

**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： CS2009班**

**学 号： U202015562**

**姓 名： 徐雨梦**

**指导教师： 向文**

**报告日期： 2021.10.10**

**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。

**（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。

**（3）DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。

**（4）时间性能的测量：**基于相应的时间处理函数，记录DPLL过程执行时间， 并作为输出信息的一部分。

**（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。

**（6）SAT应用：**将二进制数独游戏问题转化为SAT问题，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。

**目录**

**任务书** 2

**1引言** 4

1.1课程背景与意义 4

1.2国内外研究现状 5

1.3课程设计的主要研究工作 5

**2 系统需求分析与总体设计**×××××× 6

2.1系统需求分析 6

2.2系统总体设计 6

**3系统详细设计** 8

3.1有关数据结构定义 8

3.2主要算法设计 10

**4系统实现和系统测试** 19

4.1系统实现 19

4.2系统测试 27

**5总结与展望** 37

5.1全文总结 37

5.2工作展望 37

**6体会** 38

**附录 源代码** 40

**引言**

## 1.1课题背景与意义

**1.1.1背景**

SAT问题，即可满足性（SATisfiability）问题，是历史上第一个被证明为NP-完全的问题，也是人工智能领域的核心问题，并在许多工业应用领域中具有非常重要的意义和广泛的应用。例如，Intel芯片和Windows操作系统验证中都用到了SAT求解器。为促进SAT问题求解算法及工具的研发和应用，国际SAT学会自2002年以来每年（或每两年）组织SAT Competition国际比赛，至今已经举办了13届。 长期以来，各国学者对其进行了广泛而深入的研究，而SAT算法竞赛也成为了该领域中的标志性学术事件。

SAT 求解领域在过去几年取得了巨大进步。许多十年前似乎完全无法解决的问题（例如硬件和软件验证）现在可以常规处理。除了新算法和更好的启发式方法外，改进的实现技术对于这一成功也至关重要。

**1.1.2意义**

作为第一个被证明的NP完全问题，SAT问题的求解对于NP=P问题的推理至关重要，被称为NP完全问题的种子。

SAT 问题的解决对其他的NP类问题有很大作用。所有NP类问题都可以在多项式时间内转换为SAT问题。如果SAT问题可以在多项式时间内解决，其他的所有NP 类完全问题都可以在多项式时间内解决，这对解决NP完全类的问题至关重要。

SAT求解器的应用领域也十分广泛。它不仅是计算机科学和人工智能领域的核心问题，也可以应用到实际应用中的许多领域，工程技术、军事、工商管理、交通运输及自然科学研究中的许多重要问题，如程控电话的自动交换、大型数据库的维护、大规模集成电路的自动布线、软件自动开发、机器人动作规划等，都可以转化成SAT问题，利用SAT求解器进行问题的解决。而在数学研究领域中，它还能用来解决旅行商问题和逻辑算术问题。

而目前SAT问题在最坏情况下的求解时间非常长，不仅是计算机科学上的一个难以翻越的大山，在实际应用中，求解效率低也会产生额外的大量的消耗。因此致力于寻找求解SAT问题的快速而有效的算法，不仅在理论研究上而且在许多应用领域都具有极其重要的意义。

## 1.2国内外研究现状

作为计算机科学和人工智能领域的核心问题，sat问题引来了国内外众多的计算机学者的关注。针对sat问题的求解，每两年还会举办一次 SAT Competitions and SAT-Races / Challenges。吸引和激励了广大计算机领域的研究人员对SAT问题进行研究。因此出现了多种多样的SAT解决方案。

在最经典的求解SAT问题的算法DPLL的基础上，出现了许多改进的算法。比如CDCL算法，在决策变量选择策略上，VSIDS的基础上，又产生了许多改进算法，比如EVSIDS等。

国外研究SAT问题较早，最基础的dpll算法由国外的Davis 和 Putnam 等

人于1960年提出，之后的国外的一些学者也提出了很多改进算法，比如walksat等，对sat问题的求解有很大的推动作用。

我国在SAT方面的研究逐年深入，96年黄文奇教授提出了solar算法，获得了sat彼年的冠军，最近几年，我们国家的代表队都取得了很不错的成绩，17年华科获得了SAT竞赛的冠军，值得一提的是，黄文奇教授也是华科代表队的。20年中国科学院软件研究中心在SAT竞赛中获得了冠军，可见，我们国家对该方面的内容的研究逐渐深入。

**1.3课程设计的主要研究工作**

本次课设主要学习了DPLL算法和几个优化算法以及数独的生成与求解。DPLL算法是解决SAT问题的完备算法，是函数的主干。优化可以从数据结构的构建、决策变量的选择策略、回溯机制的改进、递归改迭代等方面优化，但是本次课设只从决策变量的选择策略上进行了改进。数独的生成与求解，生成具有唯一解的数独，采取了挖洞法，将数独转换成sat问题解决得标准格式CNF范式，用sat求解器求解后，将答案的标准格式转化为数独，得到数独的解。

因此本次课设的主要研究工作为：

①DPLL实现sat问题的求解

②优化算法

③实际应用：数独

**系统需求分析与总体设计**

**2.1系统需求分析**

本次课设所要设计的系统所要实现的目标功能：

①输入输出功能：cnf文件读取，输出为指定文件形式。

②公式的解析与验证：根据自己设计的数据结构将读取的数据进行保存和数据构建。

③DPLL代码实现：根据DPLL框架，用代码实现。

④时间性能测试：给出DPLL的求解时间。

⑤程序优化：在DPLL原有的框架基础和自己写的原代码上进行优化并分析优化效率。

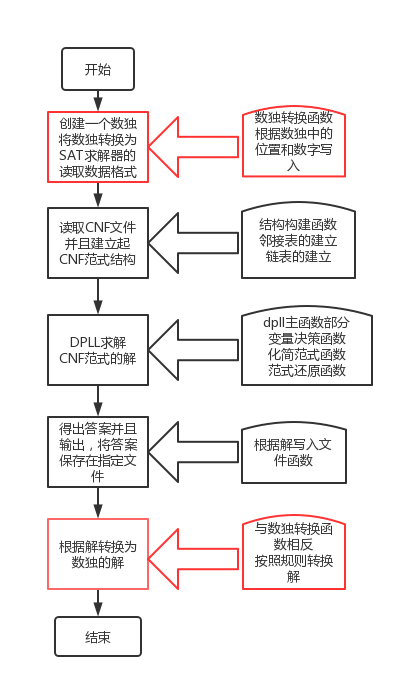
⑥SAT应用：将数独转换为sat问题求解。

以上为SAT问题的基本解决框架和利用SAT求解器解决实际问题的大框架。解决数独问题的过程大概为：将需要求解的数独转变为标准的SAT求解器读取的数据格式；SAT求解器对该组数据进行读取并且求解，输出SAT求解器的标准输出格式；将输出的数据再转换为数独的解，数独求解完毕。

**2.2系统总体设计**

系统分为两大块：SAT求解部分和数独部分。

如图所示为系统的总体设计，根据系统设计需求，分为五个部分。分别是CNF文件读取部分、DPLL求解部分、保存解部分、数独转换CNF部分、文件转换数独解部分。



图中黑色框架部分为DPLL求解的模块，红色框架部分为SAT应用数独所添加模块。

**系统详细设计**

**3.1有关数据结构的定义**

首先给出数据结构：

#define ElemType int

#define maxsize 1005

#define SUCCESS 2

#define INFEASIBLE -1

#define TRUE 1

#define FASLE 0

#define OUT -2

#define ERROR 0

#define OK 1

typedef struct letter//式子中的变量文字 结构体

{

int flag;

ElemType prop;//变量文字

struct letter \*next; //子句的下一个文字

}LETTER,\*LETS;

typedef struct formula //子句结构体

{

int flag;//标志该子句是否已经被处理

int pos;//子句所在位置

int number;//一个子句中的变量文字的数目

LETTER \*headp;//一个子句中的变量文字头结点

struct formula \*next;//下一个子句

int satisfied;

int unsatisfied;

}FORMULA,\*FORM;

typedef struct variable//与变量文字有关的所有子句构成的结构体

{

ElemType key;//文字

ElemType \*prop;//逻辑变量的栈

int number;//与该变量文字有关的子句的数目

int maxlength;//最多存放

LETS \*lets;

FORM \*s;//存放上述子句的栈

}VARY;

创建的数据结构有三个，变量文字结构体、子句结构体、与变量文字有关的所有子句构成的结构体。子句结构体为读取文件后保存CNF范式的结构体，是一个邻接表，主枝保存链接的子句的头指针和下一个子句结构体的结点，链接的子句为一个变量文字结构体链接的链表，里面储存了一个变量和下一个变量的结点。VARY结构体中存放保存的文字变量，和三个栈：包含该文字的子句的栈、保存该文字所在的子句中的该位置存放的位置的栈、保存该子句中该文字存在的形式的栈。

子句结构体中额外设置flag，用来标志该子句是否已经可满足，number用来计数子句中的文字的个数，satisfied和unsatisfied用来计数子句中真值为1的文字的数目和真值为0的文字的数目。

LETTER结构体保存了逻辑变量和真值。

VARY中的三个栈为指针型，设定了一个初始最大值，为其分配空间，如果栈满，就重新分配，保证所需内存不需过多但是不能溢出。

**3.2主要算法设计**

整个项目分为四大板块：文件的读取和公式的解析、DPLL求解过程、程序优化部分、数独部分。

**3.2.1 文件的读取和公式的解析**

SAT问题要求解一个CNF范式的解，无解时输出s0，有解时输出使CNF范式真值为1的一组文字变量的解。

CNF范式保存在后缀为.cnf的文件中，本次的文件中的变量文字用编号表示，-号表示为变量取非，文件读取的过程中对CNF范式进行解析和结构体构建。

根据文件读写的规则，首先创建好结构体，在读文件的时候，将读取到的变量直接保存到邻接表里面，同时对邻接表的其他变量进行初始化和更新。在文件读取过程中，由于有注释，在读取过程中，读取到p后的第一个数字即可以开始建立邻接表，因此可以使用ungetc函数对读取到的第一个数字字符返回输入流，之后便开始读取数字。

在建立好邻接表后，需要建立好VARY类型的变量，声明一个VARY类型的数组，数组长度为变量文字个数，对该数组中的每个成员进行赋值。此时可以遍历建立好的邻接表，在遍历CNF范式，对变量文字所具有的性质进行初始化和更新。同时在更新栈的时候需要注意是否溢出，溢出则重新分配。

至此，文件的输入功能已经实现，并且已经实现了对cnf公式的解析和所需数据结构的建立。

**3.2.2 DPLL求解过程**

DPLL求解为整个项目的核心部分，对建立好的CNF范式结构进行求解。

它是基于树/二叉树的回溯搜索算法，在DPLL求解过程中，需要对CNF范式不断地进行化简，化简规则大概有：单子句规则、纯文字规则、消除原子公式规则、重言子句规则、包含规则、分裂规则。其中的大多数规则技巧性更强，因此在代码实现过程中，主要使用两种基本处理策略：单子句规则和分裂策略。

单子句规则。如果子句集S中有一个单子句L,那么L一定取真值，于是可以从S中删除所有包含L的子句（包括单子句本身），得到子句集S1，如果它是空集，则S可满足。否则对S1中的每个子句，如果它包含文字¬L,则从该子句中去掉这个文字，这样可得到子句集合S2。S可满足当且仅当S2可满足。单子句传播策略就是反复利用单子句规则化简S的过程。

分裂策略。按某种策略选取一个文字L.如果L取真值，则根据单子句传播策略，可将S化成S2；若L取假值（即¬L成立）时，S可化成S1.

交错使用上述两种策略可不断地对公式化简，并最终达到终止状态，其执行过程可表示为一棵二叉搜索树。

基于单子句传播与分裂策略的DPLL算法可以描述为一个递归或者迭代过程。

算法过程：

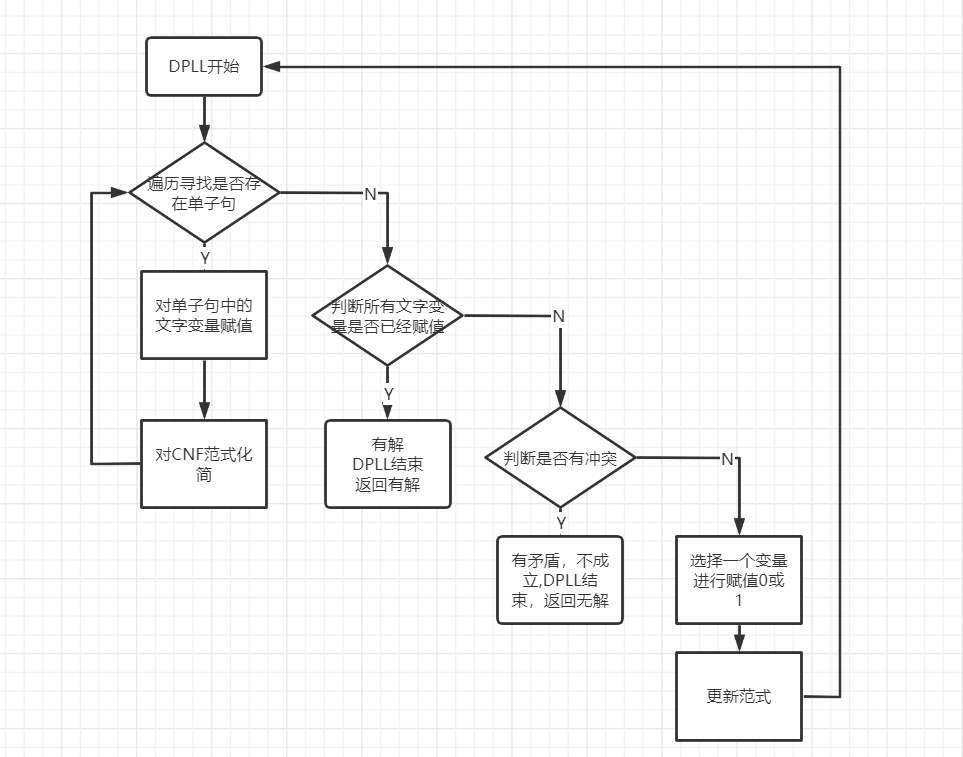
①遍历范式寻找一个单子句，如果存在单子句，对该单子句中的变量文字进行赋值，反之执行步骤③；

②根据上面已经被赋值的变量文字对范式进行化简，范式被更新，继续执行步骤①；

③判断当前是否已经确定CNF的解，如果当前所有的文字变量都被赋值，返回TRUE，表示有解；如果有冲突，就返回FASLE；

④根据某种策略选择一个变量赋值0或1，得到了一个新的范式，递归进行求解。

算法流程图如下；



**3.2.3 程序优化**

程序优化可以从数据结构、变量决策策略等方面进行优化。本次课设主要从递归改迭代和变量选取策略方面进行优化。

**3.2.1 递归改迭代**

递归在整个过程中所需空间和时间都太长，尤其在回溯阶段，因此，将回溯写进DPLL主函数中，可以当作一个不错的优化策略。

设置两个栈，一个栈s保存被处理过的文字变量，另一个栈record用来保存决策变量在s中所处的位置，在原本的DPLL过程中，判断是否存在单子句部分优化为是否找到了可以用来化简的文字，对选择决策变量之后递归改为选择了决策变量后对变量进行保存，这样就可以让决策变量在循环中返回判断是否找到了化简文字方面；发生冲突返回FASLE部分优化为对CNF范式恢复为冲突点的状态。

算法过程为：

①根据单子句规则找到一个变量文字key，如果没有找到，key赋值0，执行步骤⑤；

②对key进行入栈s，并根据key对范式进行化简，化简采用标记的方法；

③遍历范式寻找一个单子句，如果存在单子句，对该单子句中的变量文字进行赋值，反之执行步骤⑤；

④对所找的单子句中的文字变量赋给key，继续执行步骤②；

⑤判断当前是否已经确定CNF的解，如果当前所有的文字变量都被赋值，返回TRUE，表示有解；如果有冲突，执行⑦；反之，继续执行；

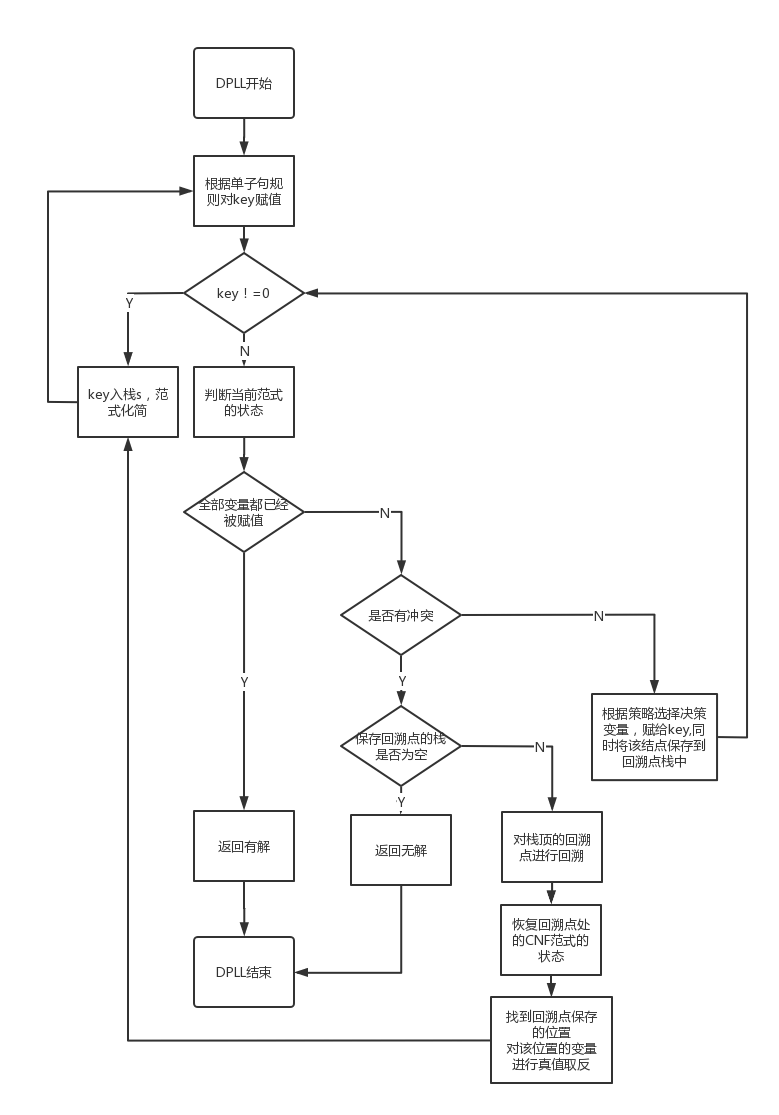
⑥根据某种策略选择一个变量赋给key，返回步骤②；

⑦发生冲突，判断保存回溯点的栈record是否为空，如果为空，该CNF范式无解，返回FASLE；如果不为空，执行⑧；

⑧根据被处理的文字栈和回溯点保存栈对回溯点保存栈的栈顶保存的结点进行回溯，对范式恢复原本的状态；

⑨对回溯点保存的节点位置的值赋给key，对key进行取反，返回执行②。

流程图部分如图所示：



**3.2.2变量决策策略优化**

对决策变量的选择对整个求解器的效率有很大的作用，如果一开始就选对变量，那么就会减少很多的搜索和回溯过程，在空间上和时间上对程序都有很大程度的优化。

但是目前为止，没有一种决策策略可以适应于所有的算例，对于不同的算例，策略的优劣不同，因此，每一种变量决策测优化策略都是针对某一部分算例来说的，在运行的时候，可以按照算例的复杂程度等特点选择合适的优化策略。

1. MOM策略

MOM策略又称最短子句出现频率最大优先策略。也就是说与文字有关的所有子句的某种权值和最大的文字优先作为决策变量。该权值和的计算方法为：指数为子句长度的负数，对所有有关的子句进行该操作。

一个子句的长度越大（所含文字数多），那么使得该子句可满足的“候选”文字数目就越多，相对而言该子句就越容易得到满足；而如果子句长度越小（所含文字数少），则越不易得到满足。因此选择对应的子句的长度和较小的文字选为决策变量。

MOM 策略对于子句长度相差较大的范式来说较为使用，对于长度均匀相等的范式，每个文字对应的权值基本相同，此时利用MOM选择决策变量没有什么特别之处，反而会增加时间，降低效率，因为每次选择决策变量是，都需要进行一次遍历比较，在最坏每一条路径都需要n²等级的复杂度，会大大降低效率。

|  |  |  |
| --- | --- | --- |
| 算例 | 原代码 | MOM |
| 1.cnf | 43629ms | 47562ms |
| 2.cnf | 283356ms | 603209ms |
| 3.cnf | 8ms | 10ms |
| 4.cnf | 1817ms | 1630ms |
| 5.cnf | 35ms | 36ms |

根据表中数据可知，MOM策略在所测得几个算例中，并没有起到很好的效率提高的效果。甚至降低了效率。

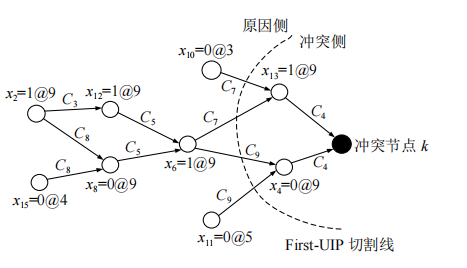
在观察了这几个测试算例可知，这几个算例有一个共同的特点，各个子句的结构体的长度都非常接近，以至于MOM策略中的各个文字的权重非常接近，从而并没有一般的变量选择快，甚至MOM策略还需要时间去给权值排序，从而消耗了更多的时间。

1. VSIDS策略

VSIDS策略又名独立变量状态衰减和策略。该策略是适应范围较为广泛并且提高效率程度更大的一个，目前的许多策略都是它的变体。

该策略主要为计数、冲突子句学习、 “权值”排序得到决策变量、定期衰减更新权值。

冲突子句学习过程是很关键的一个过程：



若布尔传播过程中产生冲突, 则求解器进入冲突分析过程, 冲突分析结束后将会产生一个学习子句。 经证明认为, 在冲突分析过程中根据第一唯一蕴含点学习得到的子句是最有效的。

将产生冲突的几个点组合构成一个子句，该子句加入到范式中去，可以给之后的变量决策提供经验，减少回溯的次数，提高效率。

VSIDS 策略的具体操作如下：

①为每一变量的正、负文字各设立一个计数器，初始值一般设置为该文字在所有的子句集中出现的次数。

②若文字在某一子句中出现（包含最初的子句，也包含学习子句），则将文字对应的计数器值增 1；有冲突发生的时候，就将冲突导出子句添加到子句集中并增加冲突导出子句中相应文字的计数器值。

③根据未赋值变量中的文字计数器的值的排序，从中选择一个使得计数器值最大的文字作为决策变量，若存在几个相同的最大值，那么随机选择其中一个文字对其赋值并执行布尔约束传播过程即可。

④为了防止一些变量长时间得不到赋值，经过一定的决策次数后，每一文字的计数器的值都会除以一个常数，这样做也可以增加新近学习到的一些子句对搜索过程的贡献。

**3.2.4 数独部分**

**1.数独生成**

要产生一个只有唯一解的数独，首先需要得到一个已经满足数独规则的9\*9的数组，然后在该数组中找一些位置，作为一个要填的数独的提示值。

1. 随机产生

首先确定所给数的个数num，num随机生成，加一个确定值使其取值在一定的范围内。

然后产生num个数组，数组中包含三个值，一个是行位置，一个是列位置，一个是该位置所要填写的数字，然后根据数独规则判断该位置填该值是否合法，如果不合法，为无效数组，合法为有效数组，直到生成num个有效数组，得到一个数独。其他位置填0。在生成随机数时，需要取余保证行列位置在0-8，数字在1-9。

但此时不能保证数独有解。为了保证数独有解，对数独进行SAT问题转换然后求解，如果有解，该数独有效并输出，否则继续生成。直到生成一个有解的数独并得到它的解。

1. 交换行列法

首先需要一个已经符合规则的数独格局，每一个空都有数字。

对数独的进行交换和列交换。由于数独的规则，在进行行交换和列 交换时，需要在1-3，4-6，7-9这几个板块内进行交换。故可以先产生一 个随机数确定要进行交换的模块，然后再产生两个随机数表示该模块里 交换的位置的。

行列交叉进行上述操作，得到一个新的数独格局。

（3）挖洞法保证为一解

首先根据上述的方法得到了一个数独格局，然后现在要生成一个可 以填写进行互动的数独。

首先随机生成一个数num，对其进行取余和加某个确定值的操作对 其进行取值范围的确定。num为我们需要挖掉的洞数。

整个过程如下：

随机产生一个数对（行，列）；判断如果该处被挖掉，该数独是不 是就有其他解了，如果有，那就保留该位置不挖，如果没有，该位置可 挖，直到将可挖的挖到num个为止。

判断一个位置是否可挖的方法为：在将数独问题转换为SAT问题求 解的时候，将该位置标记为无值并且不可以填写原本的那个数字，然后 求解，此时如果无解，则说明该空可挖，如果有解，就说明该空不可挖。

1. **数独的求解**

数独的求解过程主要为：数独转换为CNF范式，利用SAT求解器求解，输出格式转换为数独格式。SAT求解器就是上面的DPLL部分。

1. 数独转换为CNF范式

对数独的每个位置和数字有一个（行，列，数字）的数组，对该数组进行下列转换：文字=行\*100+列\*10+数字；这样可以对每个位置上的每个数进行文字转换，每个位置都可以填1-9，故每个位置可对应九个逻辑变量。

在转换过程中，文字\*（-1）表示该位置不可填对应的数字。根据数独的格局，和数独的规则：每一行、每一列、每一块不能有重复的值，所以可以的到一些位置的一些值是不可取的，此时可以写入对应文字的非值。对每一个有数字的位置都进行对它所在的块、行、列的其他位置的标记。

此外，还要保证每一行、每一列、每一块都要又1-9九个不一样的数字。比如：行\*100+k\*10+数字代表的九个文字组成一个子句写入文件表示该行一定要有该数字，同理其他行、列、块都可以这样表示。这样保证了数独的基本规则：每一行、每一列、每一块都不能有重复的数字。由于每一行、列、块都有1-9九个不同的数字，而且只有九个位置，所以上述子句可以使数独的基本规则成立。

两个部分的子句相结合，便可以得到一个完整的求解一个数独所需要的CNF范式。

1. CNF的解转换为数独

CNF的解为每个文字变量的真值确定。

根据文字变量和行、列、数字的转换，可以得到逆转换。根据逆转换和真值，可以确定每个位置哪个值为真。便可以得到数独的解。

**4系统实现与测试**

**4.1系统实现**

**4.1.1 系统目标**

设计该系统旨在求解SAT的基准算例，并将该求解器应用到实际应用中去。本次课设的实际应用为数独求解器。

**4.1.2 软硬件环境**

操作系统：Windows10家庭中文版

系统版本：基本x64的系统 版本20H2 内部版本19042.1237

编译调试：TDM-GCC 4.9.2 64-bit Debug

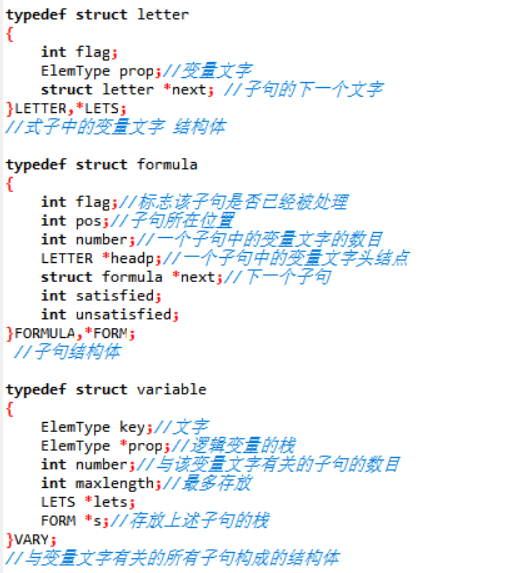
编译器：Dev C++

运行内存：7.8GB

CPU：i7-10510U

**4.1.3 函数实现**

1.数据结构：



2.具体函数：

（1）文件输入函数

函数声明：FORMULA\* cnf\_read(char \*filename,int \*num1,int \*num2)

函数功能：读取名称为fiename的cnf格式文件，将范式存放在链表中

函数实现：构建一个邻接表的头部，边读取数据边延申，直到读到0为 一个子句，每再读取到一个子句就向下next。同时在读取建立邻接表的过程 中，对每个子句的一些标志值进行更新。

（2）建立文字数组

函数声明：VARY\* RelaCreate(FORMULA \*head,int num)

函数功能：根据范式将与每个变量文字有关的子句联系起来，将与某 文字有关的子句都存放到指定的栈中

函数实现：遍历文件输入函数建立的邻接表，对文字的性质进行统计， 将文字所在的子句保存在栈中。还有变量位置、逻辑变量、个数等进行更新。

（3）输出子句函数

函数声明：void cnftraverse(FORMULA \*head)

函数功能：遍历建立的范式并输出。

函数实现：遍历邻接表，对每个子句进行输出，输出以一个子句为一行，以0作为一个子句的结束。

（4）查找单子句函数

函数声明：int get\_singlekey(FORMULA \*head)

函数功能：从单子句中找文字

函数实现：遍历整个邻接表，找出unsatisfied的值为1的子句，并将该子句中唯一一个真值尚未确定的变量作为返回值返回。从而使程序能够实现单子句规则。

1. CNF范式化简函数

函数声明：void cnfsimply(FORMULA \*head,ElemType key,int \*flag,VARY \*rela)

函数功能：根据已经确定的变量真值对范式进行化简

函数实现：首先对传入的变量进行真值赋值，如果传入的变量为文字的非值，对文字的真值标记-1，不标记为0是为了方便辨别哪个文字已经被赋值了。

根据该文字对应的VARY结构体，找到文字所出现的子句，在这些子句中，找到与该文字有关的变量，根据文字的真值判断该变量的真值，并将该真值赋给该变量结构体中储存真值的部分，继而更新所在子句的satisfied的值和unsatisfied的值。对文字出现的所有子句都进行该操作。同时，如果所在子句中的变量真值为真的话，对子句结构体中保存真值的变量进行更新。

1. CNF状态回溯函数

函数声明：void cnfrestore(FORMULA \*head,int \*s,int start,int end,VARY \*rela,int \*flag)

函数功能：对栈中的需要回溯的一部分，恢复回溯点时的状态

函数实现：该函数需要将s中从start到end的变量重新归于不确定，将CNF范式回归到start前的状态。对栈中的这一部分的变量遍历。

对于栈中的每一个变量，首先将其对应的flag标0，恢复其真值的不确定性，然后针对该文字对应的VARY结构体中保存的子句栈进行遍历，对比子句中的变量和s栈中的变量，如果不相同，那么该子句中的不可满足的减一，如果相同，该子句中真值为真的文字减一，并对子句中的变量结构体中表示真值的变量进行恢复。同时判断该子句的真值为真的文字的个数，如果为0，就对该子句的真值进行更新。

对栈中的start到end部分的每一个变量都进行上述操作，邻接表回溯到之前的状态。

1. 选择决策变量

函数声明：int get\_key(int \*flag,int num,VARY \*a)

函数功能：选取决策变量

函数实现：从第一个变量遍历，如果对应的flag为0,返回该文字的值，如果全部都被复制了，就返回0。在之前的赋予真值的过程中，真值为假的变量标记为-1而不是0，有助于这里判断。

1. mom策略为变量积分

函数声明：void getscore(FORMULA \*head)

函数功能：为每个变量计分

函数实现：遍历邻接表，遍历邻接表的每个子句，对每个变量文字对应的score进行计数累加，根据子句的长度进行指数运算，得到一个子句的分值。加在该子句中拥有的文字的计数上。

对每一个子句和每一个子句中的文字都进行上述操作。

1. MOM策略获取决策变量

函数声明：int get\_key\_mom(FORMULA \*head,int \*flag,int num)

函数功能：mom法获得决策变量

函数实现：从第一个变量开始遍历，找到第一个对应的flag为0的文字，将其指定为初始的key值，继续遍历，如果找到对应的flag为0的文字，与key比较权重值，如果更大，就更新key。

遍历结束，返回key值，如果全部被赋值了，就返回0。

1. 范式检查函数

函数声明：int dpllcheck(FORMULA \*head)

函数功能：检查当前范式的状态，真值已确定/不可满足/尚未确定

函数实现：遍历邻接表，对每个子句的状态进行检查，如果遍历到一个子句的不可满足的文字个数等于子句中文字的个数，就返回INFEASIBLE，表示出现了冲突。

重新遍历邻接表，如果有子句的真值为0，那么就返回TRUE，表示该CNF范式尚未确定。如果全部的真值都为1，那么该CNF范式已经可满足，已经有解，返回SUCCESS。Flag数组对应每个文字的真值。

1. DPLL过程

函数名称：int dpll(FORMULA \*head,int \*flag,int num,VARY \*rela)

函数功能：dpll求解范式真值的过程

函数实现：首先找一个单子句，对CNF进行第一步的化简，采用do-while循环，循环条件为key不为0，也就是尚有文字未赋真值。采用do-while是为了保证初始能够进入循环。

根据key对CNF范式进行化简，并将key入栈s，s是保存已经赋过真值的文字的栈。然后针对已经化简的CNF范式找单子句，如果找到了，就将单子句中的文字赋给key，循环此过程。直到没有单子句，即key赋值为0。

对CNF范式进行检查，得出CNF范式的状态，是已满足、发生冲突还是继续。

如果已满足，就返回TRUE，并且此时得到一组解flag。

如果发生了冲突，就回溯，回溯record保存的可能冲突点，如果保存回溯点的栈为空，说明冲突发生在CNF范式，而不是决策变量的赋值，返回FASLE，表示该CNF范式无解。

如果没有满足也没有发生冲突，就通过变量决策策略选取一个文字作为决策变量，并且赋真值，文字赋给key，同时将该文字所在的位置保存在回溯点栈中，方便冲突发生时回溯。

继续循环，直到key为0，也就是所有变量都已经被赋值为止。

1. MOM策略的DPLL

函数名称：int dpll\_mom(FORMULA \*head,int \*flag,int num,VARY \*rela)

函数功能：dpll求解范式真值的过程

函数实现：第一步对每个文字的权重进行赋值。接下来按照DPLL的步骤求解。首先找一个单子句，对CNF进行第一步的化简，采用do-while循环，循环条件为key不为0，也就是尚有文字未赋真值。采用do-while是为了保证初始能够进入循环。

根据key对CNF范式进行化简，并将key入栈s，s是保存已经赋过真值的文字的栈。然后针对已经化简的CNF范式找单子句，如果找到了，就将单子句中的文字赋给key，循环此过程。直到没有单子句，即key赋值为0。

对CNF范式进行检查，得出CNF范式的状态，是已满足、发生冲突还是继续。

如果已满足，就返回TRUE，并且此时得到一组解flag。

如果发生了冲突，就回溯，回溯record保存的可能冲突点，如果保存回溯点的栈为空，说明冲突发生在CNF范式，而不是决策变量的赋值，返回FASLE，表示该CNF范式无解。

如果没有满足也没有发生冲突，就通过变量决策策略选取一个文字作为决策变量，并且赋真值，文字赋给key，同时将该文字所在的位置保存在回溯点栈中，方便冲突发生时回溯。此处的变量决策变量采用mom策略。

继续循环，直到key为0，也就是所有变量都已经被赋值为止。

1. 写入文件函数

函数声明：int resprintf(char \*filename,int judge,int \*flag,int num,int period)

函数功能：将结果写入指定文件

函数实现：将原文件的文件名复制到一个字符数组中，然后对该数组进行最后三个字符的修改，根据每个文字和对应的flag将文字的真值写入。

1. 数独合法性判断函数

函数声明：int sudoku\_judge(int a[10][10],int i,int j,int number)

函数功能：判断当前的数独是否合法

函数实现：判断第i行第j列中填写number是否符合规则。

首先判断该位置是否已经有值，如果已经有值了，就返回FASLE。

对数独进行行、列、块判断。看每一行中是否已经存在number了，返回FASLA；看每一块中是否已经存在number了，返回FASLA；看每一块中是否已经存在number了，返回FASLA；如果以上都未出现，返回TRUE。在判断块的时候需要主要每一块的三个从哪里开始。

1. 随机生成数独函数

函数声明：int sudoku\_make(int a[10][10])

函数功能：生成一个数独

函数实现：首先srand初始化一个随机种子。

之后随机数一个数num，表示数独格局给予的数。对num进行取余和加一个固定值的操作使num处于一个合法的取值范围。接下来生成数组（行，列，数），并调用sudoku——judge函数判断该数组是否合法，如果合法就赋值，继续生成数组操作，直到生成num个合法的数组，得到数独格局。

1. 数独输出函数

函数声明：void sudoku\_print(int a[][10])

函数功能：输出数独

函数实现：对数独格局输出，给的提示数进行输出，没有提示数的位置输出.。

1. 数独转换为CNF函数

函数声明：int sudoku\_cnf(int \*flag,int a[][10],char \*filename)

函数功能：将数独转换成cnf格式并保存至文件

函数实现：根据生成的数独格局，里面有已经赋值过的位置，对这些位置进行遍历，对于每个位置，根据数独规则，对文字变量进行读取。

每个位置，根据规则，将该位置所在的行的其他位置标记不可以填写该数字，也就是该位置该数字对应的文字为假；将该位置所在的列的其他位置标记不可以填写该数字，也就是该位置该数字对应的文字为假；将该位置所在的块的其他位置标记不可以填写该数字，也就是该位置该数字对应的文字为假；对该位置标记该数字对应的文字为真，该位置其他数字对应的文字为假。

将所有已给数字的位置执行上述操作。

再对每一行、块、列都要有1-9的规则写入。具体实现为：111 121 131 141 151 161 171 181 191 0为一个子句写入，表示第一行一定有一个1，其他行、列、块同理。

对已经标记过真值的数独进行每个位置每个数字的真值的写入，再对每一行、列、块不能有重复的数字写入文件，也就是每一行列块都要有1-9不同的数字。

将上述写入文件，就转换成了SAT求解器要读取的格式。

1. CNF 转换为数独函数

函数声明：int cnf\_sudoku(int \*flag,int a[][10])

函数功能：将得到的cnf格式的解翻译为数独的解

函数实现：数独在转换为CNF格式的时候，是将每个位置每个数字都转换为文字，在SAT求解器求解了转换后的CNF范式后，会得到每个位置每个数字的真值，对每个位置找真值为1的数字，即填入该数字。

将每个位置都那样找数字，就可以得到数独的解。

1. 有解数独创建函数

函数声明：void sudoku\_create(int a[][10],int b[][10],FORMULA \*head,VARY \*rela,int \*flag,char\* filename)

函数功能：创建一个一定有解的数独

函数实现：首先sudoku——make函数创建一个数独格局，再将数独格局转换为CNF范式，然后转换SAT问题进行求解，如果有解，那么就生成了一个有解的数独，如果无解，重复上述步骤，直到生成一个有解的数独。将有解的数独的解从CNF格式转换为数独格式，保存在b中。

1. 挖洞判断函数

函数声明：int hole\_judge(int chx,int chy,int a[10][10],int p[10][10])

函数功能：判断该位置是否可以删除空掉

函数实现：如果该位置已经被挖过或者被标记着不可被挖就返回FASLE；

首先将该位置置为0，按照sudoku\_cnf函数将更新后的数独转换为cnf范式格式，此外还需要加一个该位置不能填原来的数字的子句，表示如果该位置不是该数字的解的情况。得到更新后的范式后，将范式用SAT求解器求解，如果有解，那就说明该位置填其他值也有解，就不能保证唯一性，将该位置赋回原本的值，返回FASLE；如果无解，那就说明该位置的只可以填这个值，返回TRUE。

1. 挖洞法函数

函数声明：int dig\_hole(int a[10][10],int b[10][10])

函数功能：挖洞法生成一个只有唯一解的数独

函数实现：定义一个标记二维数组，表示位置是否可以被挖掉，初始化为0。Srand随机一个随机种子，之后再随机一个num，对num进行取余和加一个固定值的操作使num处在一个合法的取值范围中。

随机一个数对(行，列)，调用hole\_judge函数对该位置进行判断是否可以挖去，如果可以挖去，将该位置标记为可挖，反之，标记为不可挖。同时在数独解题格局中对该处赋值为0。重复上述操作，直到挖掉num个可挖的位置。

1. 行列交换得到数独格局函数

函数声明：int sudoku\_new(int a[10][10],int b[10][10])

函数功能：生成一个只有唯一解的数独

函数实现：首先读取一个已知的符合数独规则的数独答案格局。

Srand函数随机一个随机种子，生成一个随机数number，表示交换次数；之后开始进行交换。

随机一个0-2的数，表示交换哪个区域。然后随机两个0-2的数字，表示交换该区域的哪两部分。

行列交叉重复上述步骤，得到一个数独。

1. SAT问题总函数

函数声明：void sat(void)

函数功能：sat算例的求解

函数实现：switch-case结构调用sat部分的函数实现各个功能。

1. 数独总函数

函数声明：void sudoku(void)

函数功能：数独的求解

函数实现：switch-case结构调用数独部分的函数实现各个功能。

**4.2系统测试**

软件测试主要有功能测试、效率测试、边界条件测试等。主要是根据要求设置测试数据，然后根据测试数据对软件的各个方面进行测试。

本次的系统测试主要分为两个模块：SAT求解器测试模块和数独模块。

SAT部分的测试：要求不少于18个SAT算例，其中可满足的算例不少于15个，不满足的算例不少于3个，大中小算例各占三分之一。本次测试将测试大中小分别5个的可满足算例和大中小各一个的不可满足算例。算例规模为：小型算例变元数为100个左右；中型算例变元数介于200-500个； 大型算例变元数600个以上。该部分测试需要包含算例名、算例变元数、子句数与变元数比值、满足还是不满足或不确定、DPLL求解时间(t与to)以及优化率等信息。

数独部分的测试：主要检测数独的生成、数独的解、数独的填写等功能。

**4.2.1 SAT测试**

（1）测试大纲：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算例名 | 算例变元数 | 子句/变元 | 满足YorN | 求解时间/ms | 优化后时间/ms |
| S1 | 25 | 4 | 可满足 | 0 | 9 |
| S2 | 100 | 2 | 可满足 | 19150 | 20506 |
| S3 | 50 | 2 | 可满足 | 59 | 50 |
| S4 | 100 | 3.4 | 可满足 | 2130 | 2129 |
| S5 | 100 | 6 | 可满足 | 69 | 69 |
| M1 | 200 | 1.05 | 可满足 | 0 | 0 |
| M2 | 224 | 7.87 | 可满足 | 49 | 49 |
| M3 | 301 | 9.24 | 可满足 | 9 | 10 |
| M4 | 265 | 21.3 | 可满足 | 52099 | 59107 |
| M5 | 303 | 9.4 | 可满足 | 0 | 0 |
| L500-2 | 500 | 6 | 可满足 | 19348 | 19328 |
| L500-1 | 500 | 6.2 | 可满足 | 120 | 120 |
| L729 | 729 | 30.3 | 可满足 | 90 | 99 |
| L4 | 50073 | 4 | 可满足 | / | / |
| L5 | 1075 | 2.93 | 可满足 | 349163 | 645952 |
| Un60 | 60 | 15.6 | 不可满足 | 201618 | 207931 |
| Un512 | 512 | 18.9 | 不可满足 | 1630 | 1618 |
| Un825 | 825 | 3 | 不可满足 | / | / |

（2）运行结果分析

本次测试，针对S、M、L三个规格的算例进行了各五个可满足算例和各1个不可满足算例测试。

S的五个可满足算例和一个不可满足算例都已成功求解。所测算例都可解。

M的三个可满足算例和一个不可满足算例都已成功求解。所测算例都可解。

L的两个可满足算例和一个不可满足算例都已可以求解。

针对优化，出现了一些负优化现象，此次优化采用的是MOM策略，但是选取的算例在观察了特点之后发现，大多为子句长度较为平均，所以在该策略下并不能很好的得到优化。

（3）SAT运行结果截图

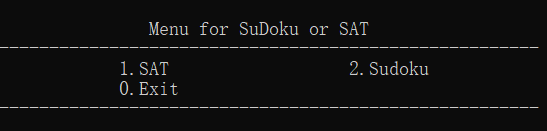


图1 主函数的菜单功能

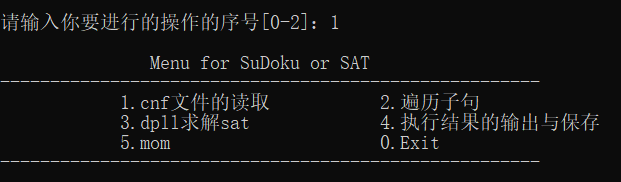


图2 SAT部分的菜单

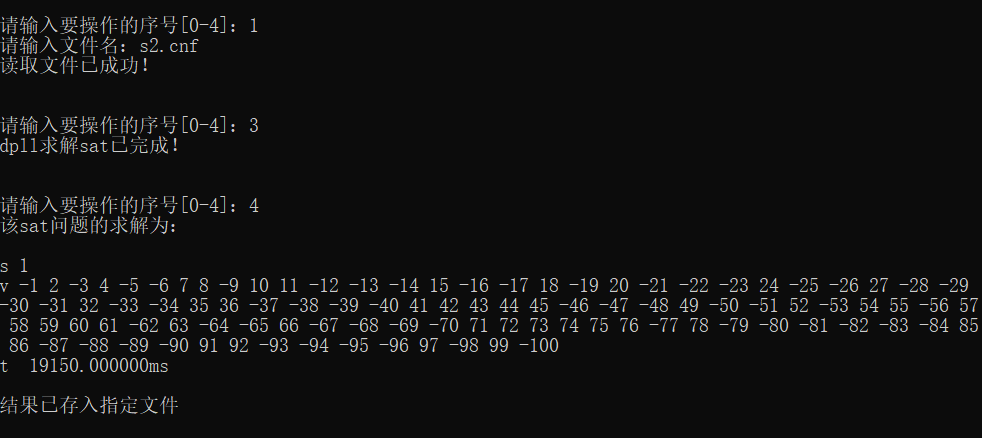


图3对S规模的算例s2的优化前结果

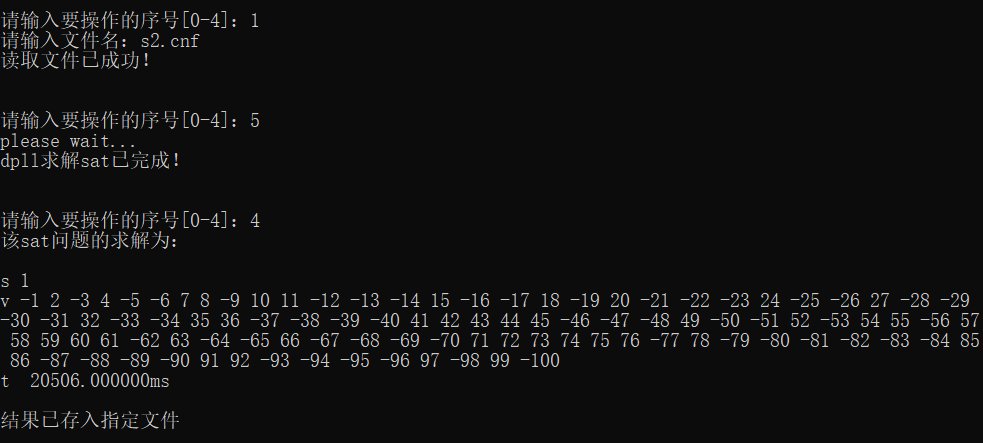


图4对S规模的算例s2的优化后结果

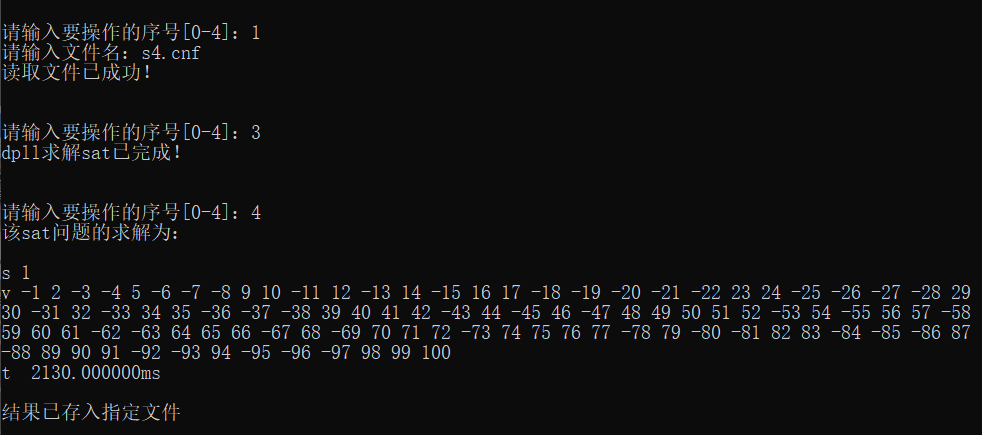


图5 对S规模的算例s4的优化前结果

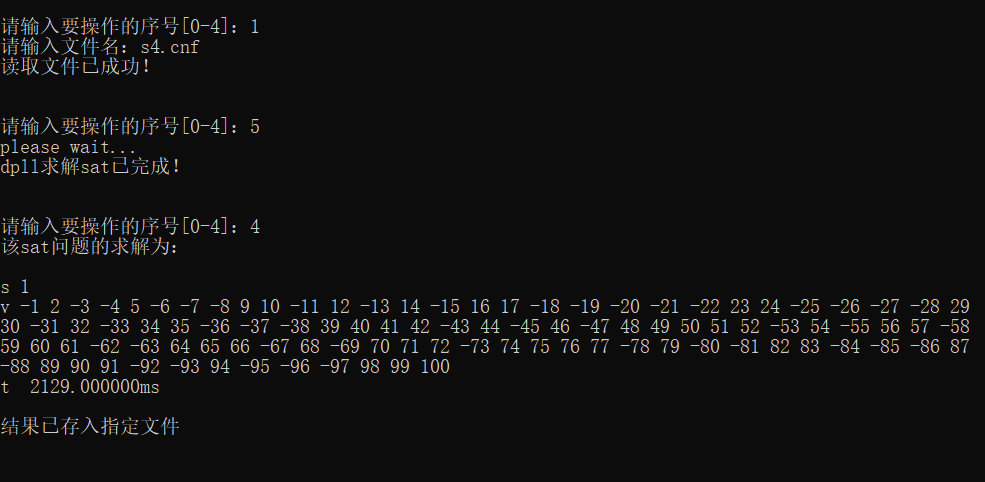


图6 对S规模的算例s4的优化后结果

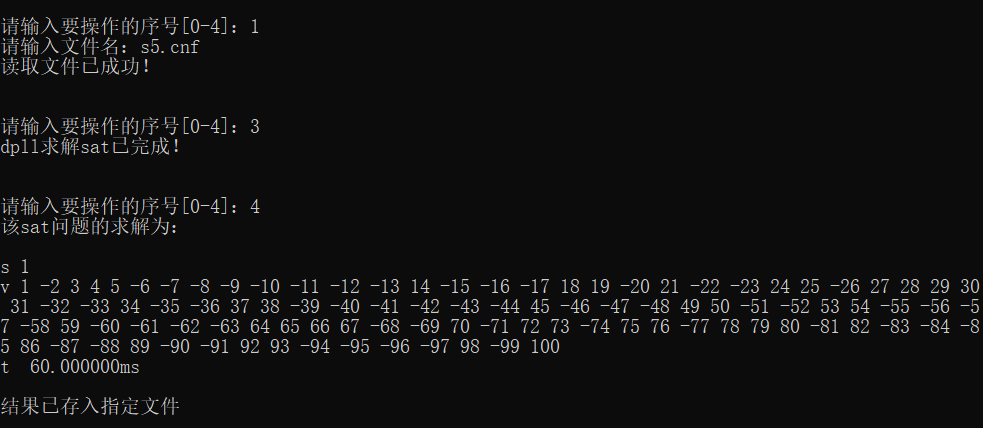


图7 对S规模的算例s5的优化前结果

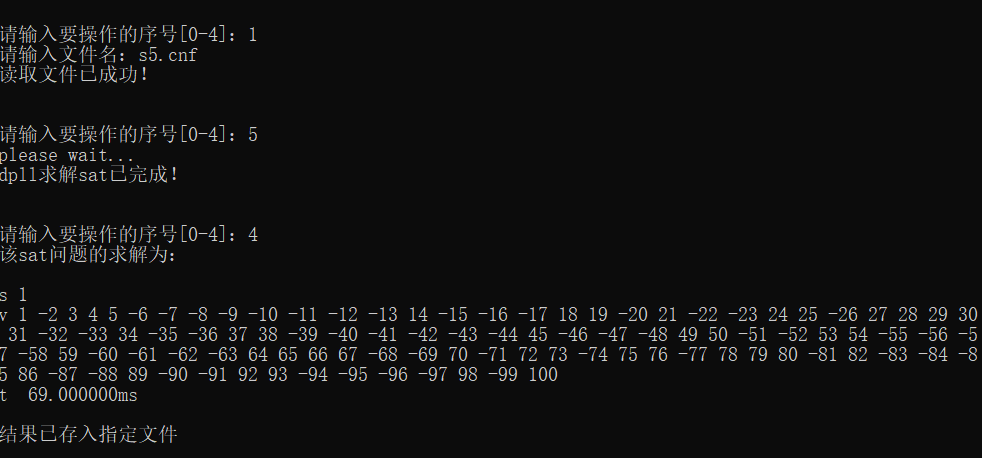


图8 对S规模的算例s5的优化后结果

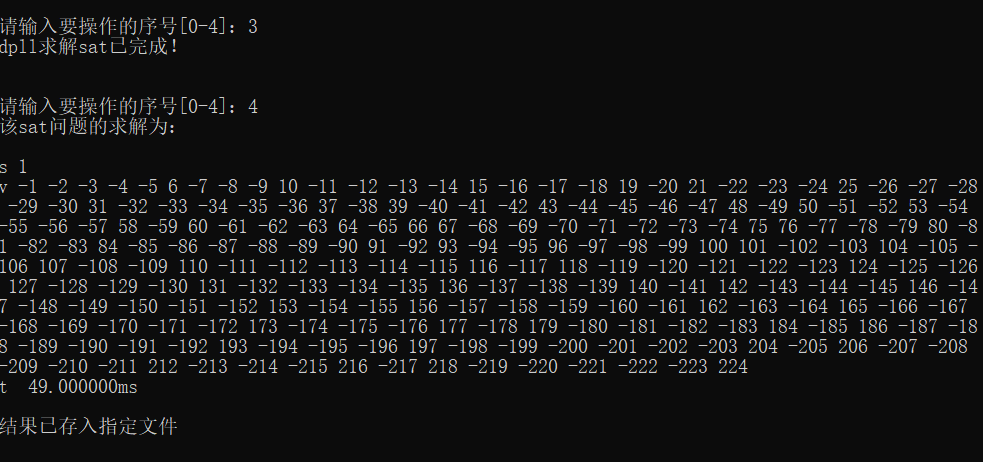


图9对M规模的算例m2的优化前结果

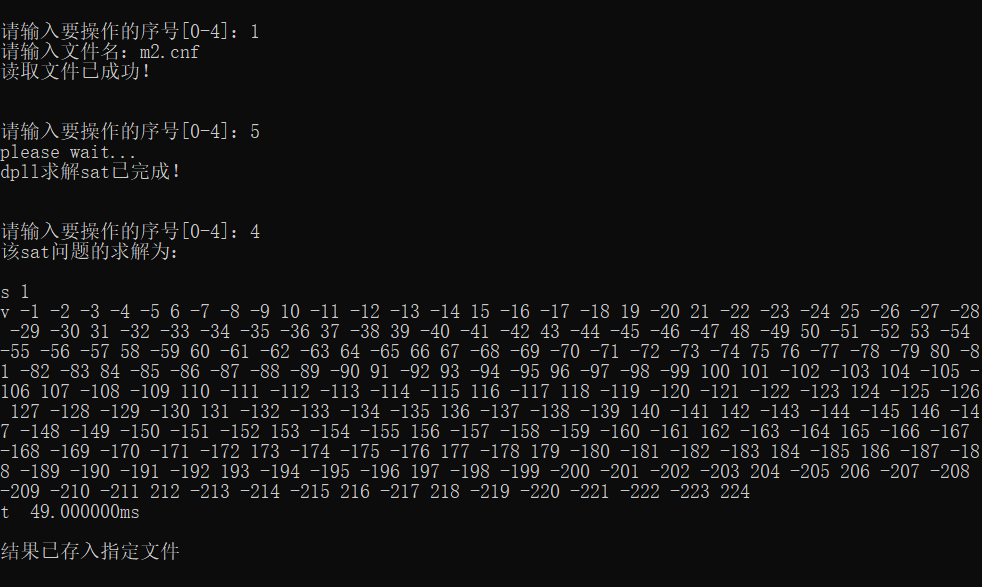


图10对M规模的算例m2的优化后结果

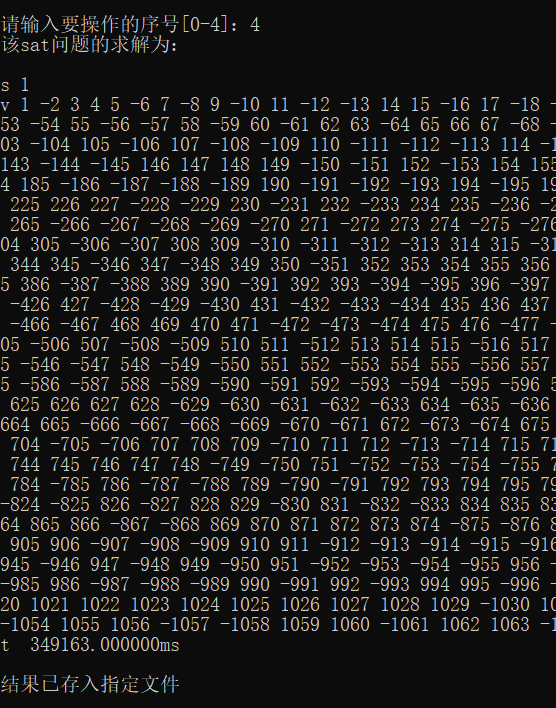


图11对L规模的算例L5的优化前结果

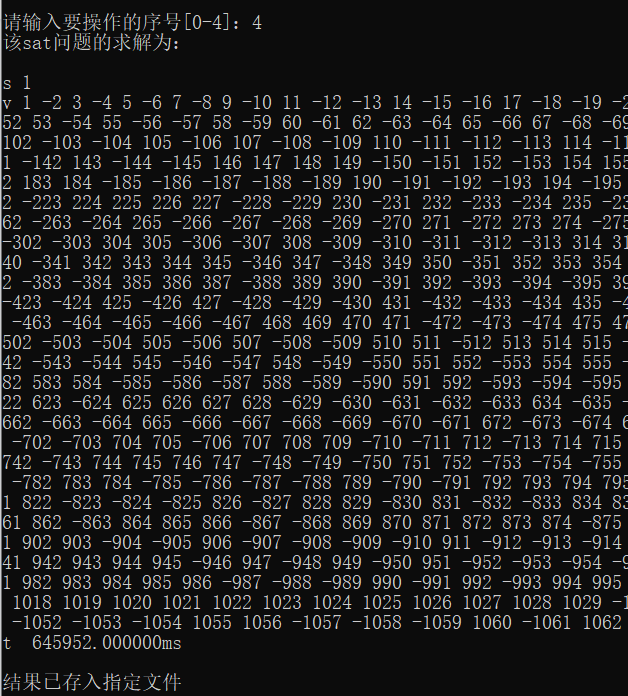


图12对L规模的算例L5的优化后结果

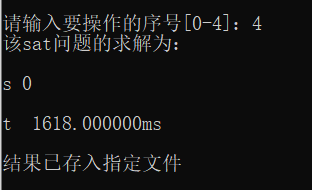
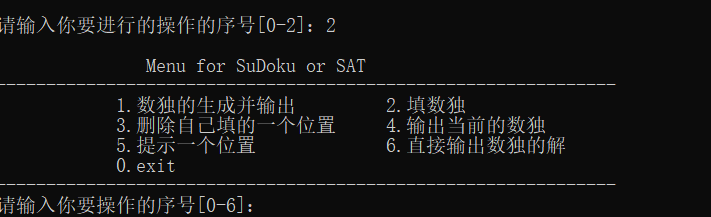


图13对M规模的不可满足算例的优化后结果

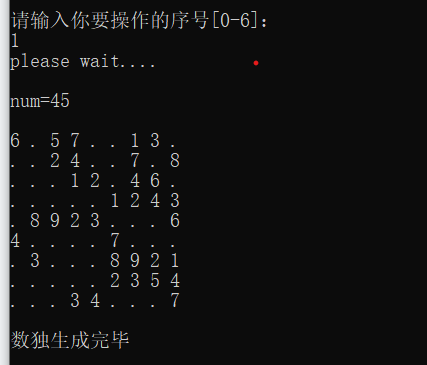
（4）数独部分

在数独部分，主要对数独的各个功能的完整性进行测试，同时在数独的难度等级低、中、高三个方面生成数独。

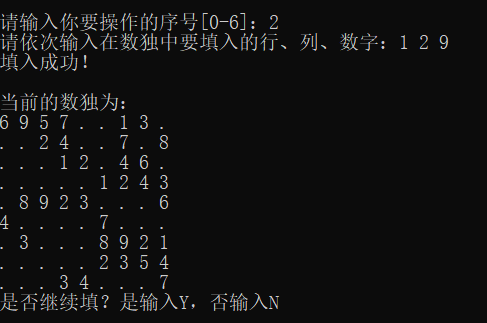
数独菜单：



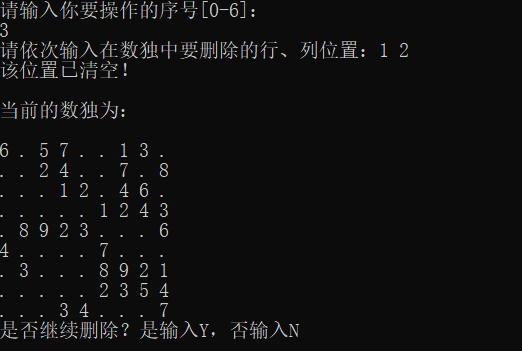
生成一个中度难度的数独：



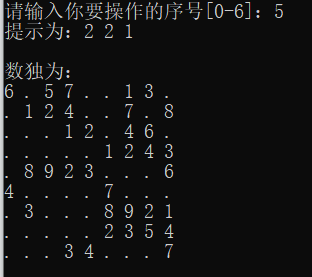
填写数独操作：



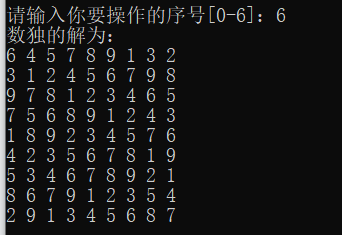
删除自己填的一个位置：



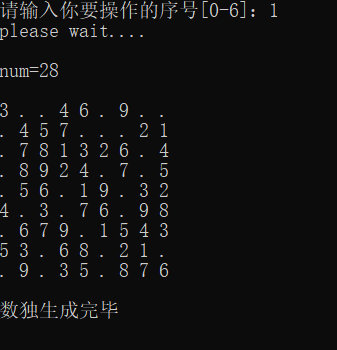
给予提示：



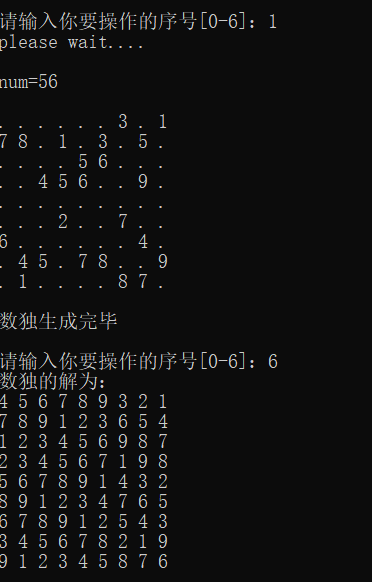
输出数独的解：



生成一个低难度的数独：



生成一个高难度的数独并得出解：



**5总结与展望**

**5.1全文总结**

本次课设所需要完成的主要内容为：SAT问题的基础算法的代码实现、程序优化、SAT问题的实际应用----数独。

SAT求解的代码实现主要使通过DPLL过程、变量决策、范式化简、回溯机制等几个关键步骤实现的。程序优化体现在对数据结构的优化、决策变量决策策略、回溯机制等几个会对时间有影响的部分。数独部分主要包括数独的生成、数独唯一解的保证、数独的解、数独的一般操作等几个方面的代码。主要实现了DPLL算法、挖洞法等基本的方法和思想。此外还有一些小的可以让效率提高的小策略等。

基于上述的要求和基本内容，设计并写出了最终的系统。系统可以实现的功能即为要求中所需求的功能。系统完成后也进行了系统测试，不仅对基础功能进行了测试，还根据不同的需求进行了测试，对边界条件进行了测试，进一步确保程序可以更稳定的运行。

在整个课设工作中，我掌握了SAT问题的一些基础知识和基本的代码实现，了解了SAT的背景和发展历史，知道了我们学校我们学院在此方面做到的成就。学习了数独生成的一些算法。努力做到多种方法实现最终需求。

**5.1工作展望**

虽然此次课设已经结束了，但SAT的研究还没有结束。每次的SAT竞赛都会出现新的更高效率的SAT求解器，所以，目前我们所实现的优化还远远没有达到理想的效率，对于解决实际问题效率还是有些低的。

因此，在之后的工作中，应该在学习了更多的算法的基础上，对算法有了更深入的认识后，将其他更优秀的算法和它相结合，提高效率。

此外，阅读更多的SAT相关书籍才能更深入的了解SAT的优化算法和当前最可能实现的优化方向。

当然，除了SAT的研究，之后还要学习更多的其他课题。虽然这次没有去做另外一个课设，但也应该是我们所要完成的一项工作。

# 6体会

本次的课设已经结束了。既是开心又是略有遗憾。

在一开始写这个程序的时候，我一开始是头懵且没有思路的，数据结构的建立没有思路使我头秃。终于在看了几篇文章之后我建立好了我第一个数据结构，并且二话不说，直接根据指导报告的伪代码实现了递归算法。然后我就开启了我痛苦的一周……我花了一周时间去放弃了我的第一个代码。

子句化简的粗暴和递归的暴力使用，使我的程序卡死了，一个都运行不出来，即使是最小的算例。我反复的对我的代码进行调试和debug，每天都铺在上面，但众所周知，当你的代码陷入了沼泽的时候，最好的方法是重新写一个，但当时不想放弃我写的第一遍的代码，我反复地debug，花了四五天时间，找出来了一个错误，但是还是运行不了，当一周过去了，我终于决定放弃原来的代码了。

之后又学习其他的化简规则和改进算法，在修改了两遍数据结构，重新写了三遍代码之后，代码终于没有报错并且运行出来了第一个算例。

之后开始写数独部分、对算法优化，每一次都要重新写好几遍才能达到最终的效果。但好在效果还不错，S、M都跑出来了，L也跑出来了几个。算是很开心的部分。

但是这次的课设并没有做的很完美，特别是优化部分，在我心里像一个疙瘩一样。可能之后再写写优化吧，不过最近很忙，课设又结束了，应该又该“封尘”了。也算遗憾吧。

但是实话讲，这课设确实让人头秃，看了一天又一天，看了一夜又一夜，终于搞出来了一版代码，但是又要推翻……反反复复。着实让人头秃。中间的国庆假期更是将原来的代码进度拉下来好多，太让人头秃了，太让人头秃了。

但是静下来想想，反思了一下。这次课设其实并不算很难，是可以在能力范围内实现的，但是却耗费了这么久的时间，这么大的精力。现在结束了，回头看，一方面是因为自己前段时间太浮躁了，刚开学的事情一件又一件，又加上有情绪上的不开心，导致在学习的时候很浮躁，静不下心来，这或许不应该是真正学习的人的状态；另一方面是自己的算法和代码能力，还未达到一个合格的大二学生的水平，这方面自己应该花更多的时间和精力去提高！

在发完牢骚之后，也该真正的去想自己到底在课设的过程中做了什么和学到了什么吧。

做了DPLL……这些话在前面已经说了好几遍了。我只能说我实现了基础的要求。但是我做的远远没有达到我自己的预期。最明显的就是优化了，而且自己实现的其实还是负优化。这方面做了，但还是做的不够。

至于学到了什么，就是知识和打击吧。在一遍又一遍的失败中，肯定积累了一些经验，也可以用到其他的算法中，比如标记法代替真正的更改、迭代代替递归、利用栈高效回溯等等的经验。

报告模板中说这一部分要写具体的项目设计和自己在课设中的经验，但项目设计在前面已经写过了几遍了，自以为在这里写就太水字数了。比起模板中的要求，这个更像是半夜给课设收尾的心情。

至此课设课程算是结束了，新的任务又来了。

**附录 代码**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

#define ElemType int

#define maxsize 1005

#define SUCCESS 2

#define INFEASIBLE -1

#define TRUE 1

#define FASLE 0

#define OUT -2

#define ERROR 0

#define OK 1

typedef struct letter

{

int flag;

ElemType prop;//变量文字

struct letter \*next; //子句的下一个文字

}LETTER,\*LETS;

//式子中的变量文字 结构体

typedef struct formula

{

int flag;//标志该子句是否已经被处理

int pos;//子句所在位置

int number;//一个子句中的变量文字的数目

LETTER \*headp;//一个子句中的变量文字头结点

struct formula \*next;//下一个子句

int satisfied;

int unsatisfied;

}FORMULA,\*FORM;

//子句结构体

typedef struct variable

{

ElemType key;//文字

ElemType \*prop;//逻辑变量的栈

int number;//与该变量文字有关的子句的数目

int maxlength;//最多存放

LETS \*lets;

FORM \*s;//存放上述子句的栈

}VARY;

//与变量文字有关的所有子句构成的结构体

double score[1000];

void getscore(FORMULA \*head)

//为每个变量计分

{

FORMULA \*cur;

LETTER \*var;

cur=head->next;

int key,p,i;

double sym;

while(cur)

{

p=1;

for(i=1;i<=cur->number;i++) p\*=2;

sym=1/p;

var=cur->headp->next;

while(var)

{

key=var->prop;

if(key<0) key\*=(-1);

score[key]+=sym;

var=var->next;

}

cur=cur->next;

}

}

int get\_key\_mom(FORMULA \*head,int \*flag,int num)

//mom法获得决策变量

{

int i;

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

if(!flag[i]) break;

if(i>num) return 0;

int min=i;

for(i++;i<=num;i++)

if(!flag[i]&&score[i]>score[min]) min=i;

return min;

}

FORMULA\* cnf\_read(char \*filename,int \*num1,int \*num2)

//读取名称为fiename的cnf格式文件，将范式存放在链表中

{

char anno;

FORMULA \*head;//头结点

int var\_num,form\_num,i,p;

FILE \*file;

file=fopen(filename,"r");

if(file==NULL) return NULL;

fscanf(file,"%c",&anno);

while(anno!='p')//读取注释

{

fscanf(file,"%c",&anno);

while(anno!='\n') fscanf(file,"%c",&anno);//读了一行

fscanf(file,"%c",&anno);

}//while

for(;!(anno<='9'&&anno>='0');i++)

fscanf(file,"%c",&anno);//把文字读取掉

ungetc(anno,file);//将输出的数字放回输入流

fscanf(file,"%d",&var\_num);//文字的数目

fscanf(file,"%d",&form\_num);//子句数目

head=(FORMULA\*)malloc(sizeof(FORMULA));//头结点

head->pos=0;

head->headp=NULL;

head->next=NULL;

FORMULA \*cur\_form=NULL,\*pre\_form=head;

for(i=1;i<=form\_num;i++)//读取cnf文件，保存在链表里面

{

pre\_form->next=(FORMULA\*)malloc(sizeof(FORMULA));

cur\_form=pre\_form->next;

cur\_form->next=NULL;

cur\_form->pos=pre\_form->pos+1;

cur\_form->flag=0;

cur\_form->number=cur\_form->satisfied=cur\_form->unsatisfied=0;//初始化该子句

LETTER \*cur\_let,\*pre\_let;

cur\_form->headp=(LETTER\*)malloc(sizeof(LETTER));

pre\_let=cur\_form->headp;//子句里的头文字结点

pre\_let->next=NULL;

fscanf(file,"%d",&p);

while(p)//析取范式以0结束

{

cur\_form->number++;

pre\_let->next=(LETTER\*)malloc(sizeof(LETTER));

cur\_let=pre\_let->next;

cur\_let->prop=p;

cur\_let->next=NULL;

cur\_let->flag=0;

pre\_let=cur\_let;

cur\_let=pre\_let->next;

fscanf(file,"%d",&p);

}//while

pre\_form=cur\_form;

cur\_form=pre\_form->next;

}//for

fclose(file);//关闭文件

\*num1=var\_num;\*num2=form\_num;

return head;

}//cnf\_read

VARY\* RelaCreate(FORMULA \*head,int num)

//根据范式将与每个变量文字有关的子句联系起来

//将与某文字有关的子句都存放到指定的栈中

{

if(num==0) return NULL;

VARY \*flag;

flag=(VARY\*)malloc(sizeof(VARY)\*(num+1));//每一个文字都有对应的结构体

int i;

for(i=1;i<=num;i++)//初始化结构体

{

flag[i].key=i;

flag[i].number=0;

flag[i].prop=(ElemType\*)malloc(sizeof(ElemType)\*maxsize);

flag[i].s=(FORM\*)malloc(sizeof(FORM)\*maxsize);

flag[i].lets=(LETS\*)malloc(sizeof(LETS)\*maxsize);

flag[i].maxlength=maxsize;

}

FORMULA \*cur=head->next;

LETTER \*var;

ElemType key,temp;

int top;

while(cur)//遍历整个范式

{

var=cur->headp;

var=var->next;

while(var)//遍历整个子句

{

key=var->prop;

if(key>0) temp=key;

else temp=key\*(-1);//找到该逻辑文字对应的变量文字

top=flag[temp].number;

if(top>=flag[temp].maxlength)//如果长度超出原分配空间

{

flag[temp].prop=(ElemType\*)realloc(flag[temp].prop,top+10);

flag[temp].s=(FORM\*)realloc(flag[temp].s,top+10);

flag[temp].lets=(LETS\*)realloc(flag[temp].lets,top+10);

flag[temp].maxlength=top+10;

}

flag[temp].prop[top]=key;

flag[temp].s[top]=cur;

flag[temp].lets[top]=var;

flag[temp].number++;//记录与该文字有关的逻辑文字，子句

var=var->next;

}

cur=cur->next;

}

return flag;

}

void cnftraverse(FORMULA \*head)

//遍历建立的范式 输出

{

FORMULA \*cur;

LETTER \*var;

cur=head->next;

while(cur)

{

printf("子句%d为：",cur->pos);

var=cur->headp->next;

while(var)

{

printf("%d ",var->prop);

var=var->next;

}

printf("\n");

cur=cur->next;

}

}

int get\_singlekey(FORMULA \*head)

//函数作用：

//从单子句中找文字

{

FORMULA \*cur=head->next;

int key=0;

while(cur)

{

if(cur->flag);//该子句已经可满足

else if(cur->unsatisfied==cur->number-1)//单子句

{

LETTER \*var;

var=cur->headp->next;

while(var)//找出唯一一个没确定的

{

if(var->flag);

else

{

key=var->prop;

break;

}

var=var->next;

}

break;

}//找到了单子句

cur=cur->next;

}

return key;//没找到的话返回0

}

void cnfsimply(FORMULA \*head,ElemType key,int \*flag,VARY \*rela)

//根据已经确定的变量真值对范式进行化简

{

int x,num,i;

if(key>0)

{

x=key;

flag[x]=1;//该文字为真

}

else

{

x=key\*(-1);

flag[x]=-1;//该文字为假

}

num=rela[x].number;//对与改文字有关的句子进行遍历

FORMULA \*cur;

LETTER \*let;

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

{

cur=rela[x].s[i];//子句所在位置

let=rela[x].lets[i];//该文字所在位置

if(rela[x].prop[i]==key)//此子句的变量与指定的真值一致

{

cur->flag=1;//该子句真值取1

cur->satisfied++;//子句中真值为真的文字加一

}

else//此子句的变量与指定的真值相反

{

cur->unsatisfied++;//子句中的文字不满足加一

let->flag=-1;//对该文字标记为假

}

}

}

void cnfrestore(FORMULA \*head,int \*s,int start,int end,VARY \*rela,int \*flag)

//对栈中的需要回溯的一部分，恢复回溯点时的状态

{

int x,i,j,num;

FORMULA \*cur;

LETTER \*let;

for(i=start;i<end;i++)//对回溯点到栈顶的变量都遍历一遍

{

if(s[i]>0) x=s[i];

else x=s[i]\*(-1);//找到对应的文字

flag[x]=0;//回复真值的不确定性

num=rela[x].number;

for(j=0;j<num;j++)//与该文字有关的子句进行回复原来的状态

{

cur=rela[x].s[j];//子句

let=rela[x].lets[j];//变量

if(rela[x].prop[j]==s[i])//变量与指定的一致

{

cur->satisfied--;//可满足的加一，恢复原样

if(cur->satisfied==0) cur->flag=0;//如果真值为真的文字为0，子句的真值恢复为0

}//if

else//变量与指定的相反

{

cur->unsatisfied--;//真值为假的文字减一

let->flag=0;//该文字的标志恢复为0，不确定

}//else

}//for

}//for

}

int get\_key(int \*flag,int num,VARY \*a)

//选取决策变量

{

int i;

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

if(flag[i]==0&&a[i].number) return i;

return 0;

}

int dpllcheck(FORMULA \*head)

//检查当前范式的状态，真值以确定/不可满足/尚未确定

{

FORMULA \*cur=head->next;

while(cur)

{

if(cur->unsatisfied==cur->number)//子句中的文字全部为假

return INFEASIBLE;//不可满足

cur=cur->next;

}

cur=head->next;

while(cur)

{

if(cur->flag==0) return TRUE;//有子句的真值尚未确定

cur=cur->next;

}

return SUCCESS;//所有子句为真

}

int dpll(FORMULA \*head,int \*flag,int num,VARY \*rela)

//dpll求解范式真值的过程

{

FORMULA \*cur=head->next;

ElemType key=0,judge;

int \*s,top\_s=0,\*record,top\_record=0;

s=(int\*)malloc(sizeof(int)\*(num+1));

record=(int\*)malloc(sizeof(int)\*num);

LETTER \*var;

key=get\_singlekey(head);//从cnf范式中找单子句找到对应的文字

do

{

while(key)//找得到可以用来化简的文字

{

s[top\_s++]=key;//会被处理的文字入栈

cnfsimply(head,key,flag,rela);//利用该文字对cnf范式化简

key=get\_singlekey(head);//找下一个单子句中的文字

}

judge=dpllcheck(head);//判断是否已经完成

if(judge==SUCCESS) return TRUE;//已经全部成立

else if(judge==INFEASIBLE)//当前的取真值的方式，该范式不可满足

{

if(top\_record==0) return FASLE;//回溯至头处真值仍为假，此时cnf范式是不可满足的

else//还没回溯到头

{

top\_record--;//找回溯点

cnfrestore(head,s,record[top\_record],top\_s,rela,flag);//对被回溯的那一部分进行范式恢复

key=s[record[top\_record]]\*(-1);//更改回溯处的文字的真值取值

top\_s=record[top\_record];//出栈被回溯的部分

}

}

else//当前的取值还未发现矛盾，但是整个范式的真值未完全确定

{

record[top\_record++]=top\_s;//记录回溯点，入栈

key=get\_key(flag,num,rela);//找决策变量

}

}while(key);//直到所有的变量值都已经被确定

}

int dpll\_mom(FORMULA \*head,int \*flag,int num,VARY \*rela)

//dpll求解范式真值的过程

{

FORMULA \*cur=head->next;

ElemType key=0,judge;

int \*s,top\_s=0,\*record,top\_record=0;

s=(int\*)malloc(sizeof(int)\*(num+1));

record=(int\*)malloc(sizeof(int)\*num);

LETTER \*var;

getscore(head);

key=get\_singlekey(head);//从cnf范式中找单子句找到对应的文字

bool Judge=0;

do

{

while(key)//找得到可以用来化简的文字

{

if(!Judge) Judge=1,printf("please wait...\n");

s[top\_s++]=key;//会被处理的文字入栈

cnfsimply(head,key,flag,rela);//利用该文字对cnf范式化简

key=get\_singlekey(head);//找下一个单子句中的文字

}

judge=dpllcheck(head);//判断是否已经完成

if(judge==SUCCESS) return TRUE;//已经全部成立

else if(judge==INFEASIBLE)//当前的取真值的方式，该范式不可满足

{

if(top\_record==0) return FASLE;//回溯至头处真值仍为假，此时cnf范式是不可满足的

else//还没回溯到头

{

top\_record--;//找回溯点

cnfrestore(head,s,record[top\_record],top\_s,rela,flag);//对被回溯的那一部分进行范式恢复

key=s[record[top\_record]]\*(-1);//更改回溯处的文字的真值取值

top\_s=record[top\_record];//出栈被回溯的部分

}

}

else//当前的取值还未发现矛盾，但是整个范式的真值未完全确定

{

record[top\_record++]=top\_s;//记录回溯点，入栈

key=get\_key\_mom(head,flag,num);//找决策变量

}

}while(key);//直到所有的变量值都已经被确定

return OK;

}

int resprintf(char \*filename,int judge,int \*flag,int num,int period)

//将结果写入指定文件

{

int i;

char newfile[100];

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

newfile[i]=filename[i];

newfile[i++]='.';

newfile[i++]='r';

newfile[i++]='e';

newfile[i++]='s';

FILE \*file;

file=fopen(newfile,"w");

if(file==NULL) return ERROR;

if(judge==FASLE)

{

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

fprintf(file,"%d ms",period);

}

else if(judge==OUT)

{

fprintf(file,"s -1\n");

}

else

{

int i;

fprintf(file,"s 0\nv");

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

{

if(flag[i]==-1)

fprintf(file," %d",i\*flag[i]);

else

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

}

fprintf(file,"\nt %d",period);

}

fclose(file);

return OK;

}

int sudoku\_judge(int a[10][10],int i,int j,int number)

//判断当前的数独是否合法

{

int k,p;

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

if(a[i][k]==number) return ERROR;

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

if(a[k][j]==number) return ERROR;

int x,y;

x=(i/3)\*3;

y=(j/3)\*3;

for(k=x;k<x+3;k++)

for(p=y;p<y+3;p++)

if(a[k][p]==number) return ERROR;

return OK;

}

int sudoku\_make(int a[10][10])

//生成一个数独

{

srand((unsigned)time(NULL));

int sum;

sum=rand()%11;

sum+=21;

int col,lin,fig,cnt=0;

while(cnt<sum)

{

col=rand()%9;//行

lin=rand()%9;//列

fig=rand()%9+1;//该位置的数

if(a[col][lin]==0&&sudoku\_judge(a,col,lin,fig)==OK)

{

a[col][lin]=fig;

cnt++;

}

}

return OK;

}

void sudoku\_print(int a[][10])

//输出数独

{

int i,j;

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

{

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

{

if(a[i][j]!=0) printf("%d ",a[i][j]);

else printf(". ");

}

printf("\n");

}

}

int sudoku\_cnf(int \*flag,int a[][10],char \*filename)

//将数独转换成cnf格式并保存至文件

{

int i,j,figure,var,k,x,y;

int p,col,lin;

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

{

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

{

if(a[i][j]!=0)//该位置的值已经确定了

{

figure=a[i][j];//该位置的值

for(k=0;k<9;k++)//将与该位置的同一行的位置标记不可以填figure

{

var=i\*100+k\*10+figure;

flag[var]=-1;

}

for(k=0;k<9;k++)//将与该位置的同一列的位置标记不可以填figure

{

var=k\*100+j\*10+figure;

flag[var]=-1;

}

col=(i/3)\*3+3;

lin=(j/3)\*3+3;

for(k=(i/3)\*3;k<col;k++)//将与该位置的同一块的位置标记不可以填figure

for(p=(j/3)\*3;p<lin;p++)

{

var=k\*100+p\*10+figure;

flag[var]=-1;

}

for(k=1;k<=9;k++)//该位置不可以为其他值

{

var=i\*100+j\*10+k;

flag[var]=-1;

}

var=i\*100+j\*10+figure;

flag[var]=1;//该处标记为figure

}

}//for j

}//for i

int sum=36\*81+81\*3,v1,v2;//表示每一个位置只能放一个值和每一行、列、块都有1-9的句子数

for(k=1;k<=889;k++)//加上单子句数目

if(flag[k]) sum++;

FILE \*file;

file=fopen(filename,"w");

if(file==NULL) return ERROR;

fprintf(file,"p cnf 889 %d\n",sum);//变元数为889，句子数sum

for(k=1;k<=889;k++)//先写入单子句

if(flag[k])

fprintf(file,"%d 0\n",flag[k]\*k);

for(i=0;i<9;i++)//写入每一个位置只能填一个值

{

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

{

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

{

v1=(i\*100+j\*10+x)\*(-1);

for(y=x+1;y<=9;y++)

{

v2=(i\*100+j\*10+y)\*(-1);

fprintf(file,"%d %d 0\n",v1,v2);

}//for y

}//for x

}//for j

}//for i

for(i=0;i<9;i++)//每一行都有1-9

{

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

{

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

{

var=i\*100+j\*10+k;

fprintf(file,"%d ",var);

}//for j

fprintf(file,"0\n");

}//for k

}//for i

for(j=0;j<9;j++)//每一列都有1-9

{

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

{

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

{

var=i\*100+j\*10+k;

fprintf(file,"%d ",var);

}//for i

fprintf(file,"0\n");

}//for k

}//for j

for(i=0;i<9;i+=3)//每一块都有1-9

{

for(j=0;j<9;j+=3)

{

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

{

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

{

for(y=j;y<j+3;y++)

{

var=x\*100+y\*10+k;

fprintf(file,"%d ",var);

}//for y

}//for x

fprintf(file,"0\n");

}//for k

}//for j

}//for i

fclose(file);

return OK;

}

int cnf\_sudoku(int \*flag,int a[][10])

//将得到的cnf格式的解翻译为数独的解

{

int i,j,x,var;

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

{

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

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

{

var=i\*100+j\*10+x;

if(flag[var]==1)

{

a[i][j]=x;

break;

}

}

}

return OK;

}

void sudoku\_create(int a[][10],int b[][10],FORMULA \*head,VARY \*rela,int \*flag,char\* filename)

//创建一个一定有解的数独

{

int judge=FASLE,i,j,var\_num,prop\_num,\*num1=&var\_num,\*num2=&prop\_num;

while(judge==FASLE)

{

for(i=0;i<10;i++)

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

a[i][j]=0;

for(i=0;i<1000;i++) flag[i]=0;

sudoku\_make(a);

sudoku\_cnf(flag,a,filename);

head=cnf\_read(filename,num1,num2);

rela=RelaCreate(head,var\_num);

judge=dpll(head,flag,var\_num,rela);

}

for(i=0;i<10;i++)

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

b[i][j]=0;

cnf\_sudoku(flag,b);

}

int hole\_judge(int chx,int chy,int a[10][10],int p[10][10])

//判断该位置是否可以删除空掉

{

if(a[chx][chy]==0) return ERROR;//该位置已经被删除了

if(p[chx][chy]!=0) return ERROR;

int flag[10],i,\*sym;

sym=(int\*)malloc(sizeof(int)\*1000);

for(i=0;i<1000;i++) sym[i]=0;

for(i=0;i<10;i++) flag[i]=1;

int key=a[chx][chy];

flag[key]=0;

int j,k,m;

i=chx;

for(j=0;j<9;j++) flag[a[i][j]]=0;

i=chy;

for(j=0;j<9;j++) flag[a[j][i]]=0;

i=chx/3;i\*=3;

j=chy/3;j\*=3;

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

for(m=j;m<j+3;m++)

flag[a[k][m]]=0;

FORM head;

VARY \*rela;

char \*filename;

int judge,var\_num,prop\_num,\*num1=&var\_num,\*num2=&prop\_num;

filename=(char\*)malloc(100);

strcpy(filename,"sud.txt");

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

if(flag[i])

{

//printf("i=%d\n\n",i);

a[chx][chy]=i;

sudoku\_cnf(sym,a,filename);

head=cnf\_read(filename,num1,num2);

rela=RelaCreate(head,var\_num);

//printf("before\n\n");

judge=dpll(head,sym,var\_num,rela);

if(judge==TRUE)

{

a[chx][chy]=key;

return ERROR;

}

}

a[chx][chy]=key;

return OK;

}

int dig\_hole(int a[10][10],int b[10][10])

//挖洞法生成一个只有唯一解的数独

{

int num,p[10][10];

srand((unsigned)time(NULL));

num=rand()%40;

num+=20;//挖洞数

//num=30;

int i,j,col,lin,judge;

for(i=0;i<10;i++)

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

a[i][j]=b[i][j];//将b复制到a中去

for(i=0;i<10;i++)

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

p[i][j]=0;//标记某个位置是否可以被删除

printf("num=%d\n\n",num);

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

{

do

{

col=rand()%9;//生成行

lin=rand()%9;//生成列

judge=hole\_judge(col,lin,a,p);

if(judge==ERROR) p[col][lin]=-1;//该位置不能被删除

}while(judge==ERROR);//如果该位置不能被删除继续

a[col][lin]=0;//该位置被删除

p[col][lin]=1;//该位置被删除

}

}

int sudoku\_new(int a[10][10],int b[10][10])

//生成一个只有唯一解的数独

{

printf("please wait....\n\n");

int i,j,number,cnt;

char filename[100]="sudu.txt";

FILE \*file;

file=fopen(filename,"r");

if(!file) return ERROR;

for(i=0;i<9;i++)//读取一个完整的简单数独

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

{

fscanf(file,"%d",&b[i][j]);

}

fclose(file);

srand((unsigned)time(NULL));

number=rand()%30;

for(cnt=0;cnt<number;cnt++)//随机交换行列

{

if(cnt%2)//交换行

{

i=rand()%3;

int lin1,lin2;

lin1=rand()%3;

lin2=rand()%3;

if(lin1!=lin2)

{

i\*=3;

lin1+=i;

lin2+=i;

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

{

b[9][j]=b[lin1][j];

b[lin1][j]=b[lin2][j];

b[lin2][j]=b[9][j];

}

}

}

else//交换列

{

j=rand()%3;

int lin1,lin2;

lin1=rand()%3;

lin2=rand()%3;

if(lin1!=lin2)

{

j\*=3;

lin1+=j;

lin2+=j;

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

{

b[i][9]=b[i][lin1];

b[i][lin1]=b[i][lin2];

b[i][lin2]=b[i][9];

}

}

}

}

dig\_hole(a,b);

return OK;

}

void sat(void)

//sat算例的求解

{

printf("\n Menu for SuDoku or SAT \n");

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

printf(" 1.cnf文件的读取 2.遍历子句 \n");

printf(" 3.dpll求解sat 4.执行结果的输出与保存 \n");

printf(" 5.mom 0.Exit \n");

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

char filename[100];

FORMULA \*head=NULL;

VARY \*rela=NULL;

int var\_num,prop\_num,\*num1=&var\_num,\*num2=&prop\_num;

int \*flag,i,judge;

float t;

clock\_t start,end;

int op=1;

while(op)

{

printf("\n请输入要操作的序号[0-4]：");

scanf("%d",&op);

switch(op)

{

case 1:

printf("请输入文件名：");

scanf("%s",filename);

head=cnf\_read(filename,num1,num2);

if(!head) printf("读取cnf失败！\n");

else printf("读取文件已成功！\n");

rela=RelaCreate(head,var\_num);

if(!rela) printf("创建文字结构体失败！\n");

flag=(int\*)malloc(sizeof(int)\*(var\_num+1));

for(i=0;i<=var\_num;i++) flag[i]=0;

getchar();getchar();

break;

case 2:

if(head)

{

printf("该范式的子句为：\n\n");

cnftraverse(head);

}

else printf("尚未读取文件!\n");

getchar();getchar();

break;

case 3:

start=clock();

judge=dpll(head,flag,var\_num,rela);

end=clock();

t=end-start;

printf("dpll求解sat已完成！\n");

getchar();getchar();

break;

case 4:

printf("该sat问题的求解为：\n\n");

if(judge)

{

printf("s 1\nv");

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

{

printf(" %d",i\*flag[i]);

}

}

else printf("s 0\n");

printf("\nt %fms\n",t);

if(resprintf(filename,judge,flag,var\_num,t)==OK) printf("\n结果已存入指定文件\n");

else printf("\n结果保存文件失败\n");

getchar();getchar();

break;

case 5:

start=clock();

judge=dpll\_mom(head,flag,var\_num,rela);

end=clock();

t=end-start;

printf("dpll求解sat已完成！\n");

getchar();getchar();

break;

case 0:

printf("sat问题解决完毕！\n");

getchar();getchar();

break;

}//switch op

}//while op

}

void sudoku(void)

//数独的求解

{

printf("\n Menu for SuDoku or SAT \n");

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

printf(" 1.数独的生成并输出 2.填数独 \n");

printf(" 3.删除自己填的一个位置 4.输出当前的数独 \n");

printf(" 5.提示一个位置 6.直接输出数独的解 \n");

printf(" 0.exit \n");

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

char \*filename;

filename=(char\*)malloc(sizeof(char)\*100);

strcpy(filename,"shudu.txt");

FORMULA \*head;

VARY \*rela;

int var\_num,prop\_num,\*num1=&var\_num,\*num2=&prop\_num;

int \*flag,judge;

int op=1;

int a[10][10],b[10][10],p[10][10],i,j;

flag=(int\*)malloc(sizeof(int)\*1000);

for(i=0;i<1000;i++) flag[i]=0;

for(i=0;i<10;i++)

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

a[i][j]=0;

int col,lin,number;

char c;

while(op)

{

printf("请输入你要操作的序号[0-6]：");

scanf("%d",&op);

switch(op)

{

case 1:

//printf("生成的数独为：\n\n");

sudoku\_new(a,b);

sudoku\_print(a);

for(i=0;i<10;i++)

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

{

if(a[i][j]) p[i][j]=1;

else p[i][j]=0;

}

printf("\n数独生成完毕\n");

getchar();getchar();

break;

case 2:

do

{

printf("请依次输入在数独中要填入的行、列、数字：");

scanf("%d",&col);

scanf("%d",&lin);

scanf("%d",&number);

col--;

lin--;

if(0<=col&&col<9&&lin>=0&&lin<9&&number>0&&number<10)

{

if(p[col][lin]) printf("该位置为固定值，不可更改！\n\n");

else if(a[col][lin])

{

printf("该位置已填，是否要更改？是输入Y，否输入N。\n");

scanf("%c",&c);

if(c=='Y')

{

a[col][lin]=number;

printf("该位置数字已更改！\n\n");

}

else printf("该位置数字已报留！\n");

}

else

{

judge=sudoku\_judge(a,col,lin,number);

if(judge==ERROR) printf("tip:该数值不符合规则，建议换一个数值尝试\n\n");

else printf("填入成功！\n\n");

a[col][lin]=number;

}

}

else printf("填入的位置或者数值不合法！请输入三个1-9的数！\n\n\n");

printf("当前的数独为：\n");

sudoku\_print(a);

printf("是否继续填？是输入Y，否输入N\n");

scanf("%c",&c);

}while(c=='Y');

getchar();getchar();

break;

case 3:

do

{

printf("请依次输入在数独中要删除的行、列位置：");

scanf("%d",&col);

scanf("%d",&lin);

col--;

lin--;

if(0<=col&&col<9&&lin>=0&&lin<9)

{

if(p[col][lin]) printf("该位置为固定值，不可更改！\n\n");

else

{

a[col][lin]=0;

printf("该位置已清空！\n\n");

}

}

else printf("输入的位置不合法！请输入两个1-9的数！\n\n");

printf("当前的数独为：\n\n");

sudoku\_print(a);

printf("是否继续删除？是输入Y，否输入N\n");

scanf("%c",&c);

}while(c=='Y');

getchar();getchar();

break;

case 4:

printf("当前的数独为：\n\n");

sudoku\_print(a);

getchar();getchar();

break;

case 5:

printf("提示为：");

srand((unsigned)time(NULL));

while(1)

{

col=rand()%9;

lin=rand()%9;

if(a[col][lin]);

else

{

a[col][lin]=b[col][lin];

break;

}

}

printf("%d %d %d\n\n",col+1,lin+1,a[col][lin]);

printf("数独为：\n");

sudoku\_print(a);

getchar();getchar();

break;

case 6:

printf("数独的解为：\n");

sudoku\_print(b);

getchar();getchar();

break;

case 0:

printf("\n数独问题已解决\n");

getchar();getchar();

break;

}

}//while op

}