

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 程序设计综合课程设计**

**专业班级： CS1704班**

**学 号： U201714633**

**姓 名： 杨俊祎**

**指导教师： 袁凌**

**报告日期： 2019年3月5日**

**计算机科学与技术学院**

**任 务 书**

**设计内容**

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

**设计要求**

要求具有如下功能：

1. **输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)
2. **公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)
3. **DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)
4. **时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)
5. **程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)
6. **SAT应用：**将数独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

目录

[1 引言 3](#_Toc5647383)

[1.1 课题背景与意义 3](#_Toc5647384)

[1.2 国内外研究现状 4](#_Toc5647385)

[1.3 课程设计的主要研究工作 5](#_Toc5647386)

[2 系统需求分析与总体设计 6](#_Toc5647387)

[2.1 系统需求分析 6](#_Toc5647388)

[2.2 系统总体设计 6](#_Toc5647389)

[3 系统详细设计 7](#_Toc5647390)

[3.1 有关数据结构的定义 7](#_Toc5647391)

[3.2 主要算法设计 8](#_Toc5647392)

[4 系统实现与测试 13](#_Toc5647393)

[4.1 系统实现 13](#_Toc5647394)

[4.2 系统测试 27](#_Toc5647395)

[5 总结与展望 38](#_Toc5647396)

[5.1 全文总结 38](#_Toc5647397)

[5.2 工作展望 38](#_Toc5647398)

[6 体会 40](#_Toc5647399)

**1 引言**

**1.1 课题背景与意义**

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。SAT问题也是程序设计与竞赛的经典问题。

**1.1.1 SAT问题研究背景**

SAT一直备受人们的关注。从1960年至今，世界各国的研究人员在此方面都做了大量的工作，提出了许多求解算法。每年可满足性理论和应用方面的国际会议都会组织一次SAT竞赛以求找到一组最快的SAT求解器，而且会详细展示一系列的高效求解器的性能。国内也会经常组织一些SAT竞赛及研讨会，这些都促进了SAT算法的飞速发展。尽管命题逻辑的可满足性问题理论研究已趋于成熟，但在SAT求解器被越来越多地应用到各种实际问题领域的今天，探寻解决SAT问题的高效算法仍然是一个吸。引人并且极具挑战性的研究方向。

**1.1.2 SAT问题研究意义**

可满足性问题是计算机科学领域和人工智能等领域中的重要研究对象，但是其求解算法的时间开销和空间开销却异常惊人。对于一个含有n个命题逻辑变量的合取范式来说，如果使用穷举法来罗列所有真值指派进行求解，虽然理论上是可行的，但算法的时间复杂度却是指数级规模，计算机将负担不起这么大的开销。搜索空间如此之大，使得计算机在可等待的时间内不能计算出结果，进而产生组合爆炸问题，所花费的计算时间是人们不可忍耐的。S.A.Cook与1971年受此证明了布尔表达式的可满足性问题属于NP完全问题。

现实生产和生活中存在着大量的NP完全问题，寻找高效快速的算法来解决该类问题是计算机理论研究和实际应用领域中的重要工作。由于SAT问题是NP完全问题。如果能够将它高效解决，那么一定可以高效地解决所有其他NP完全问题，这是因为所有的NP完全问题都能在多项式时间内进行相互转化。所以设计和实现更高效的求解算法意义重大。

SAT问题的应用非常广泛，例如在数学研究和应用领域，它能用来解决旅行商和逻辑算数问题；在计算机和人工智能领域中，他能解决CSP问题、语义信息的处理和逻辑编程问题；在计算机辅助设计领域中，它能很好地解决任务规划与设计、三维物体识别等问题。许多实际的问题如人工智能、积木世界规划问题、数据库检索、Job shop排工问题、超大规模集成电路设计和图着色都可转化为SAT问题进行求解。

**1.2 国内外研究现状**

由于SAT问题在计算机理论研究和人工智能领域的重要作用，使得许多学者都在SAT问题求解领域做了大量的研究，可满足性问题进而也成为了国内外研究的热点问题，并且在算法研究和技术实现上取得了较大的突破，这也推动了形式验证和人工智能等领域的发展。

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现状进行了全面的阐述和总结。这十大挑战性问题的提出对于SAT基准问题的理论研究和算法改进都起到了强有力的推进作用。

最经典的求解SAT问题的完备算法是DPLL算法，它是由Davis和Putnam等人在1960年提出，其他的完备算法大都是在DPLL算法的基础上衍生出来的，是对DPLL算法的改进。在1996年以后，很多国家相继举办了一些SAT竞赛和研讨会，这使得越来越多地人开始关注并研究SAT问题，所以这段时间也涌现出了众多新的高效SAT算法如MINISAT、SATO、CHAFF、POSIT、和GRASP等，其中大部分新兴的算法是基于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年梁东敏提出的改进的子句加权WSAT算法，2000年金人超和黄文奇提出的并行Solar算法，2002年作者张德福提出的模拟退火算法。

尽管SAT算法已经取得了举足轻重的改进，但是仍有一些问题没有得到高效的解决，已经解决的问题可能还存在更好的求解算法，因此研究并实现高效率的求解算法仍是当前要解决的中心问题之一。

**1.3 课程设计的主要研究工作**

**1.3.1 构建一个基于DPLL算法的SAT求解器**

本次课程设计构造了一个基于DPLL算法的非递归SAT求解器，来对子句集的可满足性问题进行求解。构造数据结构的思路如下：子句表示为由文字构成的链表；而整个公式又是由子句构成的链表，如图1-1所示。



图1-1

构造好数据结构以后，完成一些实现DPLL算法所必需的函数，例如删除含有文字L的子句、删除每个子句中的文字-L、判断是否为空子句等等，为实现DPLL算法做准备。

由于采用了非递归算法，因此本次课设借助了一个栈，用于实现回溯。

**1.3.2 构造一个数独的初始格局**

本次实验要求构造一个随机的数独并且进行求解，因此每次游戏就需要构造一个随机的数独终局，再进行挖洞。

**1.3.3 将数独问题转化为SAT问题**

根据一定的规律，将数独转换为cnf文件，求解出cnf的解，在将解转换为数独的解，对数独进行填空，从而解决数独问题。

--------章与章之间插入分页符----------

**2 系统需求分析与总体设计**

**2.1 系统需求分析**

本系统用于求解数独。该系统将数独问题转化为SAT问题，并且将信息存在cnf文件中。利用系统的SAT求解功能，将cnf文件中的SAT问题的解保存至res文件中。系统再读取res文件的内容，运用来对数独题目进行填空，即可实现对数独的求解。

**2.2 系统总体设计**

基于上述系统的需求，将本次系统分为4个模块：

主控模块：可以进行用户的交互、控制、显示，主要是供用户进行操作

SAT解析模块：读取cnf文件，构成子句集，

SAT求解模块：可以读取cnf文件，对SAT问题利用DPLL算法进行求解。

数独模块：可以进行数独初始格局的随机生成、数独问题转化为SAT问题，读取SAT的解以对数独进行填空。

具体的使用如图2-1所示

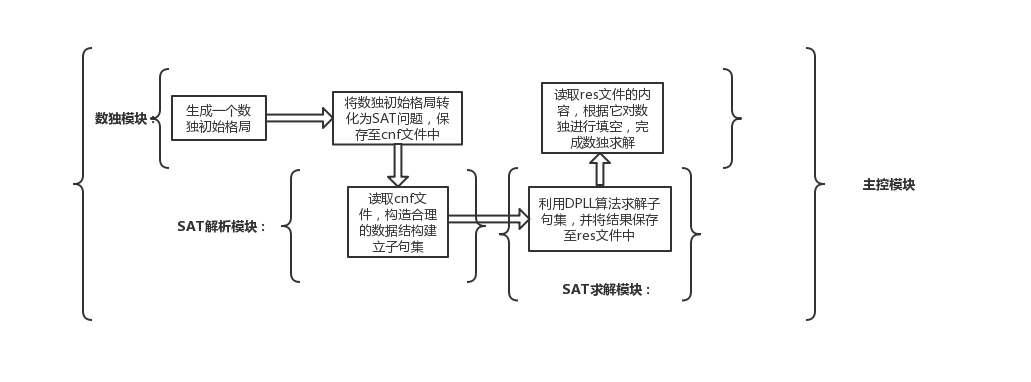


图2-1模块的示意图

--------章与章之间插入分页符----------

**3 系统详细设计**

**3.1 有关数据结构的定义**

这部分要写的：（1）首先描述系统中要处理那些数据，每种类型的数据包括哪些数据项，每个数据项的数据类型，最后可用一个表格表示出来；（2) 描述这多种数据在系统中如何关联，可通过图直观的说明这多种数据间的关联。

**3.1.1 要处理的数据**

类型名声明：

typedef int status;

typedef int ELemType;//变元的命名

SAT求解模块与解析模块中中：

头结点：

typedef struct HeadNode{

int data;//数据域存放子句的长度

int all\_data;//该子句初始状态的长度

struct HeadNode\* down;//下一个子句

struct Node\* next;//指向该子句的第一个节点

status is\_headnode\_hidden;//状态：该子句是否被删除

ELemType why;//如果子句被删除，则记录第一次被删除是因为哪个文字。若未被删除则为0

}HeadNode;

数据节点：

typedef struct Node{

ELemType data;//节点包含数据域

struct Node\* next;//指针域

status is\_node\_hidden;//状态：节点是否被删除

}Node;

子句数目：int ziju\_num

保存的总共的子句数目：all\_ziju\_num

变元数：v\_num

存放满足的文字的栈：Stack[10000][2]。每一个Stack[a][0]存放满足的文字，Stack[a][1]存放该文字是否被否定过。

**3.2 主要算法设计**

这部分主要描述系统中的模块实现的流程，可采用文字配合流程图的方式表示各模块的算法思想及流程。

**3.2.1 主控模块**

该模块主要用于与用户进行交互，用户的操作主要就是依靠主控模块实现的。程序一开始运行就打开的是主控模块。先显示一个界面，用户选择使用某一项功能，然后进行调用，从而实现功能。

**3.2.2 SAT解析模块**

该模块用于从cnf文件中读取子句集，建立二维的链表：首先从cnf文件中读取子句的个数，然后建立同样数量的HeadNode，然后依次读取每一个子句的内容，同时依次建立Node，这样就建立了一个二维的链表，子句集就构建完成了，具体流程图如下图3-1：

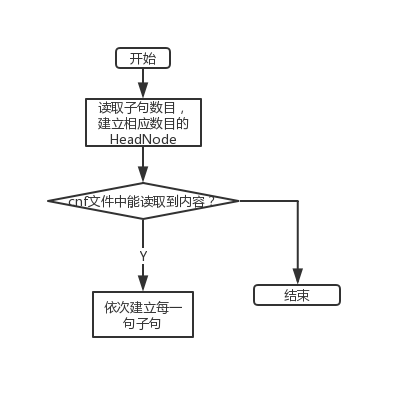


图3-1 SAT解析模块的流程图

**3.2.3 SAT求解模块**

该模块可以实现SAT-cnf问题的求解。

DPLL：本次采用了非递归的DPLL算法。具体的操作如下：只要子句集非空，就选择最短子句的第一个文字q，假设其为真，并让该文字入栈。删除含有该文字q的子句以及各子句中的-q。然后再删除单子句，并让这些单子句的文字v入栈，再删除含有v的子句以及各子句中的-v。然后再进行判断，只要有空子句，就说明之前的某次假设可能有错误。找到离栈顶最近的一个没有反置过的文字u，恢复u的影响（即将删除的含u的子句及-v进行恢复），并将栈顶元素到u之间的文字全部出栈，并且依次消除这些文字的影响。（如果连栈底元素都反置过而依然发生了错误，说明子句集不可满足）反置以后，-u入栈，并且删除含-u的子句以及u文字。然后重新去寻找最短子句的第一个文字q，依次循环下去，如果过程中子句集为空，则子句集可满足，可满足的赋值即为栈中的文字。具体的流程图如下图3-2：

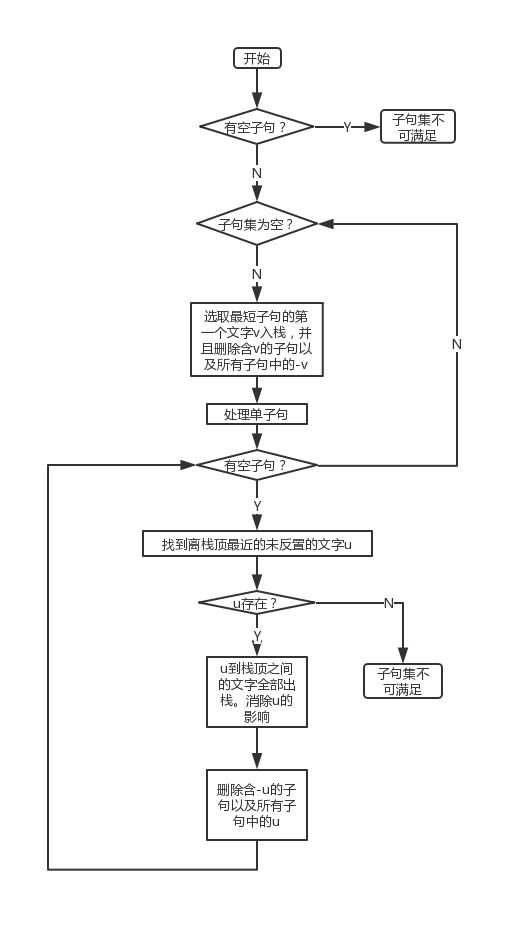


图3-2 SAT求解模块的流程图

**3.2.4 数独模块**

本模块可以进行随机数独的生成、数独问题转SAT问题以及数独的求解，具体的算法如下：

数独初始格局的生成：首先读取一个已知的数独终局，然后将该终局的数字1-9重新随机分配，这样就可以生成9！种不同的数独终局。如果读取不同的数独终局，即使有大量的重复，也一定会有数量极大的不同格局，这样就相当于实现了数独的随机生成。数独终局生成完毕以后，再选择挖洞的数量，然后进行随机挖洞，这样就生成了一个数独的初始格局。

数独转SAT：对于数独的初始格局，填空时有如下的要求：

（1）对于每一个空格而言，填的数字可以是1-9，但不能与所在的行、列、九宫格中已经存在的数字相同。

（2）对于每一行、列、九宫格中能填相同数字的空格而言，一个数字只能出现一次。

（3）对于每一个空格而言，如果能填多个数字，则最后只能填入一个，即任意两个数字不能同时出现。

以上三条是需要满足的条件。现对于所有空格能填的数字从1开始进行递增编号。然后根据以上三条规则，一一写出对应的子句，并且统计出子句的数目，最后保存至一个cnf文件中。

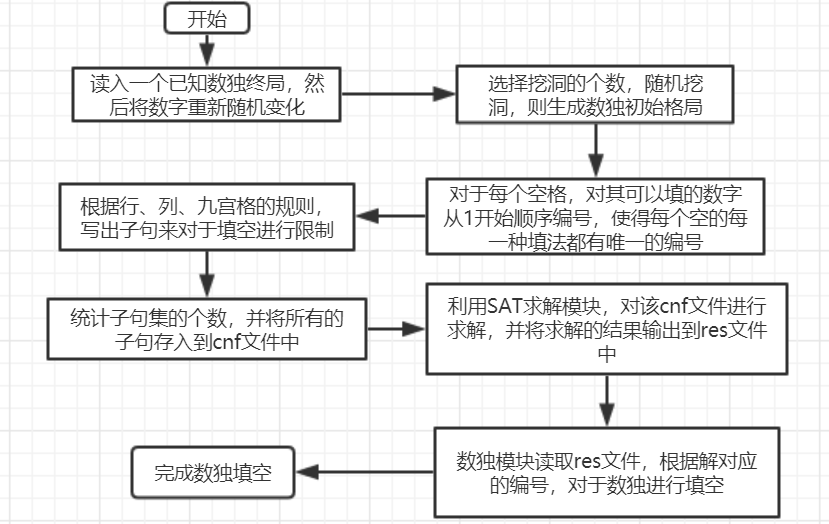
数独的求解：数独的求解要借助于SAT解析和求解模块，具体如下：

（1）利用SAT求解模块解析数独转SAT生成的cnf文件，再用SAT求解模块进行求解，把结果输出至res文件中。

（2）由于输出的结果对应的是编号，而每一个编号又唯一对应于一个空格中的一个数字填法，因此根据res文件的内容，可以对数独进行填空，从而实现对数独的求解。

该模块的流程图如下图3-3：

图3-3 数独模块的流程



--------章与章之间插入分页符----------

**4 系统实现与测试**

**4.1 系统实现**

**4.1.1 用户需求**

系统实现的目标：随机生成数独并对其进行求解。

事务处理流程：随机生成数独初始格局、将数独问题转化为SAT问题、用SAT求解器进行求解、利用求解结果对数独进行填空，从而实现数独的求解。

**4.1.2 软硬件环境**

硬件：8G内存，256G硬盘

软件：visual studio code

**4.1.3 数据类型**

（1）SAT.c中：

头结点：

typedef struct HeadNode{

int data;//数据域存放子句的长度

int all\_data;//该子句初始状态的长度

struct HeadNode\* down;//下一个子句

struct Node\* next;//指向该子句的第一个节点

status is\_headnode\_hidden;//状态：该子句是否被删除

ELemType why;//如果子句被删除，则记录第一次被删除是因为哪个文字。若未被删除则为0

}HeadNode;

数据节点：

typedef struct Node{

ELemType data;//节点包含数据域

struct Node\* next;//指针域

status is\_node\_hidden;//状态：节点是否被删除

}Node;

其他数据：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 数据名称 | 用途 | 说明 |
| int | ziju\_num | 记录当前子句的数量 |  |
| int | all\_ziju\_num | 所有子句的数量 | 由于删除子句只是将其状态改成了“隐藏”，设置这个变量可以在后期对所有的子句进行访问 |
| int | v\_num | 变元数量 |  |
| Char[ ] | filename | 读取文件的名称 | 用于从cnf文件中读取子句集 |
| int [ ][ ] | Stack | 用于存放假设成立的文字 | Stack[a][0]用于存放文字，Stack[a][1]用于存放该文字是否反置过，是则为1，以此来找到最近的未反置的文字，便于回溯 |
| clock\_t | start, stop | 用于统计DPLL的开始和结束时间 | 分析DPLL的时间性能 |
| long | duration | DPLL持续运行的时间 | 分析DPLL的时间性能 |
| HeadNode\* | root | 二维链表的头指针 |  |
| HeadNode\* | tail | 二维链表的尾指针 |  |
| HeadNode\*\* | root\_p | 头指针的指针 |  |
| HeadNode\*\* | tail\_p | 尾指针的指针 |  |
| int | op | 主控界面时用户选择的功能序号 | 通过一个while循环，让用户反复选择所要的功能，输入0退出 |
| int | choice | 用户选择是否将SAT的解输出至res文件 | 该系统提供了一个选择，不想输出结果至文件时可以不输出 |

子句数目：int ziju\_num

保存的总共的子句数目：all\_ziju\_num

变元数：v\_num

存放满足的文字的栈：Stack[10000][2]。每一个Stack[a][0]存放满足的文字，Stack[a][1]存放该文字是否被否定过。

（2）new\_Sudoku.c：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 数据名称 | 用途 | 说明 |
| int[ ][ ] | Sudoku | 用于存放数独 | 声明的时候是Sudoku[10][10]，第0行和第0列不使用，访问均从1开始 |
| int [ ][ ] | dignumber | 存放挖洞格子的行数和列数 | 声明的时候是dignumber[81][2]  其中dignumber[a][0]存放洞的行数  Dignumber[a][1]存放洞的列数 |
| int[ ][ ] | changenumber | 用于记录改变数字时改变前后的对应关系 | 声明的时候是changenumber[9][2]，其中changenumber[a][0]中存放的是原数字+10，如1就是11，changenumber[a][1]是变换后的数字。后期可以按照这个表将原有数独进行修改，变成随机数独 |
| int[ ][ ] | chosen | 用于记录改变以后的数字中哪些已经被选择了 | 数独中的数字被重新换成1-9，这9个数字依次被选择，且只能被选择一次，因此要记录哪些数字被选择了，则接下来就在剩下的数字中进行选择 |
| int | hole\_num | 利用挖洞法挖出空格，hole\_num用于记录当前洞的数量 |  |
| int | all\_hole\_num | 用挖洞法总共挖出的洞的数量 |  |
| int[ ][ ][ ] | SAT\_table | 用于存放数独转SAT的信息 | 对于每一个空格，根据数独的规则，有若干可以填的数字，根据之前的描述，从1开始给所有空格能填的数字不重复地编号，SAT[a][0][0]为空格的行号，SAT[a][0][1]为空格的列号  SAT[a][b][0]为对应的空格能填的数字（b!=0）  SAT[a][b][1]为对应的空格能填的数字所对应的序号（b!=0） |
| int | ziju\_num | 输出成cnf文件后的子句数量 |  |
| int | v\_num | 输出成cnf文件后的文字数量 |  |
| int[ ] | same\_number | 用于处理编号问题时暂存所用 | 在处理能存放同一数字的空格中同一数字的编号等问题时，可以将其暂时存于该地方，此后便可以利用该数组进行其他处理，会减少循环的次数，方便处理 |

**4.1.4 使用的函数**

（1）SAT.c中，使用了如下一些函数

status CreateClause(HeadNode\*\* root, HeadNode\*\* tail);//创造子句集

status DestroyClause(HeadNode\*\* root, HeadNode\*\* tail);//删除子句集(这个是真的删除，会free)

status AddClause(HeadNode\*\* root, HeadNode\*\* tail);//添加子句

status RemoveClause(HeadNode\*\* root, HeadNode\*\* tail, int n);//删除第n个子句

void Show\_ziju\_num(void);//显示子句个数

void ShowAllClause(HeadNode \*root);//展示所有的子句

status isUnitClause(HeadNode \*root);//判断子句集中是否存在单子句

int firstUnitClause(HeadNode \*root);//返回子句集中第一个单子句的序号(包含已隐藏的)

int Special\_lit\_place(HeadNode \*p, int lit);//找到当前子句中第一个为lit的文字的位置(包含隐藏的节点)，没有则返回0

status Dele\_special\_lit(HeadNode\*\*root,int lit);//在所有子句中删除指定的文字

status isBlankClause(HeadNode \*root);//判断子句集中是否有空子句

int firstBlankClause(HeadNode\* root);//返回子句集中第一个空子句的序号(包含被隐藏的子句)

status isSetBlank(void);//判断子句集是否为空

status DPLL(HeadNode \*root, HeadNode\* tail);//递归的DPLL算法

status RemoveSpecialClause(HeadNode \*\*root, HeadNode\*\* tail, int L);//删除包含文字L的子句

status Recover(HeadNode \*\*root,ELemType v);//恢复包含v的子句，以及包含-v的节点

status is\_lit\_in\_clause(HeadNode \*p, ELemType lit);//判断lit是否在子句中(被隐藏了也算在里面！！！！！)

status Read\_to\_Create(FILE \*fp, char \*filename, HeadNode \*\*root, HeadNode \*\*tail);//读取cnf文件并且创建子句集

status SaveFile(FILE \*fp, char \*filename, int duration, int ans);//将DPLL解析的信息存入文件中

void Analyze\_cnf(HeadNode\* root);//对cnf文件进行解析

status Validate\_ans(HeadNode \*root, HeadNode \*tail);//用栈中的变元验证cnf文件

（2）new\_Sudoku.c中：

status InitFinalSudoku(FILE \*fp);//初始化生成一个终局

void ShowSudoku(void);//输出数独格局

status MakeRandomSudoku(void);//将初始化了的终局进行改变，生成随机的终局

status ChangeNum(void);//将原有数独做相应的变换

status isNumberChosen(int number);//判断变换数字的时候数字是否已存在于数组chosen中

status DigHole(void);//对已经生成的数独进行挖洞

status sudoku\_to\_SAT(void);//将数独转换为SAT问题的信息存至数组中

status Can\_exist\_in\_line(int number, int i);//判断number在第i行中是否已存在，存在则返回NO

status Can\_exist\_in\_col(int number, int j);//判断number在第j列中是否已存在，存在则返回NO

status Can\_exist\_in\_square(int number,int i, int j);//判断number在空格所处的九宫格是否已经存在，如果存在则返回NO

status SAT\_to\_cnf\_file(FILE \*fp, char \*cnf\_filename);//将表中的SAT信息转换成cnf文件

status Solve\_Sudoku(FILE \*fp, char \*res\_filename);//将SAT解的结果读至数独中填空

**4.1.5 各函数的说明**

SAT.c中：

（1）status CreateClause(HeadNode\*\* root, HeadNode\*\* tail)

该函数用于手动创建子句集。然后通过循环创建HeadNode的链表，长度为ziju\_num，然后再循环输入每个子句的内容，每个子句以0结尾，然后会输入下一个子句的内容。当输入完ziju\_num个子句的内容以后，创建完成，返回OK。

（特殊情况1）如果根节点\*root已存在，则证明子句集已经创建了，返回INFEASTABLE。

（2）status DestroyClause(HeadNode\*\* root, HeadNode\*\* tail)

该函数用于销毁子句集。函数将根节点\*root以及尾节点\*tail全部free掉，将ziju\_num和all\_ziju\_num全部置0，并且返回OK。

（特殊情况1）如果\*root=NULL，则子句集尚未创建，不能销毁，返回INFEASTABLE。

（3）status AddClause(HeadNode\*\* root, HeadNode\*\*tail)

该函数用于向子句集中添加子句到子句集的尾部，用户输入要添加的子句的内容（以0结尾），然后创建一个子句添加到子句集的\*tail后面，并且以新添加的子句为新的\*tail，ziju\_num加1，返回OK。

（4）status RemoveClause(HeadNode\*\* root, HeadNode\*\* tail, int n)

该函数用于删除子句集中第n个子句，其中n这个序号在计数时被隐藏的子句也被包含进去。通过循环让p指向第n个子句，然后将其is\_headnode\_hidden改成YES，ziju\_num减1，返回OK。

（特殊情况1）如果n大于all\_ziju\_num，则一定不存在这个子句，函数返回ERROR。

（5）void Show\_ziju\_num(void)

该函数用于输出子句的数量，直接输出zijunum。

（6）void ShowAllClause(HeadNode \*root)

该函数用于遍历输出子句集。通过循环控制，输出is\_headnode\_hidden=NO的子句。

（7）status isUnitClause(HeadNode \*root)

该函数用于判断子句集中是否有单子句。通过循环查找，如果有子句的头结点值为1且is\_headnode\_hidden=NO，则返回TRUE；如果找遍了也没有符合条件的，则返回FALSE。

（8）int firstUnitClause(HeadNode \*root)

该函数用于查找第一个单子句的序号。通过遍历子句集，如果有子句的头结点值为1且is\_headnode\_hidden=NO，则返回该子句的序号。

（9）int Special\_lit\_place (HeadNode \*p, int lit)

该函数用于查找指定子句p中指定文字lit的位置。通过遍历找到该子句中lit所处节点的序号（计数时被隐藏的节点也被计数在内），并且将其返回。

（特殊情况1）如果该子句中不存在文字lit，则返回0。

（10）status Dele\_special\_lit(HeadNode \*\*root, int lit)

该函数用于删除子句集中所有的lit文字。通过遍历所有子句，并且调用函数Special\_lit\_place (HeadNode \*p, int lit)，如果处理的子句中含有lit，则通过lit的序号找到那个节点并且将其删除（即把is\_node\_hidden改成YES并且让p->data减1）；如果不含lit，则处理下一个子句。如果处理完了所有子句，就返回OK。

（11）status isBlankClause(HeadNode \*root)

该函数用于判断子句集中是否含有空子句。通过遍历所有子句，如果有子句长度为0并且is\_headnode\_hidden为NO，代表有空子句，则返回TRUE。如果找遍子句集都不满足条件，则返回FALSE。

（12）status isSetBlank(void)

该函数用于判断子句集是否为空。直接判断ziju\_num，如果其为0，则子句集为空；否则子句集非空。

（13）status RemoveSpecialClause(HeadNode \*\*root, HeadNode\*\* tail, int L)

该函数用于删除含文字L的子句。函数遍历子句集，如果某个子句中含有文字L，则将该子句删除，并且将该子句headnode的why改成L（这里的why是用于记录该子句被删除的原因，如果子句处于未被删除的状态则why为0），然后继续处理下一个子句，直到处理完所有的子句，然后返回OK。

（14）status is\_lit\_in\_clause(HeadNode \*p, ELemType lit)

该函数用于判断文字lit是否存在于子句p中。遍历整个子句p，如果lit存在于p中，则返回TRUE；否则就返回FALSE

（15）status Recover(HeadNode \*\*root,ELemType v)

函数用于恢复子句集到处理文字v之前的状态。由于DPLL算法需要回溯，因此在回溯的过程中，需要将之前处理过的v还原，总的来说就是将删除的含v的子句恢复，同时将文字-v恢复。具体做法如下：遍历整个子句集，如果头结点headnode的is\_node\_hidden为YES并且该子句中含有文字v，则将is\_headnode\_hidden改为NO，且ziju\_num增加。再遍历整个子句集，如果某个node的is\_node\_hidden为YES且node的内容为-v，则将该节点的is\_node\_hidden改为NO，并且让该子句的headnode的data加1。处理完整个子句集以后，返回OK。

（16）status DPLL(HeadNode \*root, HeadNode \*tail)

该函数为核心函数DPLL，即利用DPLL算法对子句集进行求解。当子句集非空的时候，找到子句集中的最短子句的第一个文字v，假设其为真，让其入栈然后根据单子句规则，以v进行化简。然后只要有单子句，就让单子句的文字入栈，并且根据单子句规则做处理。然后只要有空子句，就说明之前有假设错误，就用Recover函数恢复栈顶元素造成的影响。如果栈顶元素没有反置过，就让栈顶元素反置，再以-v对子句集进行处理；否则就让栈内元素出栈，直到找到一个没有反置过的元素（如果这个元素不存在，则子句集不可满足），对其进行反置处理。然后再判断是否有空子句，如此循环。如果某一次使得子句集为空，则子句集可以满足，返回OK。具体的流程图如下图4-1所示：

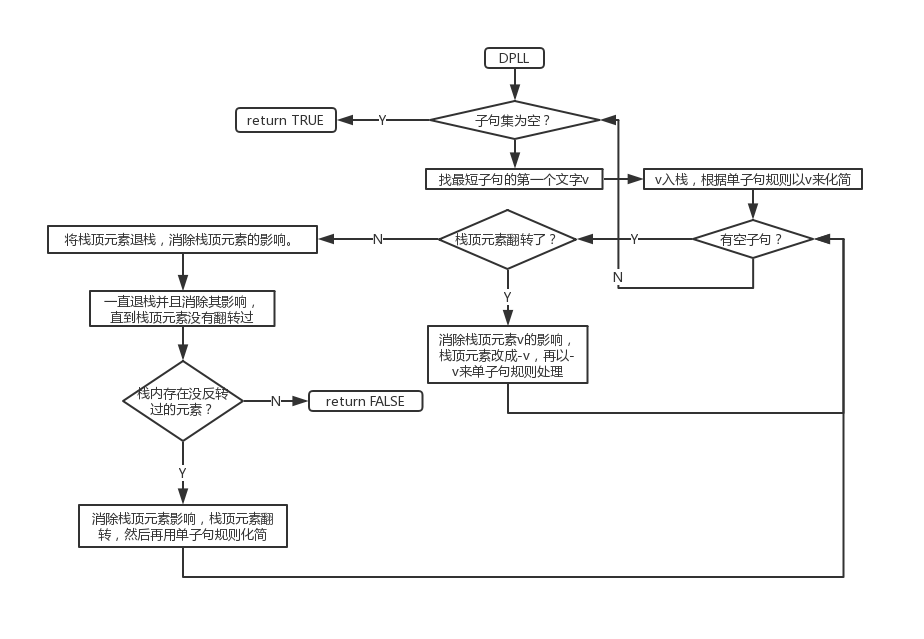


图4-1 DPLL函数的流程图

（17）status Read\_to\_Create(FILE \*fp, char \*filename, HeadNode \*\*root, HeadNode \*\*tail)

该函数用于读取cnf文件从而创建子句集。以读的方式打开文件，然后每次读取6个字符，如果不是‘p cnf ’，就将本行读完，然后读下一行。如果是的话，就代表后面的两个数字分别为文字数和子句数。建立子句数个头结点headnode。然后开始读取内容，读到0代表该子句的结束。如此读取所有的子句，建立子句集完成，返回OK

（18）status SaveFile(FILE \*fp, char \*filename, int duration, int ans)

该函数将DPLL解出的解保存到res文件中。如果有解，则会按格式依次输出解的结果、解所用的时间。如果无解，则输出0

（19）void Analyze\_cnf(HeadNode\* root)

该函数用于解析子句集，即让子句集以集合的形式写出来，这一点与函数ShowAllClause十分类似，只不过是要以集合的形式输出，而不是中间加空格，因此不再赘述。

（20）status Validate\_ans(HeadNode \*root, HeadNode \*tail)

该函数用于验证子句集。假设子句集是可满足的。首先输出解析的结果，再输出赋值的结果，然后根据栈中的内容，依次从栈底开始利用单子句规则化简。如果化简完毕以后为空子集，则验证成功，否则验证失败。

new\_Sudoku.c中：

（1）status InitFinalSudoku(FILE \*fp)

该函数用于读取一个已知的数独终局（保存在initial\_sudoku.txt）中，以便后续进行变换生成随机数独。函数循环扫描81次，完成初始数独终局的构造，返回OK。

（2）void ShowSudoku(void)

该函数用于输出现有的数独格局。函数通过遍历数独的二维数组，循环输出81次，从而输出整个数独的内容。

（3）status ChangeNum(void)

该函数用于将之前读入的已知的数独终局的数字1-9重新进行分配，以达到生成随机数独的目的。函数利用随机数函数rand()以及随机数种子srand(time(NULL))，生成1-9的随机数，然后将刚生成的数（必须与之前生成的都不同）放到chosen数组中，形成一一对应的关系。最后，将一一对应的关系体现在数组changenum[][]中（changenum[a][0]是原有数字，changenum[a][1]是变换后的数字）。

（4）status isNumberChosen(int number)

函数用于判断在变换数字的过程中number是否已经被选了。函数遍历数组chosen[]，如果number在里面，则代表已经被选了，返回YES；否则返回NO。

（5）status MakeRandomSudoku(void)

该函数用于对初始数独进行变换，从而生成随机数独。首先调用函数ChangeNum()，生成数字变换的一一对应关系。然后通过遍历数独，对于每一个一一对应的数字变换关系都进行变换，这样就生成了一个随机数独。最后返回OK。

（6）status DigHole(void)

该函数用于给已知数独终局进行挖洞，从而生成最后的数独题目。首先提示用户输入要挖洞的数量，即all\_hole\_num，然后借助随机数种子以及随机数函数rand()随机生成要挖洞的行号和列号，然后直接将这一格的内容变成0，代表已经挖了一个空格。挖完以后，返回OK。

（7）status Can\_exist\_in\_line(int number, int i)

该函数判断数字number能不能填在第i行的空格中。如果第i行已经有某个数字了，那么这一行的空格中就不能再填这个数字。因此函数通过遍历第i行，来判断number是否能填进去，能则返回OK，不能则返回NO。

（8）status Can\_exist\_in\_col(int number, int j)

该函数判断数字number能不能填在第j列的空格中。如果第j列已经有某个数字了，那么这一列的空格中就不能再填这个数字。因此函数通过遍历第j列，来判断number是否能填进去，能则返回OK，不能则返回NO。

（9）status Can\_exist\_in\_square(int number,int i, int j)

该函数用于判断数字number能否存在于第i行第j列所在的九宫格。如果九宫格中已经有某个数字了，那么这一九宫格的空格中就不能再填这个数字。因此函数通过遍历这个九宫格，来判断number是否能填进去，能则返回OK，不能则返回NO。

（10）status sudoku\_to\_SAT(void)

该函数用于将数独转化为SAT的信息。函数遍历整个数独，对于每个空格，则通过判断填的数字能否存在于行、列、九宫格来判断每个空格能填哪些数，并且对其一一进行编号，将结果存放在SAT\_table[][][]中。其中SAT\_table[a][0][0]存放空格的行号，SAT\_table[a][0][1]存放空格的列号。SAT\_table[a][b][0]存放能填的数字，SAT\_table[a][b][1]存放数字对应的序号。完成以后返回OK。

（11）status SAT\_to\_cnf\_file(FILE \*fp, char \*cnf\_filename)

该函数用于将之前保存在SAT\_table[][][]中的信息转换之后生成一个cnf文件，也就是将要满足的条件翻译成子句。总体来说，有以下几个层面的要求：

1. 对于一个空格来说，所有能填的数最后至少有一个要填进去。在子句层面，如果这个空格能填3个数，序号分别为4,5,6，则最后翻译成的子句为4 5 6 0
2. 对于一行、一列、一个九宫格中，如果有多个空格都能填一个数字，则至少有一个空格最终要填这个数字。在子句层面，如果某一行（列、九宫格）中，有三个空格都能填相同数字，对应的序号为22,27,29，则对应的子句为22 27 29 0
3. 对于一行、一列、一个九宫格中，如果有多个空格都能填一个数字，则最后只能有一个空格填入该数字。在子句层面，如果如果某一行（列、九宫格）中，有三个空格都能填相同数字，对应的序号为22,27,29，则对应的子句为-22 -27 0 -22 -29 0 -27 -29 0 总共三句。
4. 对于每个空格，最后只有一个数能最终填进去。在子句层面，如果这个空格能填3个数，序号分别为4,5,6，则最后翻译成的子句为-4 -5 0 -4 -6 0 -5 -6 0 总共三句。

打印完所有的子句以后，要将统计的子句数目及挖洞的数目注释在cnf文件的前面，于是调用函数rewind（fp），将注释添加上去，就完成了，返回OK

（12）status Solve\_Sudoku(FILE \*fp, char \*res\_filename)

该函数用于将数独填空。之前的函数SAT\_to\_cnf\_file（）将数独的问题转化成了cnf文件，在利用SAT.c求解了以后，将输出的res文件读取，主要是读取其文字的赋值。其文字的赋值又对应于数独中序号的赋值，然后再SAT\_table中寻找对应的序号赋值所对应的空格以及要填的数字，然后一一填入数独中即可完成数独的填空。

**4.1.6 函数实现模块**

（1）主控模块：主控模块是利用一个switch循环，让用户选择要是用的功能，然后调用对应的函数即可。

（2）SAT解析模块：该模块首先要利用读取创建函数Read\_to\_Create()或者自行手动创建子句集的函数CreateClause()创建一个子句集，然后调用函数Analyze\_cnf()，对子句集进行解析即可实现模块的功能。

（3）SAT求解模块：该模块首先要利用读取创建函数Read\_to\_Create()或者自行手动创建子句集的函数CreateClause()创建一个子句集， 然后在调用函数DPLL()实现对函数的求解。最后还可以选择将解输出至res文件中，调用函数status SaveFile(FILE \*fp, char \*filename, int duration, int ans)即可实现

（4）数独模块：该模块首先调用函数InitFinalSudoku（）读取一个已知的数独终局initial\_sudoku.txt，然后调用函数MakeRandomSudoku（）将数独中的数字1-9重新进行随机分配，生成一个随机的数独终局。然后再调用函数DigHole（）进行挖洞，构成数独的题目。再调用函数Sudoku\_to\_SAT（）将数独题目转SAT的信息保存至SAT\_table[][][]中。然后调用函数SAT\_to\_cnf\_file（）将其内容输出成cnf文件。利用SAT求解模块读取该cnf文件进行求解，并且将解输出到res文件中。最后利用函数Solve\_Sudoku（）读取res文件并且根据结果对数独进行填空，完成对数独的求解。

**4.1.7 函数间的调用关系**

SAT.c中：

（1）DPLL（）函数调用了函数RemoveSpecialClause（）、Dele\_special\_lit（）、firstUnitClause（）、isBlankClause（）、Recover（）。

（2）函数Validate\_ans（）调用了函数RemoveSpecialClause（）、isSetBlank（）、Analyze\_cnf（）。

（3）函数Recover（）调用了函数is\_lit\_in\_clause（）。

（4）函数RemoveSpecialClause（）调用了函数RemoveClause（）。

（5）函数Dele\_special调用了函数Special\_lit\_place（）。

new\_sudoku.c中:

（1）函数ChangeNum（）调用了函数isNumberChosen（）。

（2）函数MakeRandomSudoku（）调用了函数ChangeNum（）。

（3）函数sudoku\_to\_SAT（）调用了函数Can\_exist\_in\_line（）、Can\_exist\_in\_Col（）、Can\_exist\_in\_square（）。

**4.2 系统测试**

首先叙述一下常用的软件测试方法，在选择几个主要的功能模块（自行掌握数量，关键要体现你的水平的一些模块）描述测试过程，（1）先明确模块的功能、设计目标等。（2）分析、叙述如何选取测试数据，要求有完整的测试大纲。（3）运行结果（这时可用截图）。（4）分析运行结果、确认程序满足该模块的设计目标。

**4.2.1 测试方法**

使用VS Code进行编译，然后运行exe文件，即可开始测试。

**4.2.2 测试过程**

（1）SAT求解模块：

该模块的功能是对cnf文件给出的一个子句集进行求解，并且可以选择将解保存至res文件中。设计的目标是能让其尽量快地求出SAT问题的解。

对于测试数据，可以根据变元的数目将其分为小型、中型、大型算例。小型算例的变元数在100个之内，中型算例变元数介于200-500个； 大型算例变元数600个以上。

测试的实例：

打开SAT.c，如图4-2：

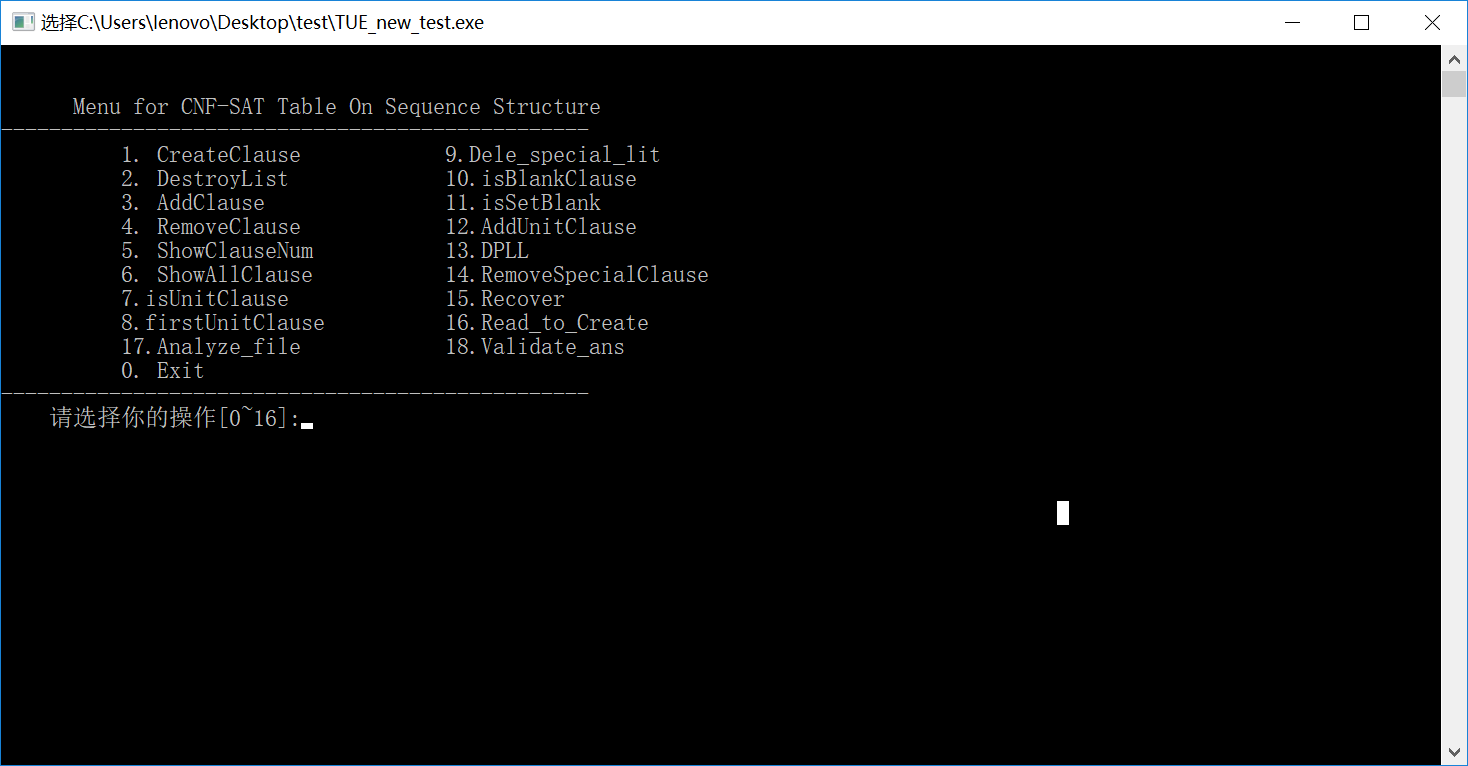


图4-2

选择16号功能，输入要读取的cnf文件名，如图4-3：

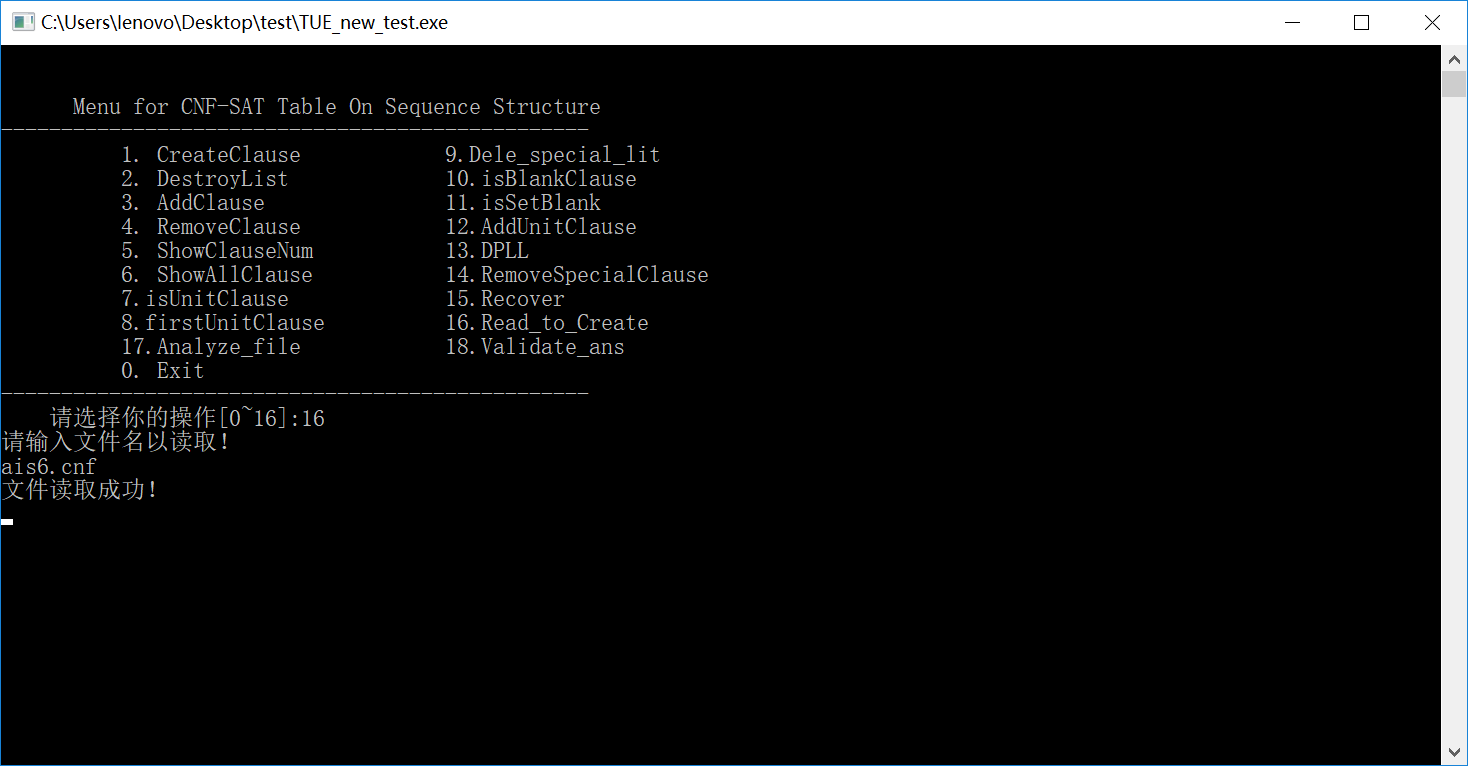


图4-3

选择13号功能，会输出解，如图4-4：

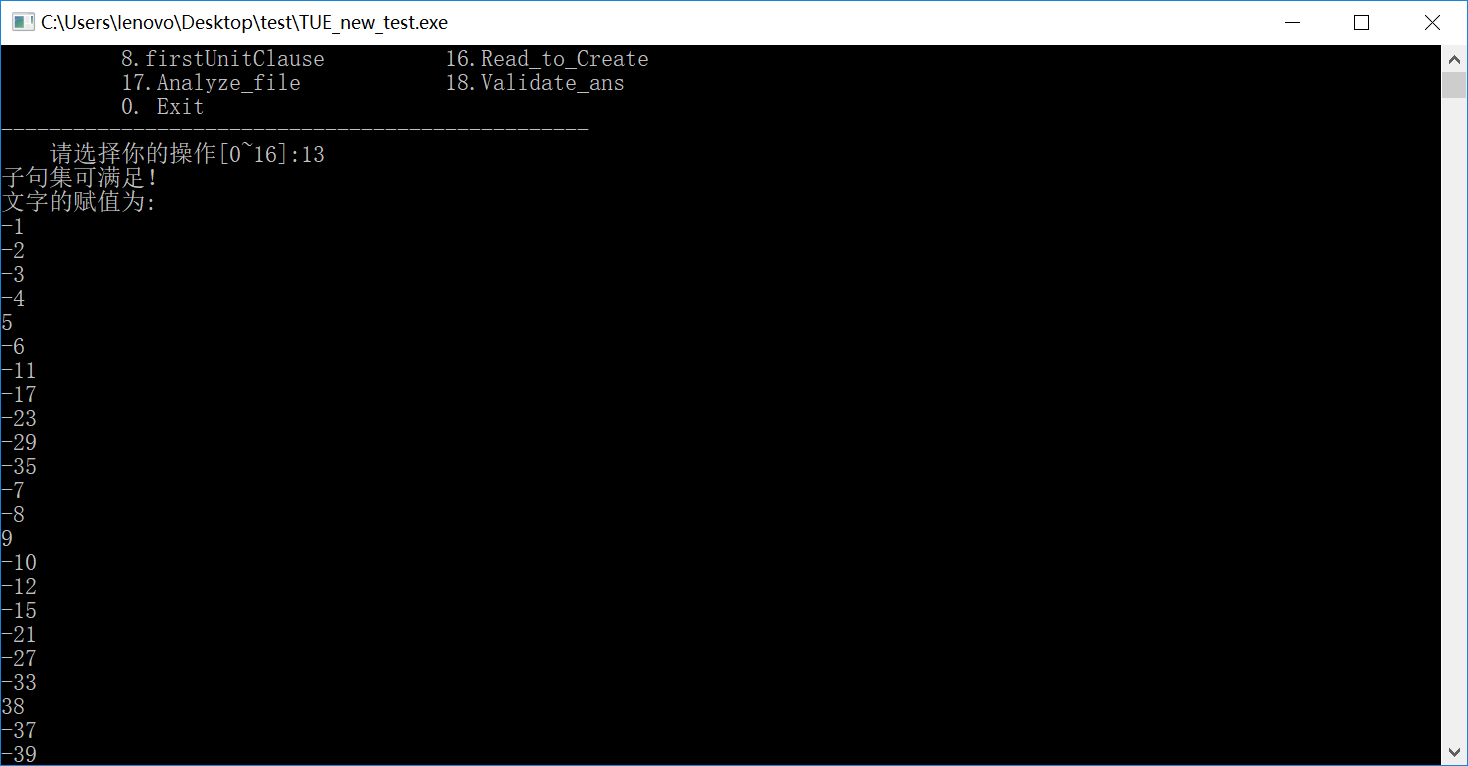


图4-4

可以选择将解保存到res文件中，如图4-5：

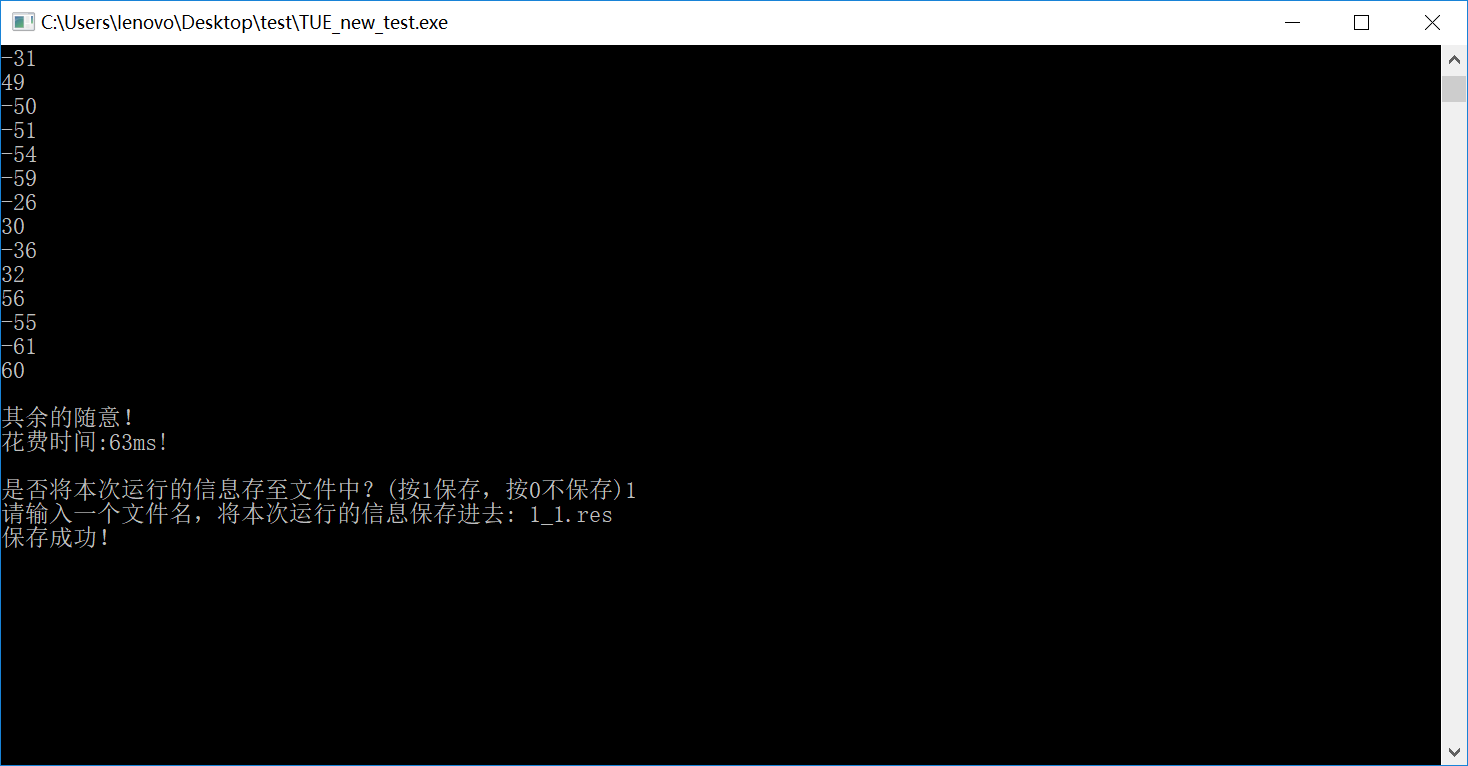


图4-5

以下为测试的结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 算例名 | 变元数 | 子句数目 | 未优化运行时间 | 已优化运行时间 |
| sat-20.cnf（下面为基准算例） | 20 | 91 |  |  |
| unsat-5cnf-30.cnf | 30 | 420 |  |  |
| ais10.cnf | 181 | 3151 |  |  |
| sud00009.cnf | 303 | 2851 |  |  |
| problem2-50.cnf（下面为小型算例） | 50 | 80 |  |  |
| ais6.cnf | 61 | 581 |  |  |
| problem1-20.cnf | 20 | 91 |  |  |
| tst\_v25\_c100.cnf | 25 | 100 |  |  |
| problem6-50.cnf | 50 | 100 |  |  |
| problem9-100.cnf | 100 | 200 |  |  |
| problem8-50.cnf | 50 | 300 |  |  |
| problem3-100.cnf | 100 | 340 |  |  |
| problem11-100.cnf | 100 | 600 |  |  |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 20 | 1532 |  |  |
| tst\_v200\_c210.cnf（中型算例） | 200 | 210 |  |  |
| sw100-1.cnf | 500 | 3100 |  |  |
| bart17.shuffled-231.cnf | 231 | 1166 |  |  |
| problem12-200.cnf | 200 | 1200 |  |  |
| sud00082.cnf | 224 | 1762 |  |  |
| sud00012.cnf | 232 | 1901 |  |  |
| sud00861.cnf | 297 | 2721 |  |  |
| sud00001.cnf | 301 | 2780 |  |  |
| sud00079.cnf | 301 | 2810 |  |  |
| sud00009.cnf | 303 | 2851 |  |  |
| ec-iso-ukn009.shuffled-as.sat05-3632-1584.cnf（大型算例） | 1584 | 16587 | 跑不出来… |  |
| tst\_v10\_c100.cnf（不满足算例） | 10 | 100 |  |  |
| u-problem7-50.cnf | 50 | 100 |  |  |
| u-problem10-100.cnf | 100 | 200 |  |  |
| php-010-008.shuffled-as.sat05-1171.cnf | 80 | 370 |  |  |
| u-5cnf\_3500\_3500\_30f1.shuffled-30.cnf | 30 | 420 |  |  |

（2）数独求解模块：

打开new\_sudoku.c，如图4-6:

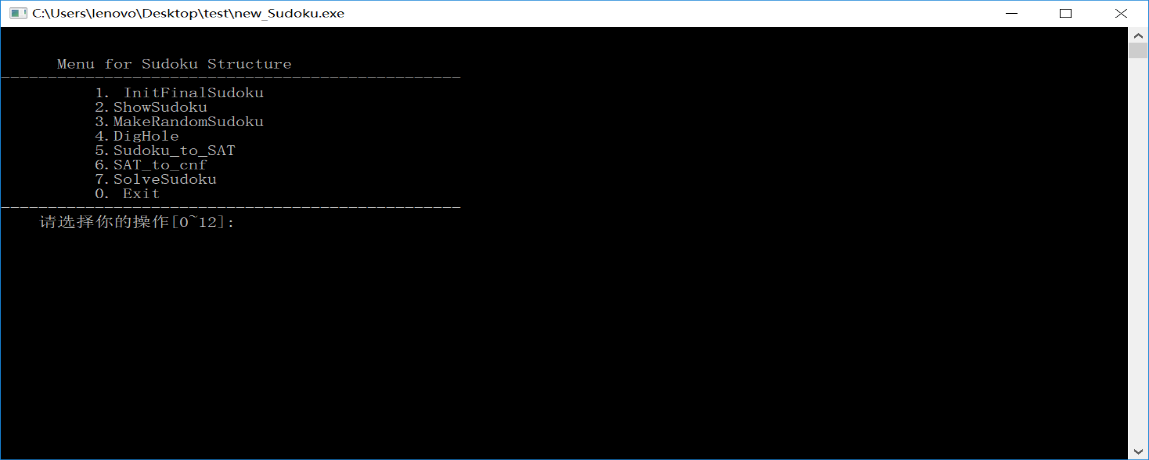


图4-6

选择1号功能，读入一个已知的数独终局，如图4-7：

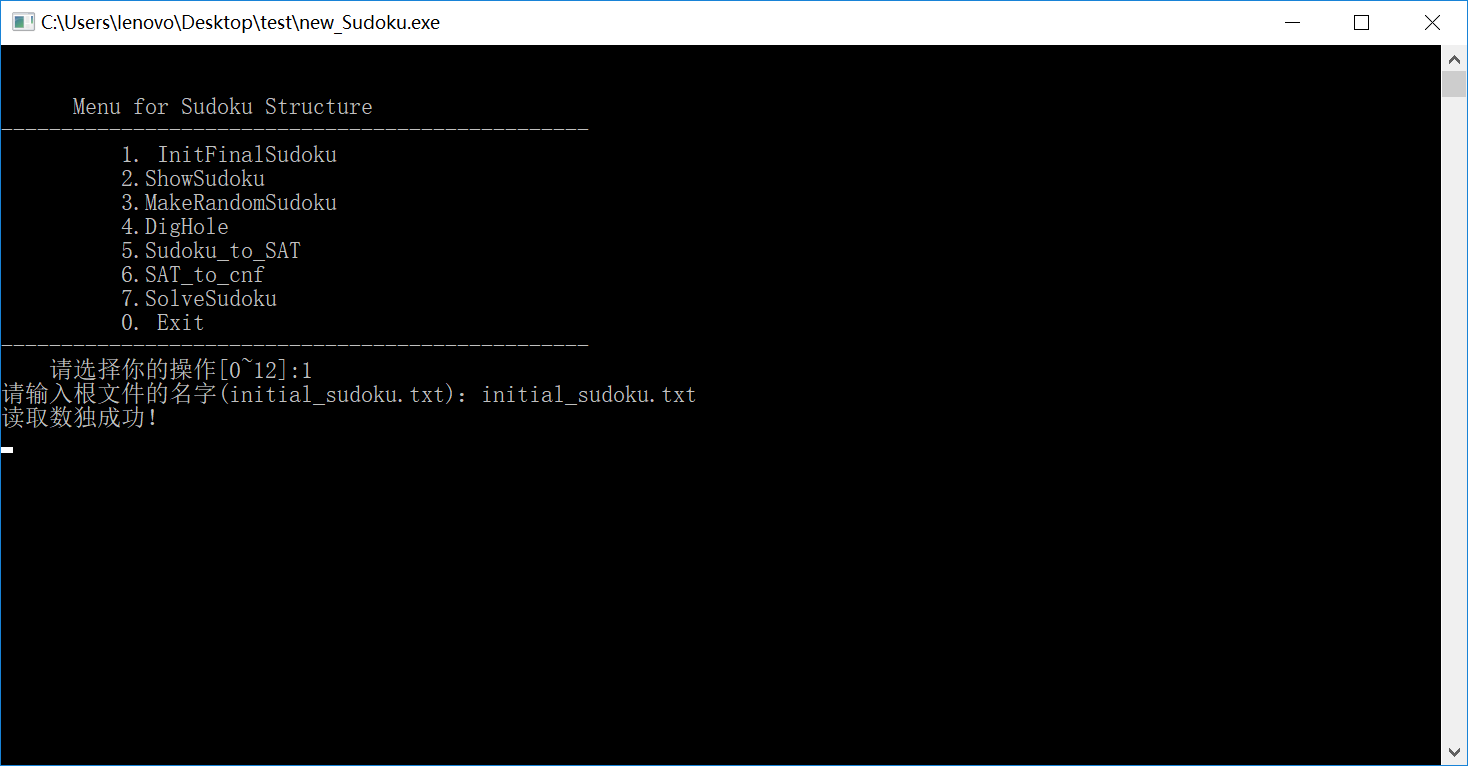


图4-7

选择3号功能，生成随机数独，如图4-8、4-9：

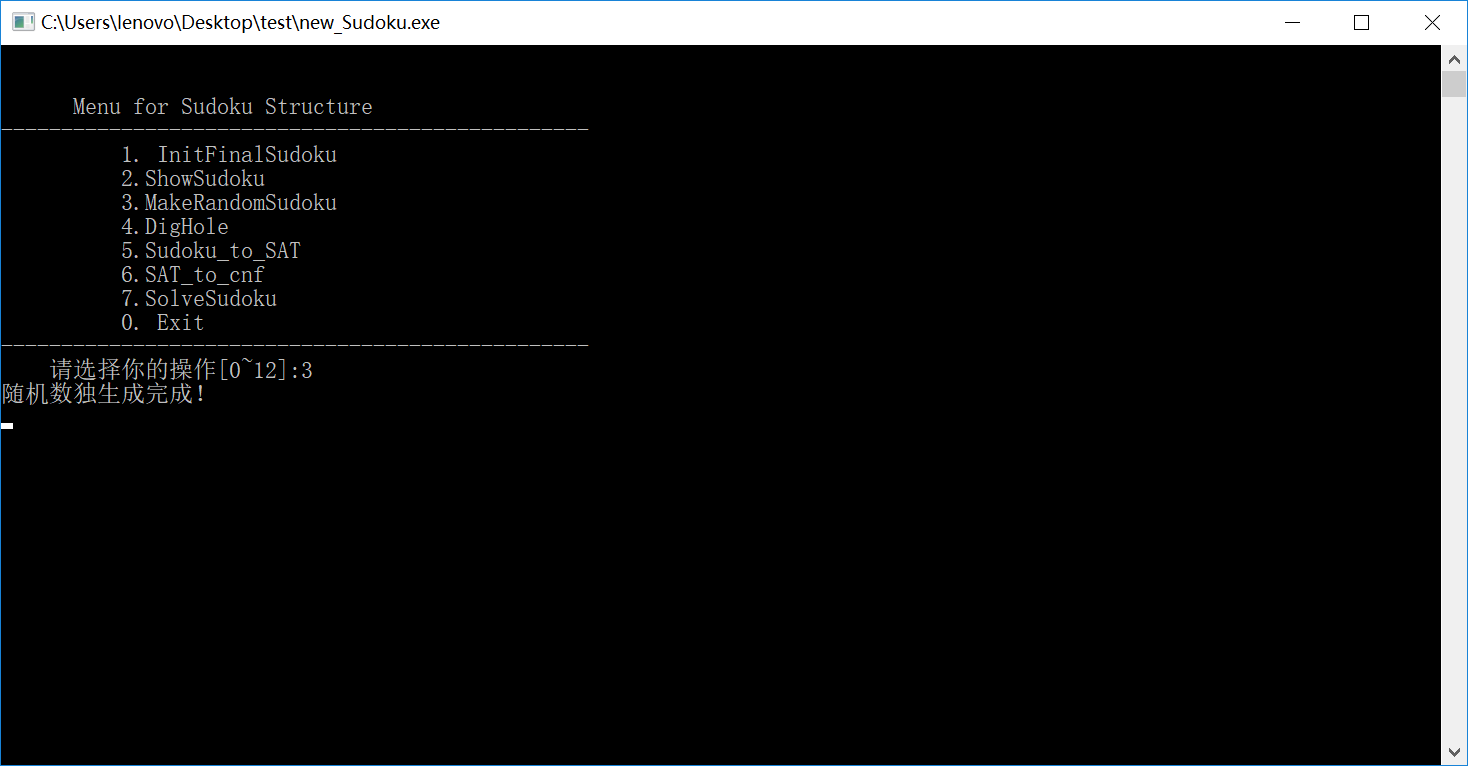


图4-8

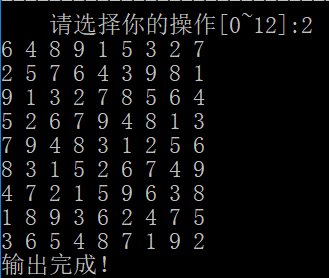


图4-9

选择4号功能，挖洞，如图4-10、4-11：

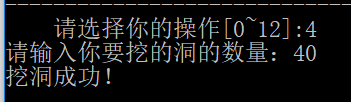


图4-10

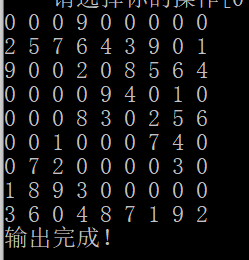


图4-11

选择5号功能，将SAT的信息转过去，如图4-12：

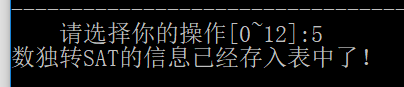


图4-12

选择6号功能，转化为cnf文件，如图4-13：

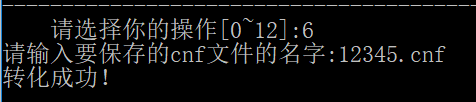


图4-13

打开SAT.c，对cnf文件进行求解如图4-14、4-15、4-16：

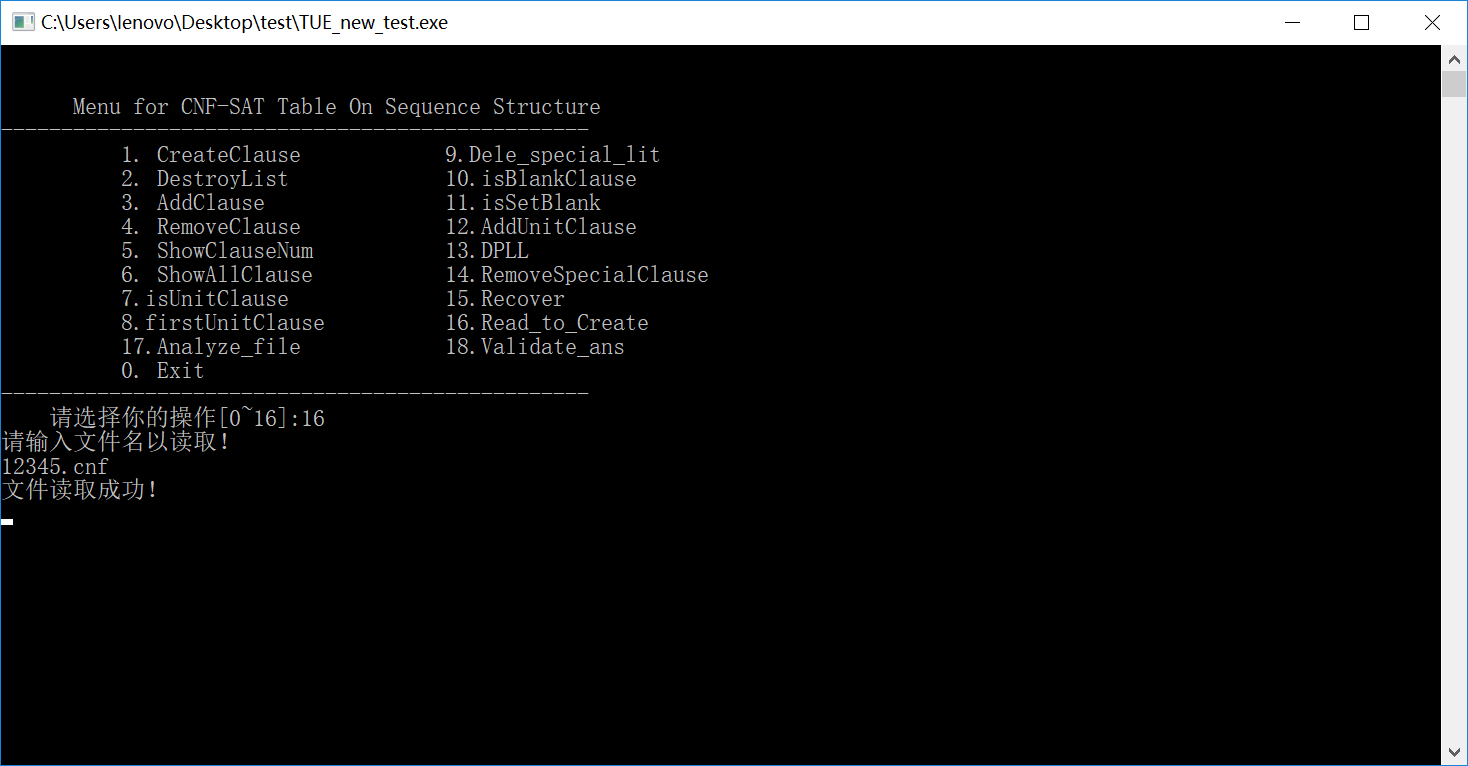


图4-14

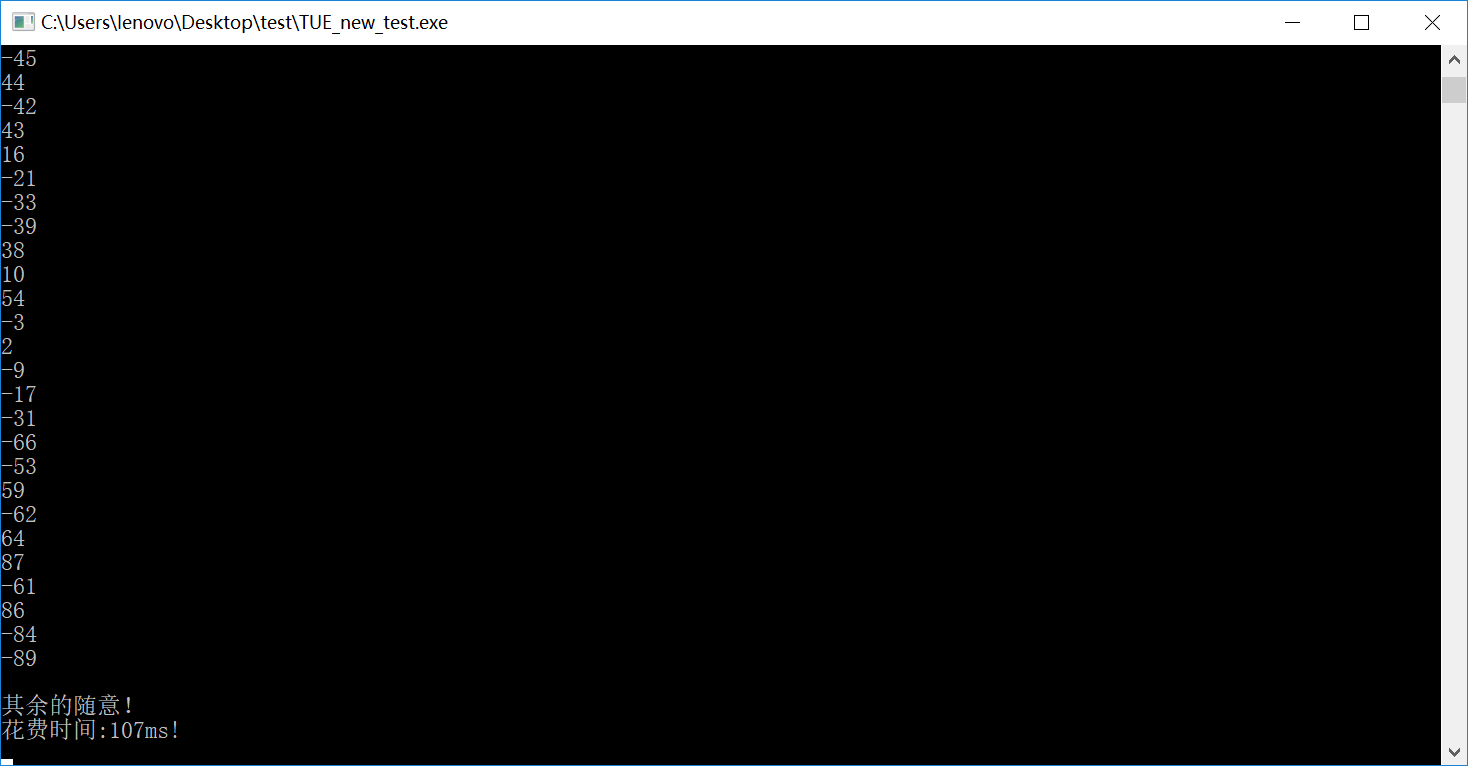


图4-15

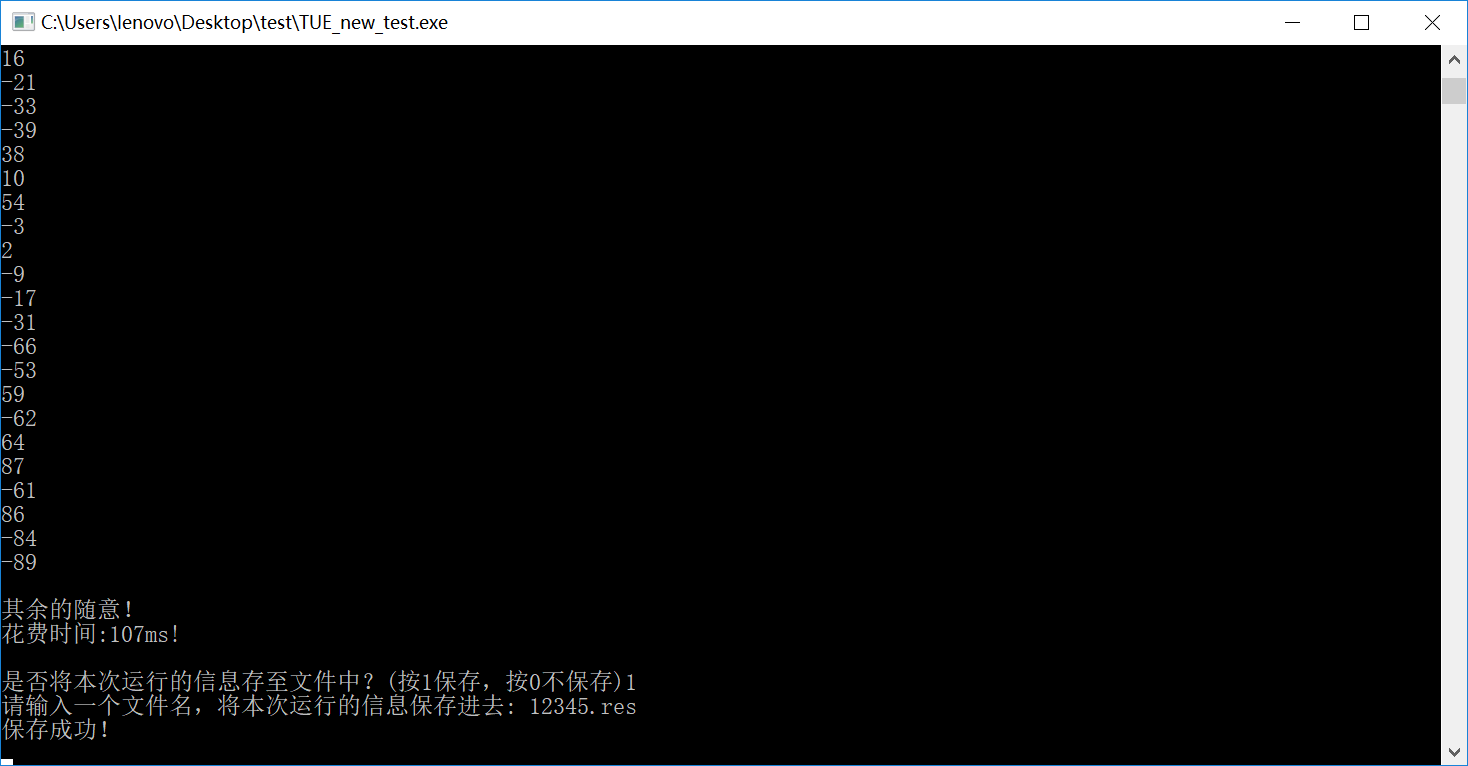


图4-16

在new\_sudoku.c中读取res文件，给数独填空，如图4-17、4-18：

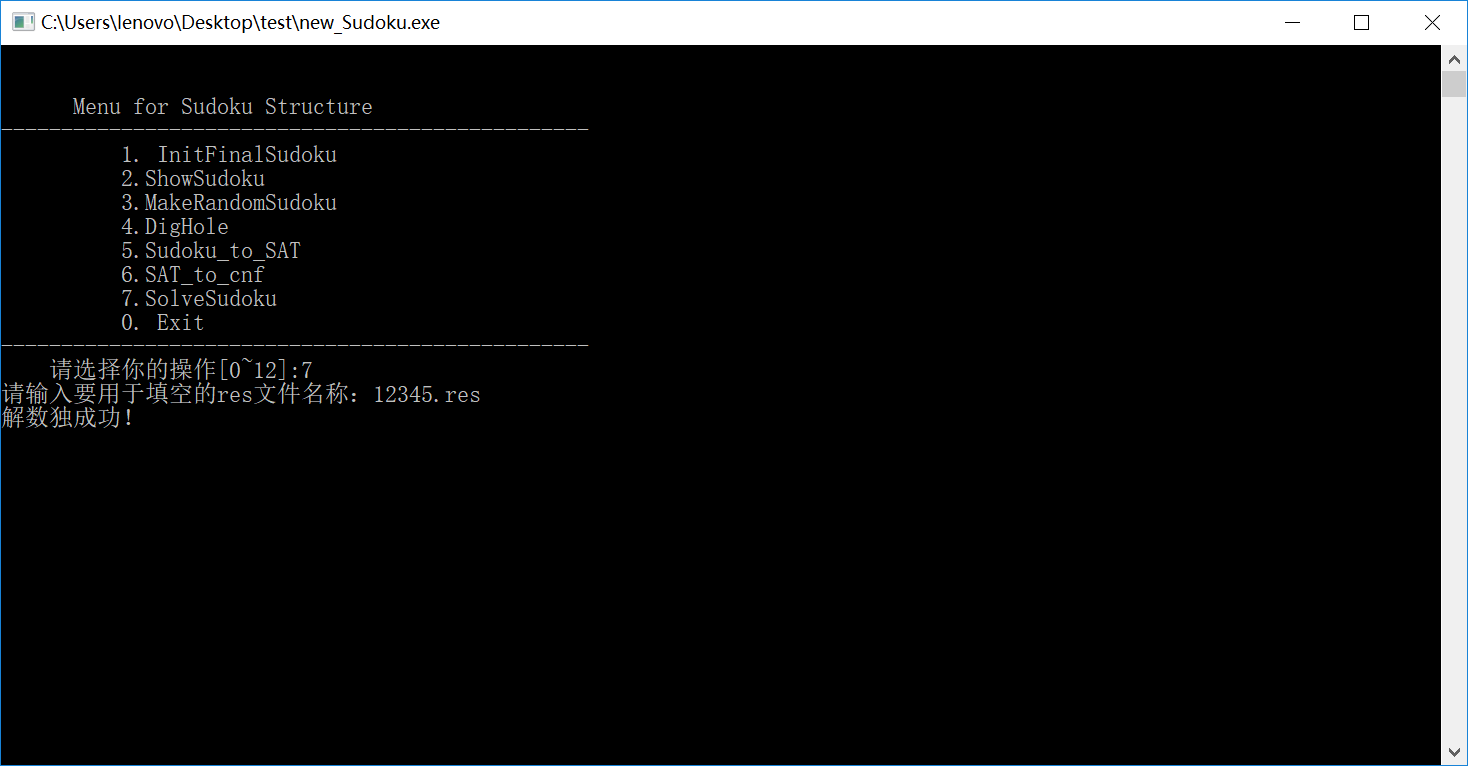


图4-17

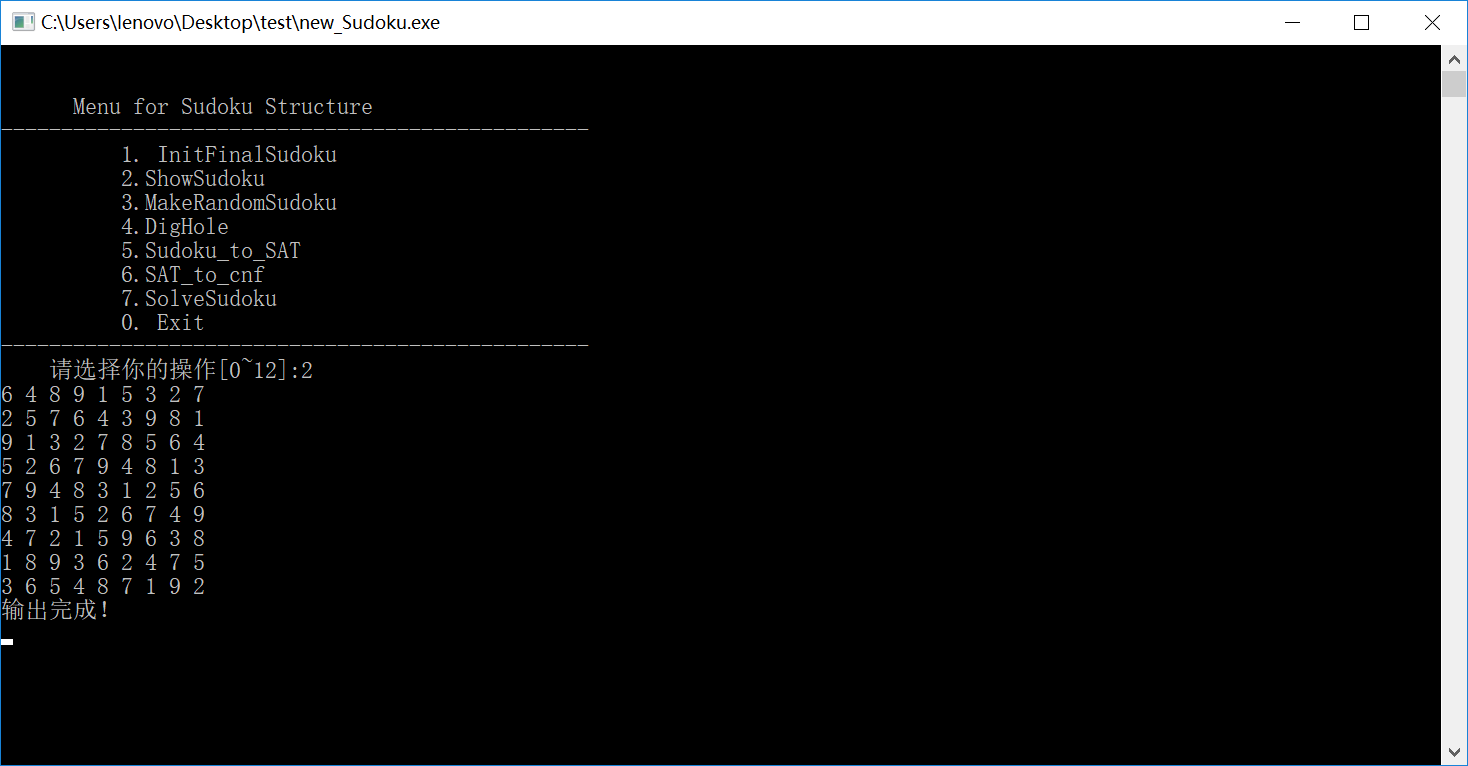


图4-18

--------章与章之间插入分页符----------

**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）对初步的数据结构的建立。本系统的数据结构是一个二维的链表，以及一个栈。二维的链表用于存放子句集，子句的长度等等重要信息；栈则用于存放假设成立的文字。

（2）初级函数的构建：写一些简单的函数，比如显示整个子句集，添加子句、删除子句、判断子句集是否为空等等，为后面构建DPLL函数做好基础的准备。

（3）DPLL函数的设计：本系统的DPLL算法用的是非递归的算法，配上一个栈来进行的。之前设计过递归的算法，但是在回溯方面遇到了困难，再加上递归算法的时间复杂度会很高，因此后期改成了利用栈的非递归算法。

（4）数独的设计：在随机数独的生成方面，我采用的是先读取已知数独终局，再对其进行变换得到数独终局，最后挖空的方法。因为这样一个终局就有9!共三十多万种数独终局，相当于是随机生成。然后再根据填数独的规则，转化为对应的序号构成的子句，这样就转化为了SAT问题，然后利用之前的SAT求解器，就能够解数独问题了。

**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）进一步的优化：我本次的优化主要是文字选择策略方面，由选择第一个子句的第一个文字改成了选择最短子句的第一个文字，以及DPLL中一些不必要的循环与判断，这样可以优化大部分程序。但是，这种优化方法仍有不足，还有空间可以继续优化，从而大大提高速度

（2）数独的挖洞不保证有唯一解：本次的挖洞是随机挖洞，但在挖洞的过程中每次没有检查本次的挖洞是否能保证有唯一解，这样会大大增加生成数独的时间。但后来老师降低了本次的要求，因此不需要去保证挖洞有唯一解。

--------章与章之间插入分页符----------

**6 体会**

本次课程设计，我花了大量的假期时间去构思，然后实现，以及最后的优化，可以说收获是很丰富的，主要体现在一下几个方面：

（1）在代码debug的方面：可以说这次课设我花了很大的精力去debug，大大提高了我debug的水平。我最印象深刻的一个如下：由于我删除子句是将其隐藏，没有free掉，所以在恢复的时候，可能恢复含v的子句，会将之前因为u而删掉的子句（该子句也含v）恢复，但是u这个假设是在v之前做出的，这样就会导致已经假设u成立了，但是仍然含有带u的子句。为了解决这个问题，我给每个子句的HeadNode设置了一个why，用于记录该子句是为什么被删除，恢复的时候要看是不是要恢复why所造成的影响。我同时也意识到，代码正是在不断的发现问题与解决问题中完善的。

（2）在优化上面，我发现：几乎没有能够优化所有算例的优化方法。有些算例，在代码优化之后反而会跑得更慢。我去查找资料，发现各种特点的算例，有着更适合这些特点的优化方法。因此，不一定优化过的算法就一定比没有优化过的算法更快。

（3）在多个模块的构建上：这次项目让我学会了如何从小的函数开始构建大的模块，从而最后实现多个模块的协作来完成最终的功能。小函数的构建看似繁琐，实际上一些实用的小函数能够大大简化后期重复的代码。而分模块来进行，可以让自己的代码结构更加清晰，更加容易理解，这在今后是非常重要的。

以上就是我本次课程设计的体会。

**附录**

SAT.c的代码：

#include <stdio.h>

#include<time.h>

#include<string.h>

#define OK 1

#define FALSE 0

#define TRUE 1

#define ERROR 0

#define INFEASTABLE -1

#define YES 1

#define NO 0

typedef int status;

typedef int ELemType;//变元的命名

//一个变元放在一个节点中

typedef struct Node{

ELemType data;//节点包含数据域

struct Node\* next;//指针域

status is\_node\_hidden;//状态：节点是否被删除

}Node;

typedef struct HeadNode{

int data;//数据域存放子句的长度

int all\_data;//该子句初始状态的长度

struct HeadNode\* down;//下一个子句

struct Node\* next;//指向该子句的第一个节点

status is\_headnode\_hidden;//状态：该子句是否被删除

ELemType why;//如果子句被删除，则记录第一次被删除是因为哪个文字。若未被删除则为0

}HeadNode;

int ziju\_num=0;//当前子句的个数

int all\_ziju\_num=0;//电脑中的子句总数，由于删除只是隐藏，因此不会把子句真正删掉

int Stack[10000][2];//栈第一列用于存放满足的文字，第二列存放changed的状态

int v\_num;//变元数

clock\_t start, stop;//用于统计DPLL所用时间

long duration;//duration是DPLL花费时间

status CreateClause(HeadNode\*\* root, HeadNode\*\* tail);//创造子句集

status DestroyClause(HeadNode\*\* root, HeadNode\*\* tail);//删除子句集(这个是真的删除，会free)

status AddClause(HeadNode\*\* root, HeadNode\*\* tail);//添加子句

status RemoveClause(HeadNode\*\* root, HeadNode\*\* tail, int n);//删除第n个子句

void Show\_ziju\_num(void);//显示子句个数

void ShowAllClause(HeadNode \*root);//展示所有的子句

status isUnitClause(HeadNode \*root);//判断子句集中是否存在单子句

int firstUnitClause(HeadNode \*root);//返回子句集中第一个单子句的序号(包含已隐藏的)

int Special\_lit\_place(HeadNode \*p, int lit);//找到当前子句中第一个为lit的文字的位置(包含隐藏的节点)，没有则返回0

status Dele\_special\_lit(HeadNode\*\*root,int lit);//在所有子句中删除指定的文字

status isBlankClause(HeadNode \*root);//判断子句集中是否有空子句

int firstBlankClause(HeadNode\* root);//返回子句集中第一个空子句的序号(包含被隐藏的子句)

status isSetBlank(void);//判断子句集是否为空

status DPLL(HeadNode \*root, HeadNode\* tail);//递归的DPLL算法

status RemoveSpecialClause(HeadNode \*\*root, HeadNode\*\* tail, int L);//删除包含文字L的子句

status Recover(HeadNode \*\*root,ELemType v);//恢复包含v的子句，以及包含-v的节点

status is\_lit\_in\_clause(HeadNode \*p, ELemType lit);//判断lit是否在子句中(被隐藏了也算在里面！！！！！)

status Read\_to\_Create(FILE \*fp, char \*filename, HeadNode \*\*root, HeadNode \*\*tail);//读取cnf文件并且创建子句集

status SaveFile(FILE \*fp, char \*filename, int duration, int ans);//将DPLL解析的信息存入文件中

void Analyze\_cnf(HeadNode\* root);//对cnf文件进行解析

status Validate\_ans(HeadNode \*root, HeadNode \*tail);//用栈中的变元验证cnf文件

int main(void)

{

int s=0;

FILE \*fp =NULL;

char filename[40];

//将栈初始化

for(; s<100;s++)

{

Stack[s][0]=0;

Stack[s][1]=NO;

}

HeadNode\* root = NULL;

HeadNode\* tail =NULL;

HeadNode\*\* root\_p=&root;

HeadNode\*\* tail\_p=&tail;

int op=1;

int choice;//用于选择是否打印出文件

while(op)

{

system("cls");  printf("\n\n");

     printf(" Menu for CNF-SAT Table On Sequence Structure \n");

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

     printf("      1. CreateClause 9.Dele\_special\_lit\n");

     printf("      2. DestroyList 10.isBlankClause \n");

printf(" 3. AddClause 11.isSetBlank\n ");

printf(" 4. RemoveClause 12.AddUnitClause \n");

printf(" 5. ShowClauseNum 13.DPLL \n");

printf(" 6. ShowAllClause 14.RemoveSpecialClause\n");

printf(" 7.isUnitClause 15.Recover\n");

printf(" 8.firstUnitClause 16.Read\_to\_Create\n");

     printf("      17.Analyze\_file 18.Validate\_ans\n");

printf(" 0. Exit \n");

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

     printf(" 请选择你的操作[0~16]:");

     scanf("%d",&op);

switch(op){

case 1:

{

printf("请输入子句的个数");

scanf("%d", &ziju\_num);

all\_ziju\_num=ziju\_num;//给all\_ziju\_num赋初值

if(CreateClause(root\_p,tail\_p)==OK)

printf("子句集创建成功！\n");

else if(root!=NULL)

printf("创建失败！子句集已经存在了！\n");

getchar();getchar();

break;

}

case 2:

{

if(DestroyClause(root\_p, tail\_p)==OK)

printf("子句集销毁成功！\n");

else if(!root)

printf("销毁失败！子句集尚未创建！\n");

getchar();getchar();

break;

}

case 3:

{

if(AddClause(root\_p,tail\_p)==OK)

printf("添加子句成功!\n");

else

printf("添加子句失败!\n");

getchar();getchar();

break;

}

case 4:

{

int n;

printf("请输入要删除的子句的序号:");

scanf("%d",&n);

if(!root)

printf("删除失败!子句集尚未初始化！\n");

else if(RemoveClause(root\_p,tail\_p,n)==OK)

printf("删除子句成功！\n");

else

printf("删除失败！\n");

getchar();getchar();

break;

}

case 5:

{

Show\_ziju\_num();

getchar();getchar();

break;

}

case 6:

{

if(!root)

{

printf("子句集尚未初始化！\n");

getchar();getchar();

}

else

{

ShowAllClause(\*root\_p);

getchar();getchar();

}

break;

}

case 7:

{

if(!root) printf("子句集尚未初始化！\n");

else if(isUnitClause(\*root\_p)==TRUE)

printf("存在单子句！\n");

else

printf("不存在单子句！\n");

getchar();getchar();

break;

}

case 8:

{

if(isUnitClause(\*root\_p)==TRUE)

printf("第一个单子句的序号是%d！\n", firstUnitClause(\*root\_p));

else

printf("不存在单子句！\n");

getchar();getchar();

break;

}

case 9:

{

int lit;

printf("请输入要删除的文字:");

scanf("%d", &lit);

if(Dele\_special\_lit(root\_p, lit)==OK)

printf("删除成功！\n");

else

printf("删除失败！\n");

getchar();getchar();

break;

}

case 10:

{

if(isBlankClause(root)==TRUE)

printf("存在空子句！\n");

else

printf("不存在空子句！\n");

getchar();getchar();

break;

}

case 11:

{

if(isSetBlank()==TRUE)

printf("子句集为空！\n");

else

printf("子句集非空！\n");

getchar();getchar();

break;

}

case 13:

{

int ans;

start = clock();//开始计时

ans=DPLL(root, tail);

if(ans==TRUE)

{

printf("子句集可满足！\n");

printf("文字的赋值为:\n");

int i=0;

while(Stack[i][0]!=0)

{

printf("%d\n",Stack[i][0]);

i++;

}

printf("\n其余的随意！\n");

}

else

printf("子句集不可满足！\n");

stop=clock();

duration = ((long)(stop - start));

printf("花费时间:%ldms!\n",duration);

getchar();getchar();

printf("是否将本次运行的信息存至文件中？(按1保存，按0不保存)");

scanf("%d", &choice);

if(choice==1)

{

printf("请输入一个文件名，将本次运行的信息保存进去: ");

scanf("%s",filename);

if(SaveFile(fp,filename,duration,ans)==TRUE)

printf("保存成功！\n");

else

printf("保存失败！\n");

}

else

printf("你说不存就不存吧\n");

getchar();getchar();

break;

}

case 14:{

int L;

printf("请输入文字L，将删除含有L的子句：");

scanf("%d",&L);

RemoveSpecialClause(root\_p,tail\_p,L);

printf("删除成功！\n");

getchar();getchar();

break;

}

case 15:{

ELemType v;

printf("请输入文字v，用于恢复之前处理v带来的影响\n");

scanf("%d",&v);

if(Recover(root\_p,v)==OK)

printf("恢复成功！\n");

getchar();getchar();

break;

}

case 16:{

printf("请输入文件名以读取！\n");

scanf("%s",filename);//将一个指定的链表存到filename中

if(Read\_to\_Create(fp,filename,root\_p, tail\_p)==OK)

printf("文件读取成功！\n");

else

printf("文件读取失败！\n");

getchar();getchar();

break;

}

case 17:

{

printf("文件解析如下：\n");

Analyze\_cnf(root);

printf("解析完成!\n");

getchar();getchar();

break;

}

case 18:

{

if(Validate\_ans(root, tail)==OK)

printf("\n验证结果为满足！\n");

else

printf("\n验证结果为不满足！\n");

getchar();getchar();

break;

}

}//end of switch

}//end of while

return 0;

}

status CreateClause(HeadNode\*\* root, HeadNode\*\* tail){

if(\*root != NULL)

return INFEASTABLE;

else

{

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

(\*root)->next=NULL;

(\*root)->is\_headnode\_hidden=NO;

(\*root)->why=0;

HeadNode\* p=\*root;

Node \*q=NULL;

int i;//用于初始化多个链表

int number;//用于读取输入

//创建头结点

for( i=1 ;i<ziju\_num;i++)

{

p->data=0;

p->all\_data=0;

p->down=(HeadNode\*)malloc(sizeof(HeadNode));

p=p->down;

p->next=NULL;

p->is\_headnode\_hidden=NO;

p->why=0;

}

p->data=0;//最后一个子句初始长度为0

p->all\_data=0;

\*tail=p;

p->down = NULL;

//输入每个链表的信息

p=\*root;

for(i=1;i<=ziju\_num;i++)

{

if(scanf("%d", &number)&&number==0)

{

p->next=NULL;

p=p->down;

}

else

{

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

q->data=number;

q->is\_node\_hidden=NO;

p->data++;

p->all\_data++;

p->next=q;

while(scanf("%d",&number)&&(number!=0))

{

q->next=(Node\*)malloc(sizeof(Node));

q=q->next;

q->is\_node\_hidden=NO;

q->data=number;

p->data++;

p->all\_data++;

}

q->next=NULL;

p=p->down;

}

}

return OK;

}

}

status DestroyClause(HeadNode\*\* root, HeadNode\*\* tail){

if(!(\*root)) return INFEASTABLE;

else

{

\*root = NULL;

free(\*root);

\*tail=NULL;

free(\*tail);

ziju\_num=0;

all\_ziju\_num=0;

return OK;

}

}

status AddClause(HeadNode\*\* root, HeadNode\*\*tail){

Node\* q=NULL;

HeadNode\* p=NULL;

int number;

ziju\_num++;

all\_ziju\_num++;

printf("请输入要添加的子句:\n");

if(!(\*root))//还没有初始化

{

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

\*tail=\*root;

(\*tail)->data=0;

(\*tail)->all\_data=0;

(\*tail)->is\_headnode\_hidden=NO;

int number;//用于读取输入

if(scanf("%d", &number)&&number==0)

{

(\*tail)->next=NULL;

(\*tail)->down=NULL;

}

else

{

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

q->data=number;

q->is\_node\_hidden=NO;

(\*tail)->data++;

(\*tail)->all\_data++;

(\*tail)->next=q;

while(scanf("%d",&number)&&(number!=0))

{

q->next=(Node\*)malloc(sizeof(Node));

q=q->next;

q->data=number;

q->is\_node\_hidden=NO;

(\*tail)->data++;

(\*tail)->all\_data++;

}

q->next=NULL;

(\*tail)->down=NULL;

}

return OK;

}

else//已经初始化

{

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

(\*tail)->down=p;

\*tail=p;

(\*tail)->data=0;

(\*tail)->all\_data=0;

(\*tail)->is\_headnode\_hidden=NO;

if(scanf("%d", &number)&&number==0)

{

(\*tail)->next=NULL;

(\*tail)->down=NULL;

}

else

{

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

q->data=number;

q->is\_node\_hidden=NO;

(\*tail)->data++;

(\*tail)->next=q;

(\*tail)->all\_data++;

while(scanf("%d",&number)&&(number!=0))

{

q->next=(Node\*)malloc(sizeof(Node));

q=q->next;

q->data=number;

q->is\_node\_hidden=NO;

(\*tail)->data++;

(\*tail)->all\_data++;

}

q->next=NULL;

(\*tail)->down=NULL;

}

return OK;

}

}

status RemoveClause(HeadNode\*\* root, HeadNode\*\* tail, int n){

int i;//n是要删除的子句的序号，i用于找到那个子句

HeadNode\* p=\*root;//用于找到那个子句的头指针

HeadNode\* q=\*root;

if(n>all\_ziju\_num)

{

printf("该子句不存在！\n");

return ERROR;

}

else

{

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

{

p=p->down;//p是要删除的子句

}

p->is\_headnode\_hidden=YES;

ziju\_num--;

return OK;

}

}

void Show\_ziju\_num(void){

printf("子句的数目为%d",ziju\_num);

}

void ShowAllClause(HeadNode \*root){

int i=1,j=1;

HeadNode\* p=root;

Node\* q=p->next;//不能写NULL->next!!!!!!!!!!!!

printf("总共有%d个子句\n",ziju\_num);

if(ziju\_num>0)

for(;i<=all\_ziju\_num;i++)

{

if(p->is\_headnode\_hidden==NO)

{

printf("第%d个子句: ",j);

while(q!=NULL)

{

if(q->is\_node\_hidden==NO)

printf("%d ",q->data);

q=q->next;

}

printf("共%d个变元,",p->data);

printf("位置序号:%d\n",i);

j++;

}

p=p->down;

if(p!=NULL)

q=p->next;//这里不能直接写q=p->next，因为没有说明NULL->next=NULL

}

}

status isUnitClause(HeadNode \*root){

HeadNode \*p=root;

int i=1;

for(; i<=all\_ziju\_num; i++)

{

if((p->data==1)&&(p->is\_headnode\_hidden==NO))

return TRUE;

p=p->down;

}

return FALSE;

}

int firstUnitClause(HeadNode \*root){

HeadNode \*p=root;

int i=1;

for(; i<=all\_ziju\_num; i++)

{

if((p->data==1)&&(p->is\_headnode\_hidden==NO))

return i;

p=p->down;

}

}

int Special\_lit\_place(HeadNode \*p, int lit){

Node \*q=p->next;

int i=1;

while(q!=NULL)

{

if((q->data==lit)&&(q->is\_node\_hidden==NO))

return i;

i++;

q=q->next;

}

return 0;

}

status Dele\_special\_lit(HeadNode \*\*root, int lit){

HeadNode \*p=\*root;

Node \*q=p->next, \*r=p->next;//这里也是不能写成NULL->next=NULL

int i=1, j=1 ,place;

for(; i<=all\_ziju\_num; i++)//这里要写成all!!!!!!!!!!!

{

if(p->is\_headnode\_hidden==NO)

{

while(Special\_lit\_place(p,lit)!=0)

{

place=Special\_lit\_place(p,lit);

q=p->next;

for(j=1;j<place;j++)

{

q=q->next;

}

q->is\_node\_hidden=YES;

p->data--;

}//end of while

}

p=p->down;

}

return OK;

}

status isBlankClause(HeadNode \*root){

HeadNode \*p=root;

int i=1;

for(; i<=all\_ziju\_num; i++)

{

if((p->data==0)&&(p->is\_headnode\_hidden==NO))

return TRUE;

p=p->down;

}

return FALSE;

}

status isSetBlank(void){

if(ziju\_num==0)

return TRUE;

else

return FALSE;

}

int firstBlankClause(HeadNode\* root){

HeadNode \*p=root;

int i=1;

for(; i<=all\_ziju\_num; i++)

{

if(p->data==0)

return i;

p=p->down;

}

}

status RemoveSpecialClause(HeadNode \*\*root, HeadNode\*\* tail, int L){

HeadNode \*p=\*root;

Node \*q=p->next;

int i=1,n=ziju\_num;

while(p!=NULL)

{

if(p->is\_headnode\_hidden==NO)

{

while(q!=NULL)

{

if((q->data==L)&&(q->is\_node\_hidden==NO))

{

RemoveClause(root,tail,i);

p->why=L;

break;

}

q=q->next;

}

p=p->down;

if(p!=NULL)

q=p->next;

}

else

{

p=p->down;

if(p!=NULL)

q=p->next;

}

i++;

}

return OK;

}

status is\_lit\_in\_clause(HeadNode \*p, ELemType lit)

{

Node \*q=p->next;

while(q!=NULL)

{

if(q->data==lit)

return TRUE;

q=q->next;

}

return FALSE;

}

status Recover(HeadNode \*\*root,ELemType v){

HeadNode \*p=\*root;

Node \*q;

while(p!=NULL)

{

//先恢复删除的子句

if((is\_lit\_in\_clause(p,v)==TRUE)&&(p->is\_headnode\_hidden==YES)&&(p->why==v))

{

ziju\_num++;

p->is\_headnode\_hidden=NO;

}

p=p->down;

}

//再恢复删除的文字

p=\*root;

while(p!=NULL)

{

if((p->is\_headnode\_hidden==NO)&&(is\_lit\_in\_clause(p,-v)==TRUE))

{

q=p->next;

while(q!=NULL)

{

if((q->data==-v)&&(q->is\_node\_hidden==YES))

{

q->is\_node\_hidden=NO;

p->data++;

}

q=q->next;

}

}

p=p->down;

}

return OK;

}

status DPLL(HeadNode \*root, HeadNode \*tail){

ELemType v;

HeadNode \*\*root\_p=&root, \*\*tail\_p=&root;

HeadNode \*p=root;

Node \*q;

int top=-1;

int n,i,shortest\_clause=0,number=1,length=100;

if(isBlankClause(root)==TRUE)

return FALSE;

while(isSetBlank()==NO)

{

number=1,length=100;

p=root;

while(p!=NULL)//p是最短的一个子句

{

if(p->is\_headnode\_hidden==NO){

if((p->data)<length){

length=p->data;

if((length==1)&&(length==2))

break;

shortest\_clause=number;

}

}

p=p->down;

number++;

}

p=root;

for(i=1;i<shortest\_clause;i++){

p=p->down;

}

q=p->next;

while((q!=NULL)&&(q->is\_node\_hidden==YES))//q是第一个非隐藏子句中第一个非隐藏文字

{

q=q->next;

}

v=q->data;

top++;//v入栈

Stack[top][0]=v;

RemoveSpecialClause(root\_p, tail\_p,v);

Dele\_special\_lit(root\_p,-v);

//再消除一些单子句

while(isUnitClause(root)==TRUE)//有单子句

{

n=firstUnitClause(root);

p=root;//p指向第一个单子句

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

{

p=p->down;

}

q=p->next;//q是单子句的节点

while((q!=NULL)&&(q->is\_node\_hidden==YES))

{

q=q->next;

}

v=q->data;

top++;//v入栈

Stack[top][0]=v;

Stack[top][1]=1;

RemoveSpecialClause(root\_p, tail\_p,v);

Dele\_special\_lit(root\_p,-v);

}

while(isBlankClause(root)==TRUE)

{

if(Stack[top][1]==NO)//栈顶is not changed

{

Recover(root\_p,Stack[top][0]);//消除v的影响

Stack[top][0]\*=-1;//v改-v

v=Stack[top][0];

Stack[top][1]=YES;

RemoveSpecialClause(root\_p, tail\_p, v);//用-v化简

Dele\_special\_lit(root\_p, -v);

}

else if(Stack[top][1]==YES)

{

Recover(root\_p,Stack[top][0]);

//将栈顶元素出栈

Stack[top][0]=0;

Stack[top][1]=NO;

top--;

if(top==-1)

return FALSE;

//找到上一个not changed的栈内元素

while(Stack[top][1]==YES)

{

Recover(root\_p,Stack[top][0]);

Stack[top][0]=0;

Stack[top][1]=NO;

top--;

if(top==-1)

return FALSE;

}

Recover(root\_p,Stack[top][0]);//消除栈顶元素的影响

Stack[top][0]\*=-1;//v改-v

v=Stack[top][0];

Stack[top][1]=YES;

RemoveSpecialClause(root\_p, tail\_p, v);//用-v化简

Dele\_special\_lit(root\_p, -v);

}

}

}

return TRUE;

}

status Read\_to\_Create(FILE \*fp, char \*filename, HeadNode \*\*root, HeadNode \*\*tail){

int number;

if((fp=fopen(filename,"r"))==NULL)//以读的方式打开文件filename

{

return ERROR;//如果失败则报错

}

char s[100];//用来读取文件的流

while(fgets(s,7, fp)!=NULL)

{

if((s[0]=='p')&&(s[1]==' ')&&(s[2]=='c')&&(s[3]=='n')&&(s[4]=='f')&&(s[5]==' '))//读到了第一行

{

fscanf(fp, "%d", &v\_num);//读取文字数

fscanf(fp, "%d", &ziju\_num);//读取子句数

all\_ziju\_num=ziju\_num;

//先建立头结点

if(\*root != NULL)

return INFEASTABLE;

else

{

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

(\*root)->next=NULL;

(\*root)->is\_headnode\_hidden=NO;

(\*root)->why=0;

HeadNode\* p=\*root;

Node \*q=NULL;

int i;//用于初始化多个链表

//创建头结点

for( i=1 ;i<ziju\_num;i++)

{

p->data=0;

p->all\_data=0;

p->down=(HeadNode\*)malloc(sizeof(HeadNode));

p=p->down;

p->next=NULL;

p->is\_headnode\_hidden=NO;

p->why=0;

}

p->data=0;//最后一个子句初始长度为0

p->all\_data=0;

\*tail=p;

p->down = NULL;

//输入每个链表的信息

p=\*root;

for(i=1; i<=ziju\_num; i++)

{

if(fscanf(fp,"%d", &number)&&(number==0))//读到空子句

{

p->next=NULL;

p=p->down;

}

else

{

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

q->data=number;

q->is\_node\_hidden=NO;

p->data++;

p->all\_data++;

p->next=q;

while(fscanf(fp ,"%d", &number)&&(number!=0))

{

q->next=(Node\*)malloc(sizeof(Node));

q=q->next;

q->is\_node\_hidden=NO;

q->data=number;

p->data++;

p->all\_data++;

}

q->next=NULL;

p=p->down;

}

}

fclose(fp);

return OK;

}

}

else if(strlen(s)>3)

fgets(s,100,fp);

}

}

status SaveFile(FILE \*fp, char \*filename, int duration, int ans){

int i=0;

if((fp=fopen(filename,"w"))==NULL)//以写的方式打开文件filename

{

return ERROR;//如果失败则报错

}

fprintf(fp ,"s %d\n",ans);//输出结果

if(ans==1)//有解

{

fprintf(fp, "v ");

while(Stack[i][0]!=0)

{

fprintf(fp ,"%d ",Stack[i][0]);

i++;

}

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

}

else//无解

{

fprintf(fp, "0\n");

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

}

fclose(fp);

return OK;

}

void Analyze\_cnf(HeadNode\* root){

int i=1;

HeadNode\* p=root;

Node\* q=p->next;//不能写NULL->next!!!!!!!!!!!!

if(ziju\_num>0)

for(;i<=all\_ziju\_num;i++)

{

if(p->is\_headnode\_hidden==NO)

{

if(q!=NULL)

printf("%d",q->data);

q=q->next;

while(q!=NULL)

{

printf("∨");

if(q->is\_node\_hidden==NO)

printf("%d",q->data);

q=q->next;

}

printf("\n");

}

p=p->down;

if(p!=NULL)

q=p->next;//这里不能直接写q=p->next，因为没有说明NULL->next=NULL

}

}

status Validate\_ans(HeadNode \*root, HeadNode \*tail){

HeadNode \*\*root\_p=&root, \*\*tail\_p=&tail;

//先输出解析的结果

Analyze\_cnf(root);

//再输出赋值的结果

int top=0;

printf("s: ");

while(Stack[top][0]!=0){

printf("%d ",Stack[top][0]);

top++;

}

//下面进行解析

top=0;

while(Stack[top][0]!=0){

RemoveSpecialClause(root\_p,tail\_p,Stack[top][0]);

top++;

}

if(isSetBlank()==TRUE)

return OK;

else

return FALSE;

}

new\_sudoku.c的代码：

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define ERROR 0

#define OK 1

#define YES 1

#define NO 0

#define TRUE 1

#define FALSE 0

#define INFEASTABLE -1

#define random(x) (rand()%x)

typedef int status;

status SudokuExist=NO;//该变量指示数独是否初始化

int Sudoku[10][10];//数独存至二维数组中，第0行空掉

int dignumber[81][2];//里面保存要挖掉的数字的行和列

int changenumber[9][2];//数组用于改变数独中的数字

int chosen[9];//被选了的数字

int hole\_num=0;//当前洞的数量

int all\_hole\_num=0;//总共要挖的洞的数量

int SAT\_table[81][10][2];//用于存放数独转SAT的信息

int ziju\_num=0;//输出成cnf文件的子句数量

int v\_num=0;//输出成cnf文件后的文字数量

int same\_number[9];//用于暂时存放能存同一个数字的空格中同一数字对应的序号

status InitFinalSudoku(FILE \*fp);//初始化生成一个终局

void ShowSudoku(void);//输出数独格局

status MakeRandomSudoku(void);//将初始化了的终局进行改变，生成随机的终局

status ChangeNum(void);//将原有数独做相应的变换

status isNumberChosen(int number);//判断变换数字的时候数字是否已存在于数组chosen中

status DigHole(void);//对已经生成的数独进行挖洞

status sudoku\_to\_SAT(void);//将数独转换为SAT问题的信息存至数组中

status Can\_exist\_in\_line(int number, int i);//判断number在第i行中是否已存在，存在则返回NO

status Can\_exist\_in\_col(int number, int j);//判断number在第j列中是否已存在，存在则返回NO

status Can\_exist\_in\_square(int number,int i, int j);//判断number在空格所处的九宫格是否已经存在，如果存在则返回NO

status SAT\_to\_cnf\_file(FILE \*fp, char \*cnf\_filename);//将表中的SAT信息转换成cnf文件

status Solve\_Sudoku(FILE \*fp, char \*res\_filename);//将SAT解的结果读至数独中填空

int main(void){

FILE \*fp=NULL;

//现将数独全部置为0

int i,j,k;

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

{

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

{

Sudoku[i][j]=0;

}

}

//将same\_number[9]置零

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

same\_number[i]=0;

}

//准备好数字的转化

for(i=0;i<=8;i++){

changenumber[i][0]=i+11;

changenumber[i][1]=0;

}

//准备好被选数字的初始化

for(i=0;i<=8;i++){

chosen[i]=0;

}

//准备好被挖数字的行列信息初始化

for(i=0;i<=80;i++){

dignumber[i][0]=0;

dignumber[i][1]=0;

}

//将SAT\_table置零

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

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

SAT\_table[i][j][0]=0;

SAT\_table[i][j][1]=0;

}

}

int op=1;

while(op)

{

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

printf(" Menu for Sudoku Structure \n");

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

printf(" 1. InitFinalSudoku \n");

printf(" 2.ShowSudoku \n");

printf(" 3.MakeRandomSudoku \n");

printf(" 4.DigHole \n");

printf(" 5.Sudoku\_to\_SAT \n");

printf(" 6.SAT\_to\_cnf \n");

printf(" 7.SolveSudoku \n");

printf(" 0. Exit \n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch(op){

case 1:

{

if(InitFinalSudoku(fp)==OK)

printf("读取数独成功！\n");

else

printf("读取数独失败！\n");

getchar();getchar();

break;

}

case 2:

{

ShowSudoku();

getchar();getchar();

break;

}

case 3:

{

if(MakeRandomSudoku()==OK)

{

printf("随机数独生成完成！\n");

SudokuExist=TRUE;

}

else

printf("生成失败！\n");

getchar();getchar();

break;

}

case 4:

{

if(DigHole()==INFEASTABLE)

printf("数独还没初始化！\n");

else

printf("挖洞成功！\n");

getchar();getchar();

break;

}

case 5:

{

if(sudoku\_to\_SAT()==TRUE)

printf("数独转SAT的信息已经存入表中了！\n");

else

printf("转化失败！\n");

getchar();getchar();

break;

}

case 6:

{

char cnf\_filename[40];

printf("请输入要保存的cnf文件的名字:");

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

if(SAT\_to\_cnf\_file(fp,cnf\_filename)==OK)

printf("转化成功！\n");

else

printf("转化失败！");

getchar();getchar();

break;

}

case 7:

{

char res\_filename[40];

printf("请输入要用于填空的res文件名称：");

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

if(Solve\_Sudoku(fp,res\_filename)==OK)

printf("解数独成功！\n");

else

printf("解数独失败！\n");

getchar();getchar();

break;

}

}

}

return 0;

}

status InitFinalSudoku(FILE \*fp){

int number;//用于读取文件中的数据

int i,j;//用于遍历读至二维数组中

char filename[40];

printf("请输入根文件的名字(initial\_sudoku.txt)：");

scanf("%s", filename);

if((fp=fopen(filename,"r"))==NULL)//以读的方式打开文件filename

{

return ERROR;//如果失败则报错

}

//读取文件中的初始终局

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

{

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

{

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

Sudoku[i][j]=number;

}

}

fclose(fp);

return OK;

}

void ShowSudoku(void){

int i,j;

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

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

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

}

printf("\n");

}

printf("输出完成！\n");

}

status ChangeNum(void){

srand(time(NULL));//初始化随机数种子

int i,number;

for(i=0;i<=8;i++){

number=rand()%9+1;

if(isNumberChosen(number)==NO)

chosen[i]=number;

else

{

do

{

number=random(9)+1;

} while (isNumberChosen(number)==YES);

chosen[i]=number;

}

changenumber[i][1]=chosen[i];

}

return OK;

}

status isNumberChosen(int number){

int i=0;

while((chosen[i]!=0)&&(i<=8)){

if(chosen[i]==number)

return YES;

i++;

}

return NO;

}

status MakeRandomSudoku(void){

int i,j,k,number;

ChangeNum();

//先将每个元素加10，然后进行变换

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

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

Sudoku[i][j]+=10;

}

}

//开始进行变换

for(k=0;k<=8;k++){

number=changenumber[k][0];

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

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

if(Sudoku[i][j]==number)

Sudoku[i][j]=changenumber[k][1];

}

}

}

return OK;

}

status DigHole(void){

if(SudokuExist==NO)

return INFEASTABLE;

else

{

srand(time(NULL));//初始化随机数种子

int i,j,k;

printf("请输入你要挖的洞的数量：");

scanf("%d",&all\_hole\_num);

for(k=0;k<all\_hole\_num;k++){

i=rand()%9+1;//i和j是行号和列号

j=rand()%9+1;

//如果该处已经被挖过了就换一个没挖过的

while(Sudoku[i][j]==0){

i=rand()%9+1;

j=rand()%9+1;

}

dignumber[k][0]=i;

dignumber[k][1]=j;

Sudoku[i][j]=0;

hole\_num++;

}

return OK;

}

}

status Can\_exist\_in\_line(int number, int i){

int j;

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

if(Sudoku[i][j]==number)

return NO;

}

return YES;

}

status Can\_exist\_in\_col(int number, int j){

int i;

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

if(Sudoku[i][j]==number)

return NO;

}

return YES;

}

status Can\_exist\_in\_square(int number,int i, int j){

//以下分为九种情况讨论

if((i>=1)&&(i<=3)){//第一行的九宫格

if((j>=1)&&(j<=3)){//第一个九宫格

int m,n;

for(m=1;m<=3;m++){

for(n=1;n<=3;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

else if((j>=4)&&(j<=6)){//第二个九宫格

int m,n;

for(m=1;m<=3;m++){

for(n=4;n<=6;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

else if((j>=7)&&(j<=9)){//第三个九宫格

int m,n;

for(m=1;m<=3;m++){

for(n=7;n<=9;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

}

else if((i>=4)&&(i<=6)){//第二行的九宫格

if((j>=1)&&(j<=3)){//第四个九宫格

int m,n;

for(m=4;m<=6;m++){

for(n=1;n<=3;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

else if((j>=4)&&(j<=6)){//第五个九宫格

int m,n;

for(m=4;m<=6;m++){

for(n=4;n<=6;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

else if((j>=7)&&(j<=9)){//第六个九宫格

int m,n;

for(m=4;m<=6;m++){

for(n=7;n<=9;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

}

else if((i>=7)&&(i<=9)){//第三行的九宫格

if((j>=1)&&(j<=3)){//第七个九宫格

int m,n;

for(m=7;m<=9;m++){

for(n=1;n<=3;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

else if((j>=4)&&(j<=6)){//第八个九宫格

int m,n;

for(m=7;m<=9;m++){

for(n=4;n<=6;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

else if((j>=7)&&(j<=9)){//第六个九宫格

int m,n;

for(m=7;m<=9;m++){

for(n=7;n<=9;n++){

if(Sudoku[m][n]==number)

return NO;

}

}

return YES;

}

}

}

status sudoku\_to\_SAT(void){

int i,j,m,n,number,xuhao;//i和j是数独的行和列，m和n是SAT\_table的行和列,number用于尝试空格能填哪些数

m=0,n=0,xuhao=1;

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

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

//如果数独的这一格是空的就存下来能填哪些数

if(Sudoku[i][j]==0){

SAT\_table[m][n][0]=i;//空格在数独中的行号和列号填进去

SAT\_table[m][n][1]=j;

n++;

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

if(Can\_exist\_in\_line(number,i)&&Can\_exist\_in\_col(number,j)&&Can\_exist\_in\_square(number,i,j)){

SAT\_table[m][n][0]=number;

SAT\_table[m][n][1]=xuhao;

xuhao++;

n++;

}

}

m++;

n=0;

}

}

}

xuhao--;

v\_num=xuhao;//文字的数量

return OK;

}

status SAT\_to\_cnf\_file(FILE \*fp, char \*cnf\_filename){

if((fp=fopen(cnf\_filename,"w"))==NULL)//以读的方式打开文件filename

{

return ERROR;//如果失败则报错

}

int i,j,m,n,number;

for(i=1;i<=300;i++){

fprintf(fp, " ");

}

fprintf(fp, "\n");

//先输入对于一个空格而言所有能填的数

for(m=0;m<all\_hole\_num;m++){

n=1;

while(SAT\_table[m][n][0]!=0)

{

fprintf(fp, "%d ", SAT\_table[m][n][1]);

n++;

if(SAT\_table[m][n][0]==0){

fprintf(fp, "0\n");

ziju\_num++;

}

}

}

m=0,n=0;

//再输入对于一行而言能填相同数字的空格中相同数字的序号

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

for(number=1;number<=9;number++){//对于一个数字而言，找到能填这个数字的空格

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if(SAT\_table[m][n][0]==i){

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

}

//再输入对于一列而言能填的数字相同的空格中相同数字的序号

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

for(number=1;number<=9;number++){//对于一个数字而言，找到能填这个数字的空格

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if(SAT\_table[m][n][1]==j){

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

}

//再输出对于一个九宫格而言能填相同数字的空格中相同数字的序号

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=1)&&(SAT\_table[m][n][0]<=3)&&(SAT\_table[m][n][1]>=1)&&(SAT\_table[m][n][1]<=3)){

//第一个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=1)&&(SAT\_table[m][n][0]<=3)&&(SAT\_table[m][n][1]>=4)&&(SAT\_table[m][n][1]<=6)){

//第2个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=1)&&(SAT\_table[m][n][0]<=3)&&(SAT\_table[m][n][1]>=7)&&(SAT\_table[m][n][1]<=9)){

//第3个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=4)&&(SAT\_table[m][n][0]<=6)&&(SAT\_table[m][n][1]>=1)&&(SAT\_table[m][n][1]<=3)){

//第4个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=4)&&(SAT\_table[m][n][0]<=6)&&(SAT\_table[m][n][1]>=4)&&(SAT\_table[m][n][1]<=6)){

//第5个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=4)&&(SAT\_table[m][n][0]<=6)&&(SAT\_table[m][n][1]>=7)&&(SAT\_table[m][n][1]<=9)){

//第6个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=7)&&(SAT\_table[m][n][0]<=9)&&(SAT\_table[m][n][1]>=1)&&(SAT\_table[m][n][1]<=3)){

//第7个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=7)&&(SAT\_table[m][n][0]<=9)&&(SAT\_table[m][n][1]>=4)&&(SAT\_table[m][n][1]<=6)){

//第8个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

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

int k,k\_k;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=7)&&(SAT\_table[m][n][0]<=9)&&(SAT\_table[m][n][1]>=7)&&(SAT\_table[m][n][1]<=9)){

//第9个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

if(k==0)

continue;

for(k\_k=0;k\_k<k;k\_k++){

fprintf(fp,"%d ",same\_number[k\_k]);

}

fprintf(fp, "0\n");

ziju\_num++;

}

//下面输出同行中能填相同数字的空格，每个数字只能在一个空格中出现

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

for(number=1;number<=9;number++){//对于一个数字而言，找到能填这个数字的空格

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if(SAT\_table[m][n][0]==i){

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

}

//下面输出同列中能填相同数字的空格，每个数字只能在一个空格中出现

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

for(number=1;number<=9;number++){//对于一个数字而言，找到能填这个数字的空格

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if(SAT\_table[m][n][1]==j){

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

}

//再输出同一个九宫格中能填同一数字的空格中同一数字只能在一个地方出现

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=1)&&(SAT\_table[m][n][0]<=3)&&(SAT\_table[m][n][1]>=1)&&(SAT\_table[m][n][1]<=3)){

//第1个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第2个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=1)&&(SAT\_table[m][n][0]<=3)&&(SAT\_table[m][n][1]>=4)&&(SAT\_table[m][n][1]<=6)){

//第2个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第3个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=1)&&(SAT\_table[m][n][0]<=3)&&(SAT\_table[m][n][1]>=7)&&(SAT\_table[m][n][1]<=9)){

//第3个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第4个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=4)&&(SAT\_table[m][n][0]<=6)&&(SAT\_table[m][n][1]>=1)&&(SAT\_table[m][n][1]<=3)){

//第4个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第5个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=4)&&(SAT\_table[m][n][0]<=6)&&(SAT\_table[m][n][1]>=4)&&(SAT\_table[m][n][1]<=6)){

//第5个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第6个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=4)&&(SAT\_table[m][n][0]<=6)&&(SAT\_table[m][n][1]>=7)&&(SAT\_table[m][n][1]<=9)){

//第6个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第7个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=7)&&(SAT\_table[m][n][0]<=9)&&(SAT\_table[m][n][1]>=1)&&(SAT\_table[m][n][1]<=3)){

//第7个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第8个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=7)&&(SAT\_table[m][n][0]<=9)&&(SAT\_table[m][n][1]>=4)&&(SAT\_table[m][n][1]<=6)){

//第8个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//第9个九宫格

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

int k,k\_k,start,end,middle;

k=0,k\_k=0;

for(m=0;m<all\_hole\_num;m++){

if((SAT\_table[m][n][0]>=7)&&(SAT\_table[m][n][0]<=9)&&(SAT\_table[m][n][1]>=7)&&(SAT\_table[m][n][1]<=9)){

//第9个九宫格

n++;

while(SAT\_table[m][n][0]!=0){

if(SAT\_table[m][n][0]==number){

same\_number[k]=SAT\_table[m][n][1];

k++;

break;

}

n++;

}

}

n=0;

}

end=k;

if((k==0)&&(k==1))

continue;

for(start=0;start<end;start++){

middle=start+1;

for(;middle<end;middle++){

fprintf(fp, "%d %d 0\n",-same\_number[start], -same\_number[middle]);

ziju\_num++;

}

}

}

//再输出对于每个空格，只能有一个数填进去

for(m=0;m<all\_hole\_num;m++){

n=0;

int start=1,end=start,middle;

while(SAT\_table[m][end][0]!=0){

end++;

}

if(end==2){

fprintf(fp,"%d 0\n",SAT\_table[m][1][1]);

ziju\_num++;

}

for(start=1;start<end;start++){

for(middle=start+1;middle<end;middle++){

fprintf(fp, "%d %d 0\n", -SAT\_table[m][start][1], -SAT\_table[m][middle][1]);

ziju\_num++;

}

}

}

//下面将前几行的注释打印出来

rewind(fp);

fprintf(fp, "c sudoku, order: 9, holes: %d\n", all\_hole\_num);

fprintf(fp, "c \n");

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

fprintf(fp, "c ");

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

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

fprintf(fp,"%d",Sudoku[i][j]);

}

else{

fprintf(fp,".");

}

}

fprintf(fp,"\n");

}

fprintf(fp,"c \n");

fprintf(fp,"p cnf %d %d\n", v\_num, ziju\_num);

fclose(fp);

return OK;

}

status Solve\_Sudoku(FILE \*fp, char \*res\_filename){

if((fp=fopen(res\_filename,"r"))==NULL)//以读的方式打开文件filename

{

return ERROR;//如果失败则报错

}

char s[100];

fgets(s,100,fp);

fgets(s,2,fp);

int m,n,i,j,xuhao;//m,n是SAT\_table的行列号、i,j是数独的行列号

while(fscanf(fp,"%d", &xuhao)&&(xuhao!=0)){

if(xuhao>0){

for(m=0;m<=all\_hole\_num;m++){

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

if(SAT\_table[m][n][1]==xuhao){

i=SAT\_table[m][0][0];

j=SAT\_table[m][0][1];

Sudoku[i][j]=SAT\_table[m][n][0];

break;

}

else if(SAT\_table[m][n][1]==0){

break;

}

}

}

}

}

fclose(fp);

return OK;

}