来恢复num

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

Listsize--;

return FALSE;

}

//探测后公式仍成立，对num进行赋值

if (l > 0) {

ValueList[l].Value = 1;

ValueList[l].IsInit = 1;

}

else {

ValueList[abs(l)].Value = 0;

ValueList[abs(l)].IsInit = 1;

}

if (FindLiteral2(r) == 0) {//公式内无剩余未赋值文字，求解成功

return TRUE;

}

l = HasUnitClause(r);//寻找公式中的单子句文字，若无则跳出循环，并重新对num赋值

}

l = FindLiteral2(r);

if (l == 0) {

return TRUE;

}

if (DPLL\_BEFORE(l,2)) {//进入下一次循环

return TRUE;

}

//探测后公式无解，说明num的真值不为1，则num真值为0，其反文字真值为1

l = -l;

if (DPLL\_BEFORE(l,2)) {//进入下一次循环

return TRUE;

}

else {//探测后公式仍无解，同上，恢复到递归进入本次函数前的状态

if (Listsize == 0) {

return FALSE;

}

for (; LiteralList[Listsize] != num; Listsize--) {

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

}

//已恢复num之后的文字，再来恢复num

Recover(r, LiteralList[Listsize]);

ValueList[abs(LiteralList[Listsize])].IsInit = 0;

Listsize--;

return FALSE;

}

}

//求解完成后将所有未赋真值的变元（在求解过程中真值为1，被化简舍去的变元）赋真值为1

void AnswerComplete(void) {

int i;

for (i = 1;i < r->lsize;i++) {

if (ValueList[i].IsInit == 0) {

ValueList[i].IsInit = 1;

ValueList[i].Value = 1;

}

}

}

//检查SAT求解结果是否正确

//正确则返回TRUE，反之返回FALSE

status AnswerCheck(int solut){

Paradigm\* p;

Clause\* c;

int flag,value;

if (solut){//公式有解

for (p = r->first;p != NULL;p = p->nextc) {

flag = 0;

for (c = p->clause;c != NULL;c = c->nextl) {

if (c->literal > 0) {

value = ValueList[abs(c->literal)].Value;

}

else {

value = 1 - ValueList[abs(c->literal)].Value;

}

if (value) {//子句中有真值为1的文字，子句成立

flag = 1;

break;

}

}

if (flag == 0) {//子句中无真值为1的文字，子句不成立，求解错误

return FALSE;

}

}

return TRUE;

}

else{//公式无解

for (p = r->first;p != NULL;p = p->nextc) {

flag = 0;

for (c = p->clause;c != NULL;c = c->nextl) {

if (c->literal > 0) {

value = ValueList[abs(c->literal)].Value;

}

else {

value = 1 - ValueList[abs(c->literal)].Value;

}

if (value) {//子句中有真值为1的文字，子句成立

flag = 1;

break;

}

}

if (flag == 0) {//子句中无真值为1的文字，子句不成立，求解正确

return TRUE;

}

}

return FALSE;

}

}

//SAT流程控制

status SAT(void) {

int i;

int op=1;

int op2;//选择优化前后算法

int solut = 2;//记录CNF公式是否有解，有解为1，反之为0

clock\_t start = 0, finish = 0;//记录DPLL函数调用时的起始时间和终止时间

int soltime = 0;//记录求解总时间

int check;//记录求解正确性

FILE\* open;

printf("请输入需要求解的cnf文件名：");

scanf("%s",filename);

open = fopen(filename, "r");

if (open == NULL) {

printf("文件打开错误！\n");

getchar();

getchar();

return ERROR;

}

if (CreateParadigm(&open)==ERROR){//创建CNF范式链表及变元表结构

getchar();

getchar();

return ERROR;

}

getchar();

getchar();

while (op) {

system("cls");

printf("\t\t请选择一个选项.\n");

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

printf(" 1.求解该算例 2.遍历CNF文件 3.保存求解结果\n");

printf(" 0.退出\n");

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

scanf("%d", &op);

switch (op) {

case 1:

if (solut == 2) {//变元表中元素已被赋值，说明公式已被求解

system("cls");

printf("\t\t请选择一个选项.\n");

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

printf(" 1.使用优化算法 2.使用优化前算法\n");

printf(" 0.退出\n");

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

scanf("%d", &op2);

switch (op2) {

case 1:

start = clock();

solut = DPLL(FindLiteral1(r),1);

finish = clock();

soltime = finish - start;

break;

case 2:

start = clock();

solut = DPLL\_BEFORE(FindLiteral2(r),1);

finish = clock();

soltime = finish - start;

break;

case 0:

break;

default:

printf("输入错误！\n");

getchar();

getchar();

break;

}

}

if (op2 != 1 && op2 != 2) {

break;

}

if (solut) {

AnswerComplete();

check = AnswerCheck(solut);

if (check) {

printf("求解正确！\n");

}

else {

printf("求解错误！\n");

}

for (i = 1;i <= r->lsize; i++) {

if (ValueList[i].IsInit == 1) {

if (ValueList[i].Value == 1) {

printf("%-7d", i);//真值为1，打印正文字

}

else {

printf("%-7d", -i);//真值为0，打印负文字

}

}

if (i % 10 == 0) {

printf("\n");//每打印十个文字换一次行

}

}

}

else {

check = AnswerCheck(solut);

if (check) {

printf("求解正确！\n");

}

else {

printf("求解错误！\n");

}

printf("公式无解！\n");

}

printf("\n计算时间为：%d ms\n", soltime);

getchar();

getchar();

break;

case 2:

if (solut == 2) {//还未对CNF公式求解

printf("请先求解CNF公式！\n");

getchar();

getchar();

break;

}

else {

ParadigmTrabverse(r);

getchar();

getchar();

break;

}

case 3:

if (solut == 2) {//还未对CNF公式求解

printf("请先求解CNF公式！\n");

getchar();

getchar();

break;

}

if (SaveAnswer(ValueList, solut, soltime)) {

printf("成功存入文件！");

}

getchar();

getchar();

break;

case 0:

break;

default:

printf("输入错误！");

getchar();

getchar();

break;

}

}

DestroyParadigm(r);//销毁CNF范式链表结构

return OK;

}

/\*--------------------------蜂窝数独--------------------------\*/

//数独流程控制

status Sudoku(void) {

clock\_t start, finish;

int soltime;

int flag=1;

int flag2 = 1;

int op=0;

int op1 = 1;

int op2 = 1;

int op3 = 0;

int solut = 0;

int lines = 0;

int i;

char filename[100];

char read;

while (flag2) {

system("cls");

printf("\t\t请选择一个选项.\n");

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

printf(" 1.挖洞法随机生成新数独 2.打开指定数独文件\n");

printf(" 0.退出\n");

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

scanf("%d", &op2);

switch (op2) {

case 1:

printf("请选择生成数独难度1.困难 2.一般 3.简单：");

scanf("%d", &op3);

switch (op3) {

case 1:

SudokuCreate(51);//使用挖洞法生成数独盘table

flag2 = 0;

break;

case 2:

SudokuCreate(46);//使用挖洞法生成数独盘table

flag2 = 0;

break;

case 3:

SudokuCreate(41);//使用挖洞法生成数独盘table

flag2 = 0;

break;

case 0:

break;

default:

printf("输入错误！");

getchar();

getchar();

break;

}

break;

case 2:

printf("请输入数独文件的文件名：");

scanf("%s", filename);

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

if (fp == NULL) {

printf("文件打开错误！\n");

getchar();

getchar();

return ERROR;

}

//统计文件行数，即数独题目数量

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

if (read == '\n') {

lines++;

}

}

fclose(fp);

while (flag) {

system("cls");

printf("\t\t请选择一个选项.\n");

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

printf(" 1.获取随机数独盘 2.选定数独盘\n");

printf(" 0.退出\n");

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

scanf("%d", &op);

switch (op) {

case 1:

srand((unsigned int)time(NULL));

op = rand() % lines + 1;

flag = 0;

break;

case 2:

printf("请输入数独编号（在文件中的行数）：");

scanf("%d", &op);

if (op <= 0 || op > lines) {

printf("行数不合法！");

getchar();

getchar();

break;

}

flag = 0;

break;

case 0:

flag = 0;

op1 = 0;

break;

default:

printf("输入错误！");

getchar();

getchar();

break;

}

}

SudokuPrepare(filename, op);//预处理数独文件

flag2 = 0;

break;

case 0:

flag2 = 0;

op1 = 0;

break;

default:

printf("输入错误！");

getchar();

getchar();

break;

}

}

while (op1) {

system("cls");

if (op != 0) {

printf("\t\t当前为第%d个数独.\n", op);

}

printf("\t\t请选择一个选项.\n");

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

printf(" 1.求解该数独 2.查看原数独盘 3.自行求解\n");

printf(" 0.退出\n");

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

scanf("%d", &op1);

switch (op1) {

case 1:

if (solut == 0) {

start = clock();

DPLL\_BEFORE(FindLiteral1(r), 1);

AnswerComplete();

SudokuComplete();

solut = 1;

finish = clock();

soltime = finish - start;

}

SudokuPrint(table);

printf("\n计算时间为：%d ms\n", soltime);

getchar();

getchar();

break;

case 2:

SudokuPrint(table0);

getchar();

getchar();

break;

case 3:

if (solut == 0) {

/\*start = clock();

DPLL\_BEFORE(FindLiteral1(r), 1, 1);

AnswerComplete();

SudokuComplete();

solut = 1;

finish = clock();

soltime = finish - start;\*/

printf("请先求解数独！");

getchar();

getchar();

break;

}

printf("请按照从左至右和从上至下的顺序依次输入数独中的数字\n");

printf("例如：36475...75684\n");

scanf("%s", tableself);

for (i = 0;i < 61;i++) {

if (table[i] == tableself[i]-'0') {

continue;

}

else {

break;

}

}

if (i == 61) {

printf("恭喜您，答案正确！");

}

else {

printf("答案错误！");

}

getchar();

getchar();

break;

case 0:

break;

default:

printf("输入错误！");

getchar();

getchar();

break;

}

}

DestroyParadigm(r);//销毁CNF范式链表结构

return OK;

}

//对数独进行归约，创建数独问题转化为SAT问题后的CNF文件

status SudokuCreateFile(void) {

int x, y, z;//x,y,z分别表示数独的行，列，数独该格中填入的数字

int i, j;

int k;//表示斜行

int p, q, s;

int columns[9] = { 5,6,7,8,9,8,7,6,5 };//记录蜂窝数独每行的列数

int rows[9] = { 0,45,99,162,234,315,387,450,504 };

FILE\* fp;

fp = fopen("SudokuTableBase.cnf", "wb");

if (fp == NULL) {

printf("文件打开失败！\n");

return ERROR;

}

fprintf(fp, "p cnf 549 %d\n",13965+count);//创建CNF文件，声明共有61\*9=549个变元，7287+个子句

//将原数独盘数字转化为CNF公式

for (p = 0;p < 61;p++) {

s = 0;

if (table[p] != 0) {

z = table[p];

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

s += columns[q];

if (s >= p+1) {

x = q;

y = p - (s - columns[q]);

break;

}

}

fprintf(fp,"%d 0\n", rows[x] + 9 \* y + z);

}

}

//每个cell只能取1~9的一个值

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

for (y = 0;y < columns[x];y++) {

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

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

fprintf(fp, "%d %d 0\n", -(rows[x] + 9 \* y + z), -(rows[x] + 9 \* y + i));

}

}

}

}

//每个行1~9只能各出现一次

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

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

for (y = 0;y < columns[x] - 1;y++) {

for (i = y + 1;i <= columns[x] - 1;i++) {

fprintf(fp, "%d %d 0\n", -(rows[x] + 9 \* y + z), -(rows[x] + 9 \* i + z));

}

}

}

}

//每个左斜行1~9只能各出现一次

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

//printf("第%d个斜行---------------------------------------\n", k);

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

if (k < 5) {

for (x = 0;x < columns[k]-1;x++) {

if (x < 4) y = k;

else y = k + 4 - x;

for (i = x + 1;i < columns[k];i++) {

if (i < 4) j = k;

else j = k + 4 - i;

fprintf(fp, "%d %d 0\n", -(rows[x] + 9 \* y + z), -(rows[i] + 9 \* j + z));

}

}

}

else {

for (x = k - 4;x < k - 4 +columns[k] - 1;x++) {

if (x < 4) y = k;

else y = k + 4 - x;

for (i = x + 1;i < k - 4 +columns[k];i++) {

if (i < 4) j = k;

else j = k + 4 - i;

fprintf(fp, "%d %d 0\n", -(rows[x] + 9 \* y + z), -(rows[i] + 9 \* j + z));

}

}

}

}

}

//每个右斜行1~9只能各出现一次

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

//printf("第%d个斜行@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n", k);

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

if (k < 5) {

for (x = 0;x < columns[k] - 1;x++) {

if (x < 4) y = columns[x]-k-1;

else y = 8-k;

for (i = x + 1;i < columns[k];i++) {

if (i < 4) j = columns[i]-k-1;

else j = 8-k;

fprintf(fp, "%d %d 0\n", -(rows[x] + 9 \* y + z), -(rows[i] + 9 \* j + z));

}

}

}

else {

for (x = k - 4;x < k - 4 + columns[k] - 1;x++) {

if (x < 4) y = x - (k - 4);

else y = 4 - (k - 4);

for (i = x + 1;i < k - 4 + columns[k];i++) {

if (i < 4) j = i - (k - 4);

else j = 4 - (k - 4);

fprintf(fp, "%d %d 0\n", -(rows[x] + 9 \* y + z), -(rows[i] + 9 \* j + z));

}

}

}

}

}

int cnt = 0;

int a, b, c, d, e;

int A1[2] = { 1,2 };

int B1[2] = { 2,8 };

int C1[2] = { 8,9 };

int A2[3] = { 1,2,3 };

int B2[3] = { 2,3,7 };

int C2[3] = { 3,7,8 };

int D2[3] = { 7,8,9 };

int A3[4] = { 1,2,3,4 };

int B3[4] = { 2,3,4,6 };

int C3[4] = { 3,4,6,7 };

int D3[4] = { 4,6,7,8 };

int E3[4] = { 6,7,8,9 };

//每个行数字必须是连续的

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

if (x == 0 || x == 8) {//第一行和倒数第一行

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

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

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

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

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

for (y = 0;y < columns[x];y++) {

fprintf(fp,"%d %d %d %d %d ", rows[x] + 9 \* y + A3[a], rows[x] + 9 \* y + B3[b], rows[x] + 9 \* y + C3[c], rows[x] + 9 \* y + D3[d], rows[x] + 9 \* y + E3[e]);

}

fprintf(fp, "0\n");

}

}

}

}

}

}

if (x == 1 || x == 7) {//第二行和倒数第二行

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

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

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

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

for (y = 0;y < columns[x];y++) {

fprintf(fp,"%d %d %d %d ", rows[x] + 9 \* y + A2[a], rows[x] + 9 \* y + B2[b], rows[x] + 9 \* y + C2[c], rows[x] + 9 \* y + D2[d]);

}

fprintf(fp, "0\n");

}

}

}

}

}

if (x == 2 || x == 6) {//第三行和倒数第三行

for (a = 0;a < 2;a++) {

for (b = 0;b < 2;b++) {

for (c = 0;c < 2;c++) {

for (y = 0;y < columns[x];y++) {

fprintf(fp,"%d %d %d ", rows[x] + 9 \* y + A1[a], rows[x] + 9 \* y + B1[b], rows[x] + 9 \* y + C1[c]);

}

fprintf(fp,"0\n");

}

}

}

}

}

//每个左斜行数字必须是连续的

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

//printf("第%d个斜行---------------------------------------\n", k);

if (k == 2) {//第三个左斜行

for (a = 0;a < 2;a++) {

for (b = 0;b < 2;b++) {

for (c = 0;c < 2;c++) {

for (x = 0;x < columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d %d %d ", rows[x] + 9 \* y + A1[a], rows[x] + 9 \* y + B1[b], rows[x] + 9 \* y + C1[c]);

}

fprintf(fp,"0\n");

}

}

}

}

if (k == 6) {//倒数第三个左斜行

for (a = 0;a < 2;a++) {

for (b = 0;b < 2;b++) {

for (c = 0;c < 2;c++) {

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d %d %d ", rows[x] + 9 \* y + A1[a], rows[x] + 9 \* y + B1[b], rows[x] + 9 \* y + C1[c]);

}

fprintf(fp,"0\n");

}

}

}

}

if (k == 1) {//第二个左斜行

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

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

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

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

for (x = 0;x < columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d %d %d %d ", rows[x] + 9 \* y + A2[a], rows[x] + 9 \* y + B2[b], rows[x] + 9 \* y + C2[c], rows[x] + 9 \* y + D2[d]);

}

fprintf(fp, "0\n");

}

}

}

}

}

if (k == 7) {//倒数第二个左斜行

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

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

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

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

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d %d %d %d ", rows[x] + 9 \* y + A2[a], rows[x] + 9 \* y + B2[b], rows[x] + 9 \* y + C2[c], rows[x] + 9 \* y + D2[d]);

}

fprintf(fp, "0\n");

}

}

}

}

}

if (k == 0) {//第一个左斜行

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

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

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

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

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

for (x = 0;x < columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d %d %d %d %d ", rows[x] + 9 \* y + A3[a], rows[x] + 9 \* y + B3[b], rows[x] + 9 \* y + C3[c], rows[x] + 9 \* y + D3[d], rows[x] + 9 \* y + E3[e]);

}

fprintf(fp, "0\n");

}

}

}

}

}

}

if (k==8){//倒数第一个左斜行

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

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

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

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

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

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d %d %d %d %d ", rows[x] + 9 \* y + A3[a], rows[x] + 9 \* y + B3[b], rows[x] + 9 \* y + C3[c], rows[x] + 9 \* y + D3[d], rows[x] + 9 \* y + E3[e]);

}

fprintf(fp, "0\n");

}

}

}

}

}

}

}

//每个右斜行数字必须是连续的

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

//printf("第%d个斜行---------------------------------------\n", k);

if (k == 2) {//第三个右斜行

for (a = 0;a < 2;a++) {

for (b = 0;b < 2;b++) {

for (c = 0;c < 2;c++) {

for (x = 0;x < columns[k];x++) {

if (x < 4) y = columns[x] - k - 1;

else y = 8 - k;

fprintf(fp, "%d %d %d ", rows[x] + 9 \* y + A1[a], rows[x] + 9 \* y + B1[b], rows[x] + 9 \* y + C1[c]);

}

fprintf(fp, "0\n");

}

}

}

}

if (k == 6) {//倒数第三个右斜行

for (a = 0;a < 2;a++) {

for (b = 0;b < 2;b++) {

for (c = 0;c < 2;c++) {

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = x - (k - 4);

else y = 4 - (k - 4);

fprintf(fp, "%d %d %d ", rows[x] + 9 \* y + A1[a], rows[x] + 9 \* y + B1[b], rows[x] + 9 \* y + C1[c]);

}

fprintf(fp, "0\n");

}

}

}

}

if (k == 1) {//第二个右斜行

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

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

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

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

for (x = 0;x < columns[k];x++) {

if (x < 4) y = columns[x] - k - 1;

else y = 8 - k;

fprintf(fp, "%d %d %d %d ", rows[x] + 9 \* y + A2[a], rows[x] + 9 \* y + B2[b], rows[x] + 9 \* y + C2[c], rows[x] + 9 \* y + D2[d]);

}

fprintf(fp, "0\n");

}

}

}

}

}

if (k == 7) {//倒数第二个右斜行

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

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

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

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

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = x - (k - 4);

else y = 4 - (k - 4);

fprintf(fp, "%d %d %d %d ", rows[x] + 9 \* y + A2[a], rows[x] + 9 \* y + B2[b], rows[x] + 9 \* y + C2[c], rows[x] + 9 \* y + D2[d]);

}

fprintf(fp, "0\n");

}

}

}

}

}

if (k == 0) {//第一个右斜行

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

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

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

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

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

for (x = 0;x < columns[k];x++) {

if (x < 4) y = columns[x] - k - 1;

else y = 8 - k;

fprintf(fp, "%d %d %d %d %d ", rows[x] + 9 \* y + A3[a], rows[x] + 9 \* y + B3[b], rows[x] + 9 \* y + C3[c], rows[x] + 9 \* y + D3[d], rows[x] + 9 \* y + E3[e]);

}

fprintf(fp, "0\n");

}

}

}

}

}

}

if (k == 8) {//倒数第一个右斜行

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

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

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

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

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

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = x - (k - 4);

else y = 4 - (k - 4);

fprintf(fp, "%d %d %d %d %d ", rows[x] + 9 \* y + A3[a], rows[x] + 9 \* y + B3[b], rows[x] + 9 \* y + C3[c], rows[x] + 9 \* y + D3[d], rows[x] + 9 \* y + E3[e]);

}

fprintf(fp, "0\n");

}

}

}

}

}

}

}

//每个行必填相应的几个数字

//第一行必填5

//第二行必填4,5,6

//第三行必填3,4,5,6,7

//第四行必填2,3,4,5,6,7,8

//第五行必填1,2,3,4,5,6,7,8,9

//第六行必填2,3,4,5,6,7,8

//第七行必填3,4,5,6,7

//第八行必填4,5,6

//第九行必填5

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

for (z = 5;z - 5 <= columns[x] - 5;z++) {

for (y = 0;y < columns[x];y++) {

fprintf(fp,"%d ", rows[x] + 9 \* y + z);

}

fprintf(fp,"0\n");

}

}

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

for (z = 4;abs(z - 5) <= columns[x] - 5;z--) {

for (y = 0;y < columns[x];y++) {

fprintf(fp,"%d ", rows[x] + 9 \* y + z);

}

fprintf(fp,"0\n");

}

}

//每个左斜行必填相应的几个数字，同上

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

//printf("第%d个斜行---------------------------------------\n", k);

if (k < 5) {

for (z = 5;z - 5 <= columns[k] - 5;z++) {

for (x = 0;x < columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

else {

for (z = 5;z - 5 <= columns[k] - 5;z++) {

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

}

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

//printf("第%d个斜行---------------------------------------\n", k);

if (k < 5) {

for (z = 4;abs(z - 5) <= columns[k] - 5;z--) {

for (x = 0;x < columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

else {

for (z = 4;abs(z - 5) <= columns[k] - 5;z--) {

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = k;

else y = k + 4 - x;

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

}

//每个右斜行必填相应的几个数字，同上

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

//printf("第%d个斜行---------------------------------------\n", k);

if (k < 5) {

for (z = 5;z - 5 <= columns[k] - 5;z++) {

for (x = 0;x < columns[k];x++) {

if (x < 4) y = columns[x]-k-1;

else y = 8-k;

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

else {

for (z = 5;z - 5 <= columns[k] - 5;z++) {

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = x - (k - 4);

else y = 4 - (k - 4);

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

}

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

//printf("第%d个斜行---------------------------------------\n", k);

if (k < 5) {

for (z = 4;abs(z - 5) <= columns[k] - 5;z--) {

for (x = 0;x < columns[k];x++) {

if (x < 4) y = columns[x] - k - 1;

else y = 8-k;

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

else {

for (z = 4;abs(z - 5) <= columns[k] - 5;z--) {

for (x = k - 4;x < k - 4 + columns[k];x++) {

if (x < 4) y = x - (k - 4);

else y = 4 - (k - 4);

fprintf(fp, "%d ", rows[x] + 9 \* y + z);

}

fprintf(fp, "0\n");

}

}

}

fclose(fp);

return OK;

}

//采用挖洞法创造一个新的随机数独题

//num:挖洞的数量

status SudokuCreate(int num) {

char read;

int lines=0;

int holes[61]={0};

int hole;

int cnt=1;

int solut;

int i;

int temp;

int flag = FALSE;

FILE\* fp,\*open;

fp=fopen("easy\_hanidoku.txt", "r");

//统计文件行数，即数独题目数量

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

if (read == '\n') {

lines++;

}

}

fclose(fp);

srand((unsigned int)time(NULL));

SudokuPrepare("easy\_hanidoku.txt", rand()%lines+1);

//先对随机数独进行求解，得到数独终盘

DPLL\_BEFORE(FindLiteral1(r), 1);

AnswerComplete();

SudokuComplete();

DestroyParadigm(r);

count = 61;

//开始挖第一个洞

srand((unsigned int)time(NULL));

hole = rand() % 61;

while (holes[hole]){//该洞已被挖过

hole = rand() % 61;

}

holes[hole] = 1;

table[hole] = 0;

count--;

SudokuCreateFile();

open = fopen("SudokuTableBase.cnf", "r");

if (open == NULL) {

printf("文件打开失败！\n");

return ERROR;

}

if (CreateParadigm(&open) == ERROR) {//读取数独CNF公式文件并创建CNF公式数据结构

getchar();

getchar();

return ERROR;

}

fclose(open);

//挖剩下的洞

while (TRUE) {

flag = TRUE;

system("cls");

printf("\t\t请选择一个选项.\n");

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

printf(" 1.挖洞法随机生成新数独 2.打开指定数独文件\n");

printf(" 0.退出\n");

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

printf("已挖第%d/%d个洞\n", cnt,num);

for (i = 0;i < 61;i++) {

if (holes[i] == 1) {

continue;

}

flag = FALSE;

}

if (flag) {

break;

}

if (cnt >= num) {

break;

}

srand((unsigned int)time(NULL));

hole = rand() % 61;

while (holes[hole]) {//该洞已被挖过

hole = rand() % 61;

}

holes[hole] = 1;

temp = table[hole];

for (i = 1;i <= 9;i++) {//若挖去该洞后该洞内填入其他数字后，数独仍有解，说明挖洞不成功，回溯到挖洞前状态

if (i != temp) {

table[hole] = i;

SudokuCreateFile();

open = fopen("SudokuTableBase.cnf", "r");

if (open == NULL) {

printf("文件打开失败！\n");

return ERROR;

}

if (CreateParadigm(&open) == ERROR) {//读取数独CNF公式文件并创建CNF公式数据结构

getchar();

getchar();

return ERROR;

}

fclose(open);

if (solut=DPLL\_BEFORE(FindLiteral1, 1)) {

//printf("1");

table[hole] = temp;

DestroyParadigm(r);

break;

}

DestroyParadigm(r);

}

}

if (i == 10) {

table[hole] = 0;

count--;

cnt++;

}

}

for (i = 0;i < 61;i++) {

table0[i] = table[i];

}

SudokuCreateFile();

open = fopen("SudokuTableBase.cnf", "r");

if (open == NULL) {

printf("文件打开失败！\n");

return ERROR;

}

if (CreateParadigm(&open) == ERROR) {//读取数独CNF公式文件并创建CNF公式数据结构

getchar();

getchar();

return ERROR;

}

fclose(open);

}

//预处理数独文件，获取原始数独盘信息并创建数独CNF公式文件

//op值为选定数独盘在所在文件中的行数

status SudokuPrepare(char \*filename,int op) {

FILE\* fp, \* open;

char read;

int i = 0;

count = 0;//记录数独盘中非0数字个数

int flag = 0;//记录开始读取数独盘信息

int line = 0;//记录数独文件行信息，第几行即为第几个数独盘

int cnt = 1;

fp = fopen(filename, "r");

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

//读取数独文件中的数独信息

while (TRUE) {//数独盘未读取完

if (read == 'G') {

if (cnt == op) {//找到指定行的数独

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

flag = TRUE;

}

else {

cnt++;

}

}

if (flag){

if (read == '\n') {

break;

}

table[i] = read-'0';

table0[i] = read - '0';

i++;

if (read != '0') {//记录非0数字个数，便于后续计算CNF公式中子句个数

count++;

}

}

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

}

fclose(fp);

SudokuCreateFile();//创建数独CNF公式文件

open = fopen("SudokuTableBase.cnf", "r");

if (open == NULL) {

printf("文件打开失败！\n");

return ERROR;

}

if (CreateParadigm(&open) == ERROR) {//读取数独CNF公式文件并创建CNF公式数据结构

getchar();

getchar();

return ERROR;

};

//DPLL\_BEFORE(FindLiteral1(r), 1, 1);

//AnswerComplete();

//SudokuComplete();

//SudokuPrint();

return OK;

}

//将变元文字信息转换成数独数字，得到数独终盘信息

void SudokuComplete(void){

int i,j,k,s;

int x, y, z;

int columns[9] = { 5,6,7,8,9,8,7,6,5 };

int rows[10] = { 0,45,99,162,234,315,387,450,504,999 };

j = 0;

s = 0;

for (i = 1;i <= r->lsize;i++) {//此时变元表中真值为1的文字顺序与数独盘顺序相同

if (ValueList[i].Value == 1) {

for (k = 0;k < 10;k++) {

s = rows[k];

if (s >= i) {

x = k-1;

z = (i - rows[x]) % 9;

z == 0 ? z += 9 : z;

break;

}

}

table[j++] = z;

}

}

}

//绘制出数独终盘图像

void SudokuPrint(int\* tab) {

int columns[9] = { 5,6,7,8,9,8,7,6,5 };

int row, column;//记录当前行和列

int i;

int cnt = 0;

for (row = 0;row < 19;row++) {

for (i = 0;i < abs(9 - row);i++) {

printf(" ");

}

if (row % 2 == 0) {

if (row < 10) {

for (column = 0;column < columns[row / 2] \* 2;column++) {

if (column % 2 == 0) {

printf("/ ");

}

else {

printf("\\ ");

}

}

}

else {

for (column = 0;column < columns[(row - 1) / 2] \* 2;column++) {

if (column % 2 == 0) {

printf("\\ ");

}

else {

printf("/ ");

}

}

}

}

else {

if (row < 9) {

for (column = 0;column < columns[row / 2] \* 2 + 1;column++) {

if (column % 2 == 0) {

printf("|");

}

else {

if (tab[cnt] != 0) {

printf(" %d ", tab[cnt++]);

}

else {

cnt++;

printf(" . ");

}

}

}

}

else {

for (column = 0;column < columns[(row - 1) / 2] \* 2 + 1;column++) {

if (column % 2 == 0) {

printf("|");

}

else {

if (tab[cnt] != 0) {

printf(" %d ", tab[cnt++]);

}

else {

cnt++;

printf(" . ");

}

}

}

}

}

for (i = 0;i < abs(9 - row);i++) {

printf(" ");

}

printf("\n");

}

}

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <stdio.h>

#include "SAT.h"

int main(int argc, const char\* argv[]) {

//SudokuCreate();

//SudokuPrepare();

//SudokuCreateFile();

//getchar();

//getchar();

sum\_op = 1;

while (sum\_op) {

system("cls");

printf("\t\t请选择一个选项.\n");

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

printf(" 1.SAT求解 2.蜂窝数独求解\n");

printf(" 0.退出\n");

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

scanf("%d", &sum\_op);

switch (sum\_op) {

case 1:

SAT();

break;

case 2:

Sudoku();

break;

case 0:

break;

default:

printf("输入错误！");

getchar();

getchar();

break;

}

}

return 0;

}