

**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： CS2003**

**学 号： U202015359**

**姓 名： 张庙松**

**指导教师： 卢萍**

**报告日期： 2021-09-27**

**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数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

**目录**

[**1引言** 1](#_Toc84777149)

[1.1课题背景与意义 1](#_Toc84777150)

[1.1.1数独游戏的发展及规则 1](#_Toc84777151)

[1.1.2 SAT问题简介 2](#_Toc84777152)

[1.2国内外研究现状 2](#_Toc84777153)

[1.3课程设计的主要研究工作 3](#_Toc84777154)

[**2系统需求分析与总体设计** 4](#_Toc84777155)

[2.1系统需求分析 4](#_Toc84777156)

[2.2系统总体设计 4](#_Toc84777157)

[**3系统详细设计** 6](#_Toc84777158)

[3.1有关数据结构的定义 6](#_Toc84777159)

[3.2主要算法设计 7](#_Toc84777160)

[**4系统实现与测试** 12](#_Toc84777161)

[4.1系统实现 12](#_Toc84777162)

[4.1.1系统功能 12](#_Toc84777163)

[4.1.2数据结构 12](#_Toc84777164)

[4.1.3函数声明 13](#_Toc84777165)

[4.2系统测试 16](#_Toc84777166)

[**5总结与展望** 29](#_Toc84777167)

[5.1全文总结 29](#_Toc84777168)

[5.2 工作展望 29](#_Toc84777169)

[**6体会** 31](#_Toc84777170)

[**参考文献 32**](#_Toc84777171)

[**附录 34**](#_Toc84777172)

[**优化前程序 34**](#_Toc84777173)

[**优化后程序 46**](#_Toc84777174)

# 1引言

## 1.1课题背景与意义

数独（Sudoku）是一个放置难题，同时也是运用纸和笔进行演算和推理的逻辑游戏，从发展至今，由于它具有开发智力，锻炼逻辑思维的优点，被大家所认识以及喜爱。数独游戏在世界上广受欢迎，但进入中国的时间并不长，但是得到了中国广泛群体的热爱，发展迅猛，许多数独俱乐部的诞生、数独游戏活动被大量的举办。如今在数独游戏协会、众多的民间数独社团和广大爱好者的共同努力下，将会更大程度上促进数独游戏在中国的推广与普及。

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 问题在人工智能领域的重要地位，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。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。

### 1.1.1数独游戏的发展及规则

数独游戏的前身为“九宫格”，最早起源于中国东汉末年。十八世纪末，瑞士数学家莱昂哈德·欧文发明了一种叫做“拉丁方块”的游戏，数独类游戏经过一系列的发展，终于在2004年真正的为世界所知晓。

数独游戏的规则很简单，在给定的一个由3×3子网格组成的9×9网格区域中，其中在某些单元格中给出了一些数字，目的是在网格中的每个单元格中输入数字1到9，以便每个行，列和子网格只包含1到9不重复的数字。

### 1.1.2 SAT问题简介

命题逻辑公式的可满足性问题（SAT），是数理逻辑、自动推理、计算机科学、集成电路的设计验证和人工智能等领域中的基本问题，是一个典型的NP完全问题。SAT 问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该类问题的高效算法意义重大。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。

## 1.2国内外研究现状

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，其它的完备算法大都是在 DPLL 算法的基础上衍生出来的， 是对 DPLL 算法的改进。由于 SAT 问题本身的特性使得其最坏情况下的时间复杂 度是指数级别，最初这使得许多的研究者望而却步。而后，S.A.Cook 在 1971年证明了SAT 问题是 NP 完全问题，这更加削弱了许多学者研究 SAT 问题的兴趣，从而导致了SAT 问题在很长的一段时间里都没有得到较好的重视，发展非常缓慢， 研究成果较少。如今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。

Bart Selman和Henry Kautz分别于1997年和2003年在人工智能第五届国际合作会议上提出了 SAT 问题面临的十大挑战性问题，并在 2001 年和 2007 年先后 对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出 对于 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。

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

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

本次课程设计主要学习命题可满足性问题的相关指示以及基于DPLL算法的相关算法思想并选择合适的分裂策略解决SAT问题，实现对于算法思想的运用。

研究的主要内容分为如下的几个方面：

1.对 SAT 问题的研究背景、意义及现状进行了简要总结，学习了命题逻辑可满足性问题的基本理论知识以及挖洞法生成唯一解数独棋盘的基本理论知识。

2.对基于DPLL的SAT 算法进行了深入研究，并仔细分析了 DPLL 算法的实现，对该算法中所使用到的数据结构和一些关键技术给予总结和分析。

3.建立合适的数据结构，对于应该要读取的cnf文件选择合适的物理存储方式，方便后续的对于DPLL算法的运用

4 .实现基于DPLL算法实现的SAT求解器，并对系统的算法以及数据结构进行优化，并给出求解对应的SAT问题所计算出来的优化率。

5.实现生成随机数独，并且基于挖洞法给出含有唯一性解的数独游戏，设计游戏交互界面，使得数独游戏具有一定的可玩性。

6.将改进后的算法应用到现实中的实际问题—数独游戏。通过编码转换为 CNF 公式的形式后，输入到改进后的 DPLL 求解器中进行求解，并利用挖洞法生成具有唯一解数独棋盘。

# 2系统需求分析与总体设计

## 2.1系统需求分析

由于数独游戏在近几年的发展以及数独游戏对于青少年的思维拓展方面的优势，数独游戏得到了广泛认可，但是对于部分数独游戏的求解时间问题仍然存在。基于数独问题，本次实验将可满足性的数独问题转化为SAT问题，基于DPLL算法对于该问题进行较高效率的求解。系统通过进行随机生成的完整数独进行挖洞，形成一个具有唯一解的数独问题，并将数独转化为cnf文件，通过DPLL算法进行求解，同时，本次实验设计的界面具有一定的交互性，可满足用户对于数独游戏的思考。

## 2.2系统总体设计

本次实验设计主要包含两个界面，实验设计的系统具分为两个板块，分别是基于DPLL算法的SAT求解器设计板块和数独游戏生成与交互板块。并且在各自的板块中含有不同的小分块。通过在SAT分块中的数独游戏分块实现两者的交互

在第一个SAT求解板块：读取CNF文件，输出CNF文件，DPLL算法求解，保存为res文件，数独游戏，退出。

在数独游戏板块：生成一个随机数独，挖洞，游戏，保存为CNF文件，使用DPLL进行求解，输出答案，退出。

程序的流程见下图2-1.



图2-1 实验系统程序流程图

# 3系统详细设计

## 3.1有关数据结构的定义

实验中需要读取CNF文件，其中分为文字和子句，按照文件中行列顺序完整的读入程序中，所以使用的物理结构为邻接表。

对于CNF文件中的文字的邻接表的定义：

typedef struct lNode {

int l; //存储文字

int mark; //标记该文字是否被删除，初始值为1，表示没有被删除

struct lNode\* next; //指向下一个文字

} lNode, \*plNode; //存储一个文字节点

对于CNF文件中的子句的邻接表的定义：

typedef struct cNode {

int l\_count; //存储一个子句的文字个数

int flag; //记录该子句因为flag而被删除，初始值为0，flag为需要删除的文字

int mark; //标记该子句是否被删除 ，初始值为1，表示没有被删除

lNode\* firstl; //指向第一个文字节点

struct cNode\* next; //指向下一条子句

} cNode, \*pcNode; //子句头节点

对于CNF的头节点的定义：

typedef struct cnf {

int varinum; //变元个数

int claunum; //子句个数

cNode\* firstc; //指向第一个子句节点

} cnf, \*pcnf; //一个cnf公式

用于存储答案的顺序表的定义：

typedef struct SqList {//用于存储答案的顺序表的

int\* elem;

int length;

} SqList;

其中的各种数据关联如图3-1所示：



**图3-1 存储邻接表物理结构示意图**

## 3.2主要算法设计

函数定义如下：

int createCNF(pcnf\* L, char filename[]);//创建CNF文件

int showCNF(pcnf L); //打印CNF公式

int InitList(pcnf L,SqList &An);//初始化存储答案的顺序表

int RemoveClause(pcnf L,int flag);//删除含有文字的子句

int AddClause(pcnf L,int flag);//恢复含有文字的子句

int EmptyClause(pcnf L);//判断子句是否为空

int DPLL(pcnf L,SqList &An);//DPLL

int ResWrite(int res,double time,SqList &An,char filename[]);//将求解的结果写入到res

int option2(pcnf L);//策略选择

int reverse(pcnf L, int flag);//换用文字的负数

int option1(int \*m);//策略选择

void printsuduko(int arr[][9]);//打印数独

int get\_sudoku(int temp[SIZE][SIZE]);//建立数独

void cleanup(int temp[SIZE][SIZE]);//清空数独

void output\_sudoku(int temp[SIZE][SIZE]);//输出数独

int check\_sudoku(int temp[SIZE][SIZE], int i, int j, int k);//检查填入数字是否合法

int sd(int sudo[9][9]);//合法性判断

int digsudu(int shudu[9][9], int num, int beifen[9][9]);//挖洞

int check(int shudu[9][9]);//检查是否完成数独游戏

int savecnf(int shudu[9][9], char\* filename, int dignum);//数独保存为CNF

int sudures(SqList& An, char filename[],int beifen);//DPLL求解结果写入到res

以上为程序所用到的头文件，接下来下来解释程序调用的函数过程：

1.CNF文件的处理

调用int createCNF(pcnf\* L, char filename[])函数，利用合适的循环判断语句创建一个CNF结构。建立完成后可以利用int showCNF(pcnf L)将CNF文件输出到屏幕

2.DPLL算法

DPLL算法的分裂选择使用两种不同的策略：

策略一：根据文字出现的频率进行选择，即以文字优先选择，在这种情况下，文字出现的次数越多，便会被优先选择出来，其中的选择利用option1函数实现，根据出现次数的多寡将文字排序，那么包含改文字的子句会被删除，包含该负文字的子句会删除该负文字。

策略二：由于在选择子句时，优先选择单子句会使算法的时间大大优化，那么同理，优先选择文字数较少的子句也能优化程序运行的时间，那么，策略二就是以子句的文字数优先，当子句的文字数较少时优先选出并将其中的某一个文字作为选择出的文字。

以下为DPLL算法的具体处理：

调用int DPLL(L, An)函数后，将CNF文件传入处理，由于采用递归溯源，一开始的时候，判断该CNF文件是否成功求解，即子句数是否为0，成功则返回TRUE，然后调用EmptyClause(L)判断是否存在空子句，如果存在，证明为不满足算例，返回FALSE，然后调用single(L)函数找到单子句的文字进行优先满足，将包含该文字的子句和该文字的负文字使用RemoveClause(L, tar)进行删除，在结束之后仍然对算例的可否满足进行判断，继续寻找单子句的文字，在没有单子句后，会根据所输入的策略选择进行文字的选择并且进行递归，如果该文字不能够得出可满足性的解，则变换该文字为负文字，其中调用了reverse(L, tar)用来删除包含负文字的子句并恢复之前删除的正文字，如果失败，调用AddClause(L, tar)恢复所有删除的文字和子句，并进行递归，如果递归的结果返回正确，利用链表An存放答案的正负。

3.数独问题转化为SAT问题

按照本次课程设计的要求，对于数独问题要转化为SAT问题，数独可以用一个9\*9的二维数组进行表示，本次实验采用合法填充后进行挖洞来生成一个随机数独，但是需要将数独的格局转化为SAT问题，并把他表示为CNF文件，对于数独格局中的每一个位置以及其中的数要有唯一的编码与之一一对应，我们需要定义问题的BOOL变元，我们可以将变元可按语义编码为1～9之间数字构成的三位整数*ijk*，*i*, *j*, *k*∈{1,2,…,9}，其中*i*表示单元格的行号，*j*表示单元格的列号，*k*表示单元格<*i*, *j*>填入的数字为*k*。如163变元表示第1行6列填入3；负文字 -452表示第4行5列不填入2。这样编码共有729个变元。但是变元的编码完成后，我们需要表示数独的三种约束，行约束，列约束，盒约束。以单元格<1,1>例，这可以表示为如下子句：

111 112 113 114 115 116 117 118 119 0

-111 -112 0

-111 -113 0

……

-118 -119 0

上述表示中，每个子句的末尾的0表示结束标记；第一个子句的含义是单元格<1,1>可填入至少一个数字；后面的子句集共同表示只能填入一个数字，子句-111 -112 0表示不能同时填1与2；其它类推。按这种方式需要对81个单元格进行类似表示，得到对应的子句集。

行约束要求每行需要填入1～9中的每个数字，且每个数字只出现一次。以第1行为例可表示为（此处在每个子句后加入注释，说明子句的含义）：

111 121 131 141 151 161 171 181 191 0 第1行含有1

112 122 132 142 152 162 172 182 192 0 第1行含有2

… …

119 129 139 149 159 169 179 189 199 0 第1行含有9

-111 -121 0 前两格不同时为1

-111 -131 0 第1与第3格不同时为1

… …

-111 -191 0 第1与第9格不同时为1

… …

同理可以得到列约束。

对于3×3的盒子约束，以左上角的盒子为例进行说明，其子句集可表示如下：

111 121 131 211 221 231 311 321 331 0 包含1

112 122 132 212 222 232 312 322 332 0  包含2

… …

119 129 139 219 229 239 319 329 339 0 包含9

-111 -211 0 11格与21格不同时为1

-111 -311 0 11格与31格不同时为1

-111 -121 0 11格与12格不同时为1

… …

最后，对于每个具体的数独游戏，已经填入了部分提示数，如图2.3中的左图，每个提示数可表示为一个单子句，如第2行3列填入5，对应单子句如下：

235 0

SAT公式CNF文件中，一般变元是从1进行连续编码的，可以将上述语义编码转换为自然顺序编码，公式为：

*ijn* → (*i*-1)\*81+(*j*-1)\*9+*n*；

以下是数独游戏的具体实现：

调用函数get\_sudoku利用随机数生成一个随机的数独格局，生成完成后，根据输入的挖洞数调用digsudu(shudu, dignum)函数随机挖洞，其中有对挖洞后数独解的唯一性的判断，挖洞后的数独应当要满足唯一解的要求（PS：由于是随机挖洞，其中的判断采用了深搜，但是只能看到下一步，所以挖洞的个数不不宜太多，否则会产生多重解）在保存为CNF文件时，根据挖洞数，会产生不同的子句个数，然后通过行约束，列约束，盒约束共10449句语句，其中，由于行约束和列约束已经完成，盒约束只用采用与该数不同行同列的数，简化了语句的个数，在写完后可以直接调用在SAT菜单中的DPLL直接求解，但是编码的唯一性表示说明需要有翻译的工作，也就是sudures函数，他的本质与ResWrite相同，多了编码的翻译，使得可以直接输出数独的答案。

# 4系统实现与测试

## 4.1系统实现

系统实现的开发工具是Visual Studio 2019，数据结构算法的实现以及程序测试的结果具体如下。

### 4.1.1系统功能

基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。达成如下6个功能：

1.输入输出功能：实现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：实现读取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；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。

3.基于DPLL的求解器：基于DPLL算法框架，实现SAT算例的求解。

4.时间性能的测量：实现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，提供明确的性能优化率结果。

6.SAT应用：将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，具有简单的交互性。

### 4.1.2数据结构

系统的实现所用的宏定义和结构定义分别如下所示。

1.宏定义

预定义常量和类型，用于判断函数结果的状态。

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define SIZE 9

2.数据结构定义

基于一定的物理结构，定义用于后续程序处理的链表和结构，分为文字节点的定义：

typedef struct lNode {

int l; //存储文字

int mark; //标记该文字是否被删除，初始值为1，表示没有被删除

struct lNode\* next;

} lNode, \*plNode; //存储一个文字节点

子句节点的定义：

typedef struct cNode {

int l\_count; //存储一个子句的文字个数

int flag; //记录该子句因为flag而被删除，初始值为0，flag为需要删除的文字

int mark; //标记该子句是否被删除 ，初始值为1，表示没有被删除

lNode\* firstl; //指向第一个文字节点

struct cNode\* next; //指向下一条子句

} cNode, \*pcNode; //子句头节点

CNF头节点的定义：

typedef struct cnf {

int varinum; //变元个数

int claunum; //子句个数

cNode\* firstc; //指向第一个子句节点

} cnf, \*pcnf; //一个cnf公式

顺序表的定义：

typedef struct SqList {//用于存储答案的顺序表的

int\* elem;

int length;

} SqList;

### 4.1.3函数声明

整个系统分为2个大板块，分别是SAT求解问题，数独游戏

1.SAT求解问题

有CNF读取，CNF输出，DPLL求解，保存为res文件等作用：

创建CNF文件：int createCNF(pcnf\* L, char filename[]);

展示CNF公式：int showCNF(pcnf L);

清空CNF公式：void destory(pcnf L);

判断是否为单子句：int single(pcnf L);

判断是否为重言式：void tauto(pcnf L);

排序函数：void quickSort(int\* array, int\* arraynum, int low, int high);

快速排序：void sort(pcnf L, int key[3500], int value[3500]);

初始化顺序表：int InitList(pcnf L,SqList &An);

删除含有文字的子句：int RemoveClause(pcnf L,int flag);

恢复含有文字的子句：int AddClause(pcnf L,int flag);

判断子句是否为空子句：int EmptyClause(pcnf L);

DPLL求解：int DPLL(pcnf L,SqList &An,int now\_l);

写入到res文件：int ResWrite(int res,double time,SqList &An,char filename[]);

策略2：int option2(pcnf L);

换用文字的负数：int reverse(pcnf L, int flag);

策略1：int option1(int \*m);

利用以上函数实现板块一，其中DPLL的函数调用如下图所示：



图4-1 DPLL算法求解板块函数调用情况

2.数独游戏

有创建数独格局，挖洞，游戏，写入到CNF文件，DPLL求解等功能：  
建立数独：int get\_sudoku(int temp[SIZE][SIZE]);

打印数独：void printsuduko(int arr[][9]);

清空数独：void cleanup(int temp[SIZE][SIZE]);

输出数独：void output\_sudoku(int temp[SIZE][SIZE]);

检查填入数独是否合法：int check\_sudoku(int temp[SIZE][SIZE], int i, int j, int k);

对于数独合法性判断：int sd(int sudo[9][9]);

挖洞：int digsudu(int shudu[9][9], int num, int beifen[9][9]);

检查是否完成数独游戏：int check(int shudu[9][9]);

保存为CNF文件：int savecnf(int shudu[9][9], char\* filename, int dignum);

保存为res文件：int sudures(SqList& An, char filename[],int beifen);

利用以上函数以及DPLL函数能够实现板块二。

## 4.2系统测试

1.读取CNF并输出测试

测试读取CNF算例文件，并输出验证是否读取正确。测试内容分别为：1读取CNF算例文件，并输出验证是否读取正确。2输出CNF文件

分析：在读取CNF文件时，可能存在读取的文件名不合法，文件内部的格式不合法等情况，需要一一验证

表4-1测试一数据样例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 测试用例 | 测试结果 |
| 1 | 测试合法文件读入 | 1.cnf | 读取成功 |
| 2 | 测试非法扩展名 | 1 | 读取失败 |
| 3 | 测试非法内容 | ceshi.cnf | 文件不合法 |
| 4 | 测试CNF文件输出 | 1.cnf | 输出成功 |

测试结果见图4-2，图4-3，图4-4，图4-5：

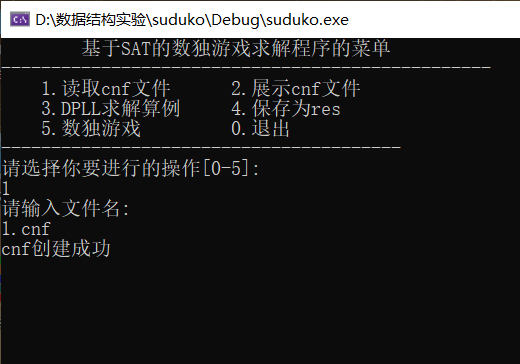


图4-2读取CNF算例文件测试1结果

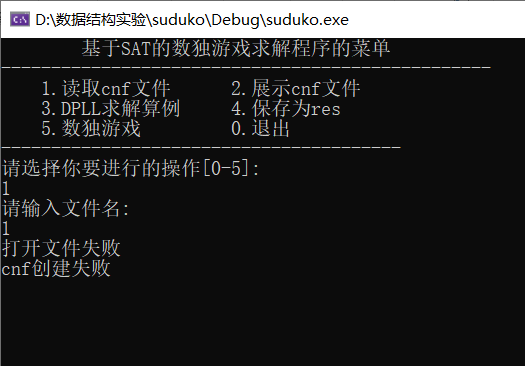


图4-3读取CNF算例文件测试2结果

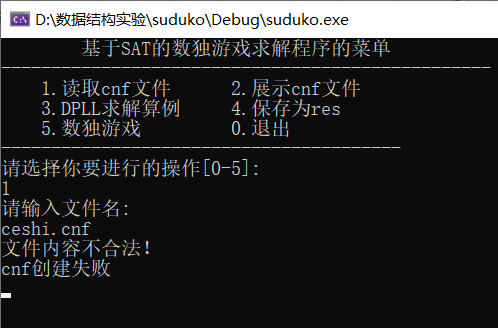


图4-4读取CNF算例文件测试3结果

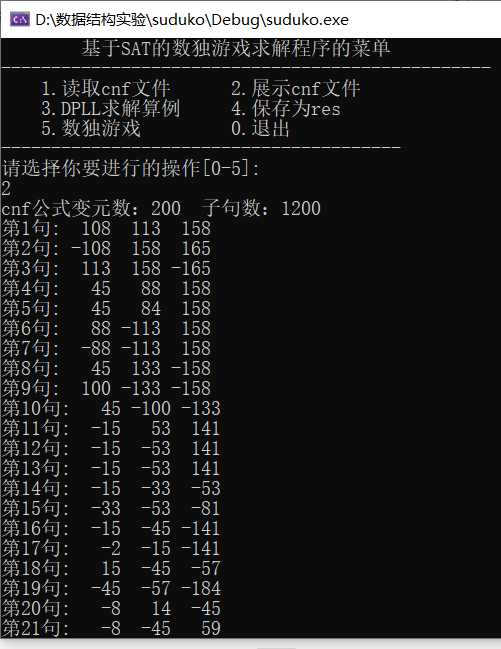


图4-5读取CNF算例文件测试4结果

2.基于DPLL的SAT求解器测试

测试基于DPLL的SAT求解器的功能以及优化过的SAT求解器的优化率。测试内容分别为：1.测试可满足算例并进行输出算例的解2.测试不可满足算例3.进行多次测试，找到最大运行算例4.将求解后的文件写入到res文件中。

分析：未优化前优化选取策略，结果随着变元数量的增加，进而算法回溯次数的增多，时间的花销非常大。基本上不能满足测试用例的需求。通过优化数据结构和综合策略选择，使得程序能够运行较为复杂的算例，经过优化后，求解时间大幅度减少。在实验过程中发现，对于数独问题的求解，结合单子句规则和按顺序选取假设变元的方法求解对应的SAT问题的效率更为明显。

表4-3测试二数据样例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 测试文件 | 测试结果 |
| 1 | 测试用策略一可满足算例的解 | 2.cnf | 该算例为可满足算例 |
| 2 | 测试不可满足算例的解 | 4.cnf | 该算例为不可满足算例 |
| 3 | 测试用策略二可满足算例的解 | 1.cnf | 该算例为可满足算例 |
| 4 | 测试写入到res文件 | 1.cnf |  |

测试结果见图4-6，4-7，4-8：

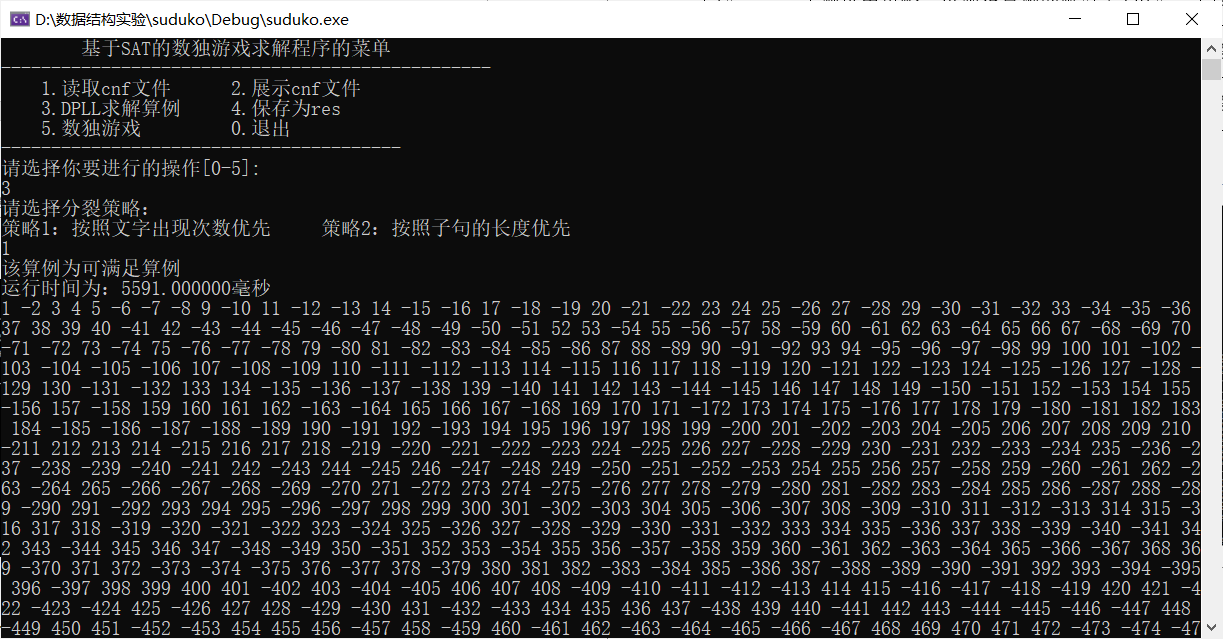


图4-6基于DPLL的SAT求解器测试1结果



图4-7基于DPLL的SAT求解器测试2结果

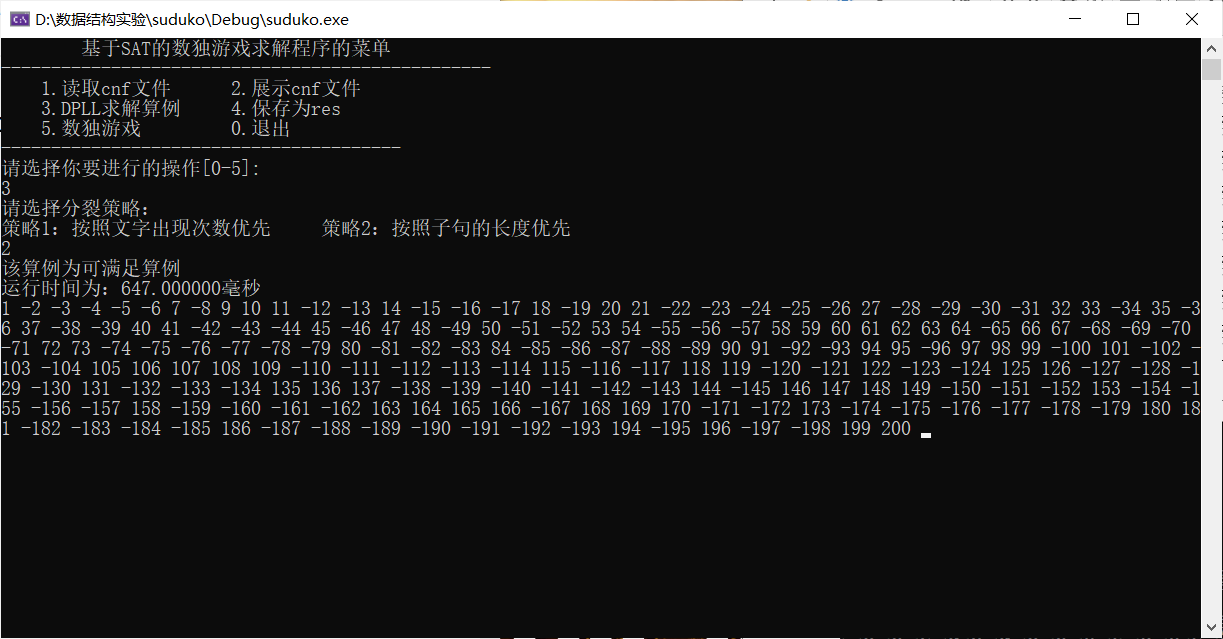


图4-8基于DPLL的SAT求解器测试3结果

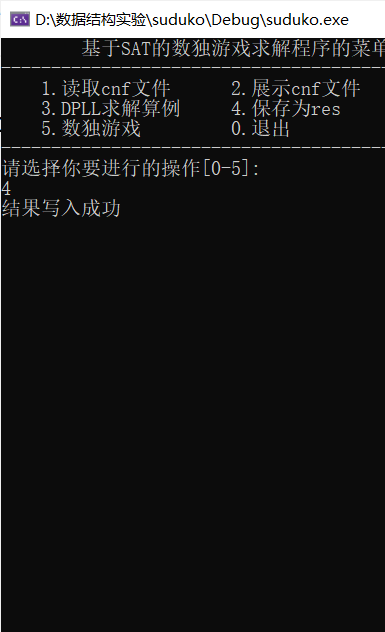
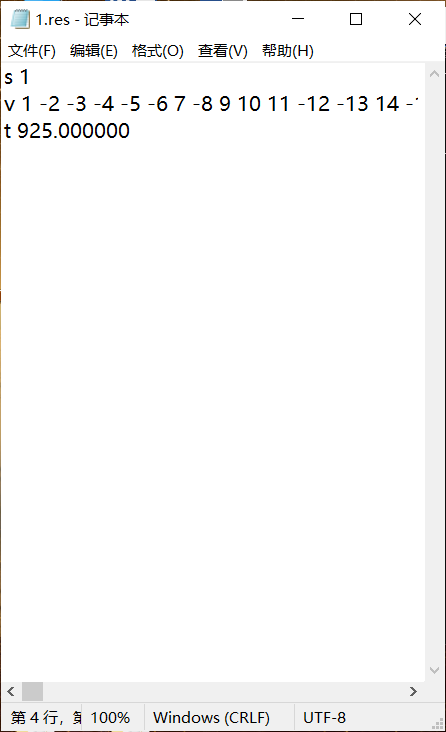


图4-9基于DPLL的SAT求解器测试4结果

测试内容3见表格4-4。由于最开始的策略选择无法完成测试的用例检查，所以只显示优化后的时间，优化前的时间默认为无限大。优化后的时间取两策略的最小值。

表4-4 可运行的测试样例结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 测试用例 | 优化前需要时间 | 最佳优化后需要时间（ms） |
| 1 | 1.cnf | 无穷大 | 647 |
| 2 | 2.cnf | 无穷大 | 5591 |
| 3 | 3.cnf | 无穷大 | 1087 |
| 4 | 4.cnf | 无穷大 | 22806 |
| 5 | 5.cnf | 无穷大 | 547 |
| 6 | 6.cnf | 无穷大 | 146120 |
| 7 | 7-1.cnf | 无穷大 | 853 |
| 8 | 8-1.cnf | 无穷大 | 1133 |

表4-4的测试结果如下列各图所示：

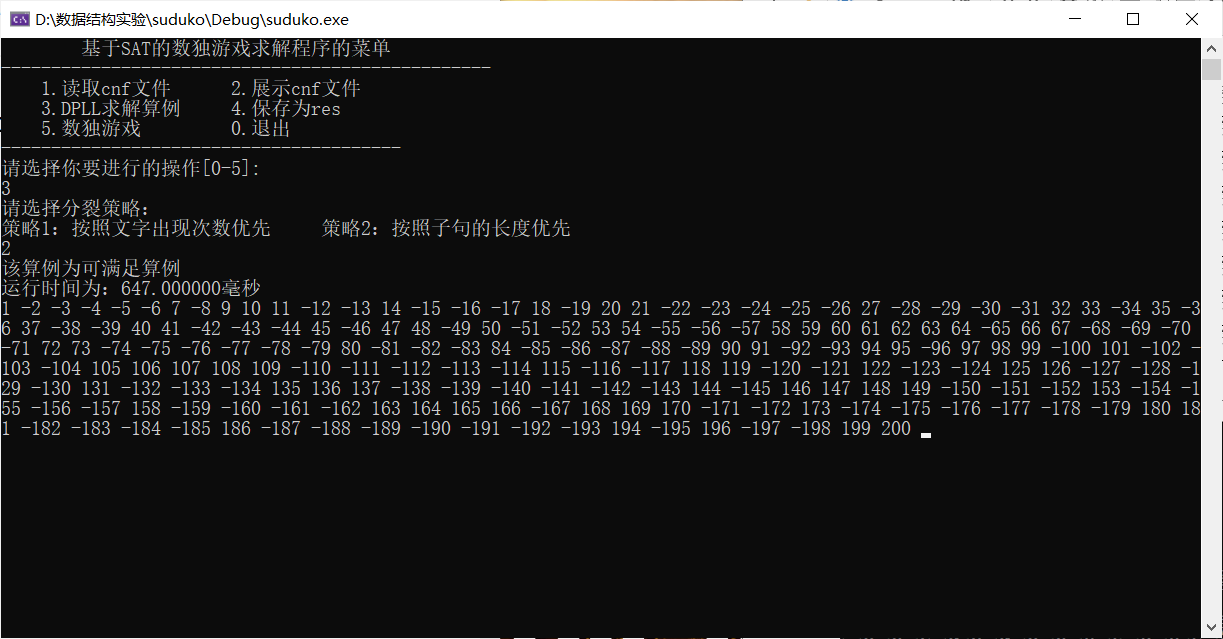


图4-10可运行的测试用例1的运行结果

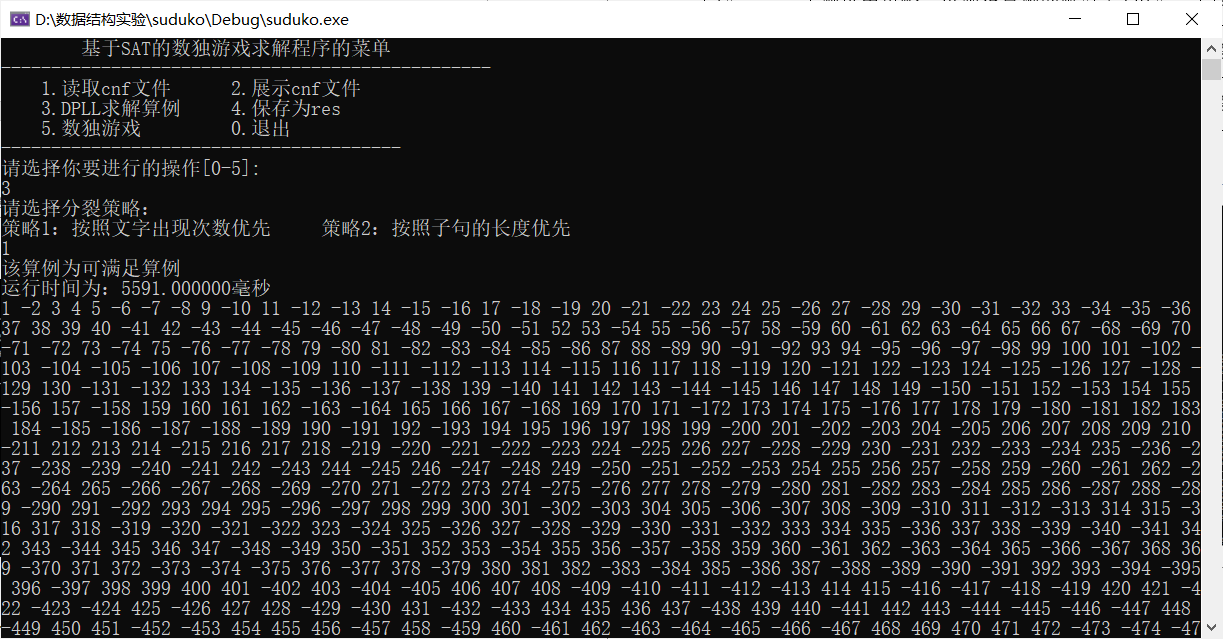


图4-11可运行的测试用例2的运行结果

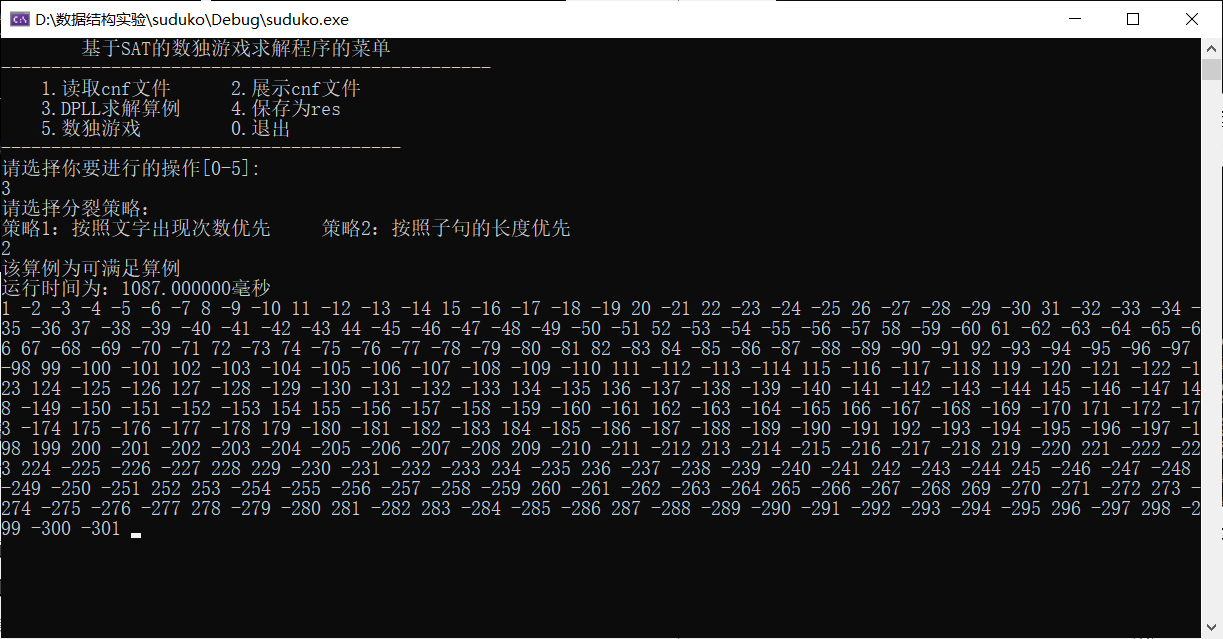


图4-12可运行的测试用例3的运行结果



图4-13可运行的测试用例4的运行结果

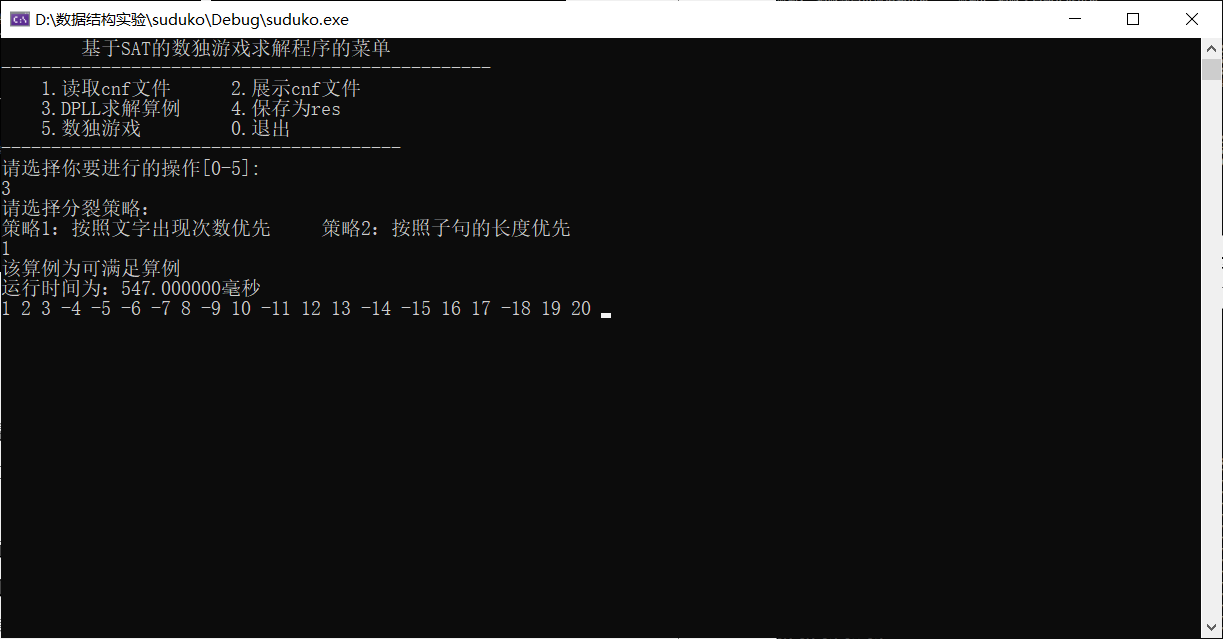


图4-14可运行的测试用例5的运行结果

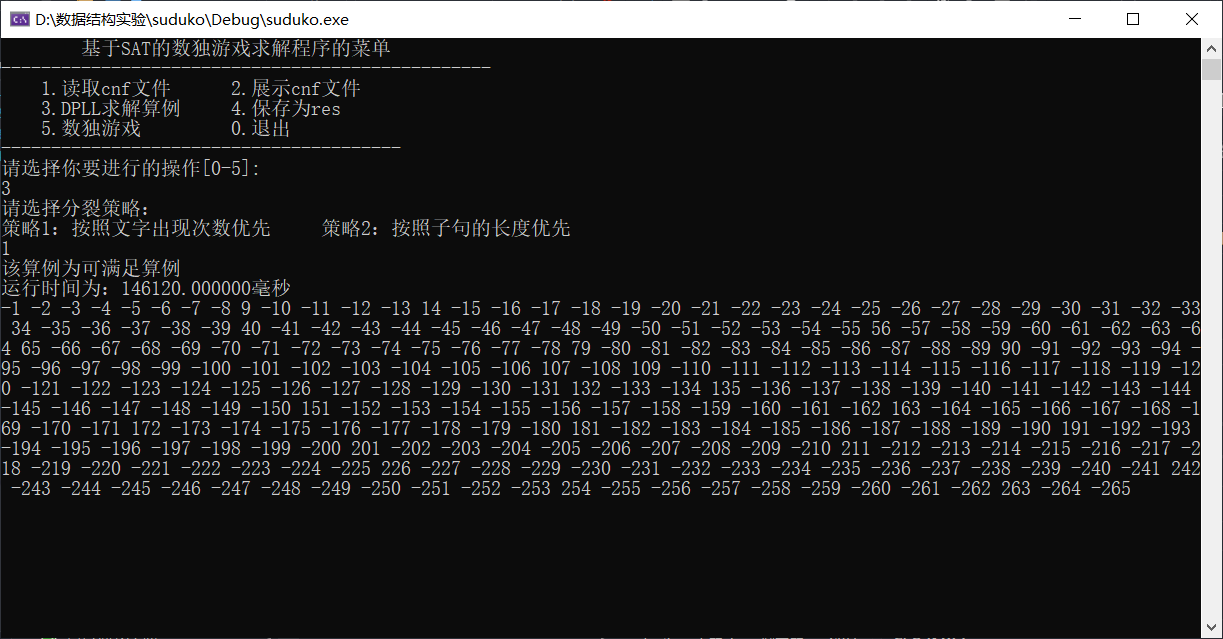


图4-15可运行的测试用例6的运行结果

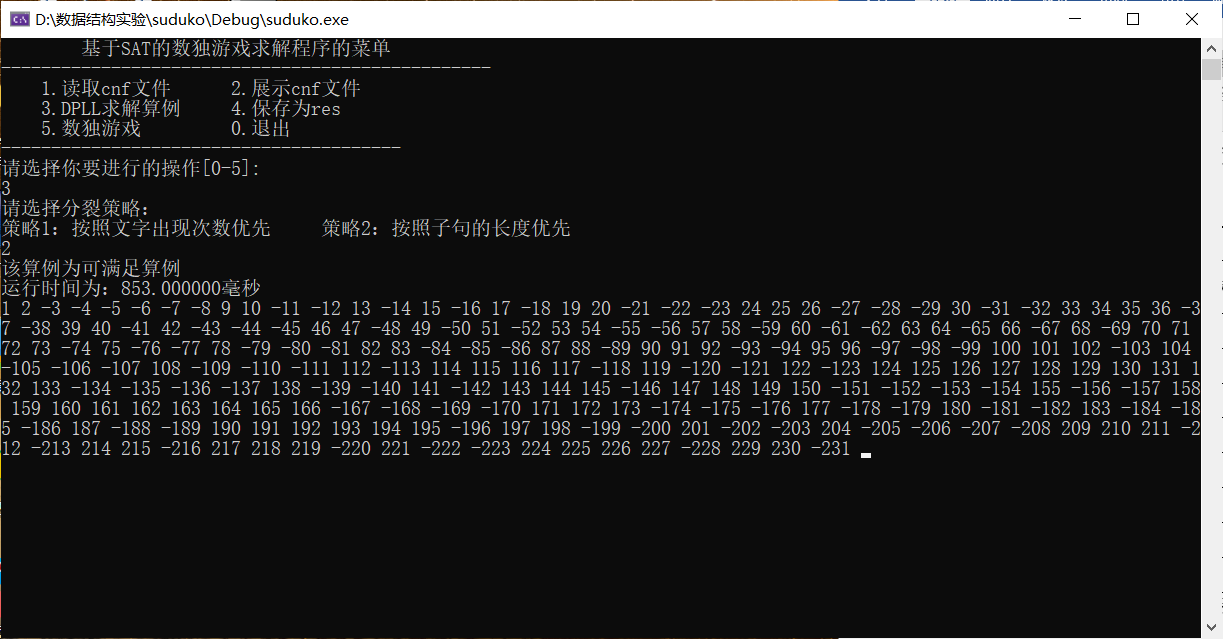


图4-16可运行的测试用例7的运行结果



图4-17可运行的测试用例8的运行结果

3.数独游戏转换成SAT问题测试

测试数独游戏转换成SAT问题测试。测试内容为：通过使用随机函数生成数独棋盘，可以实现数独游戏的功能，有输入的提示，将数独问题转换成SAT问题带入SAT问题求解器，最后得到其中一组数独解。

分析：在挖洞完成后要能够有游戏的功能，那么需要根据数独游戏的格局来对于用户输入的数据进行判断，是否符合数独游戏的格局，在保证挖洞合理的情况下，使用DPLL算法进行求解，最后要能够输出数独的最终解。

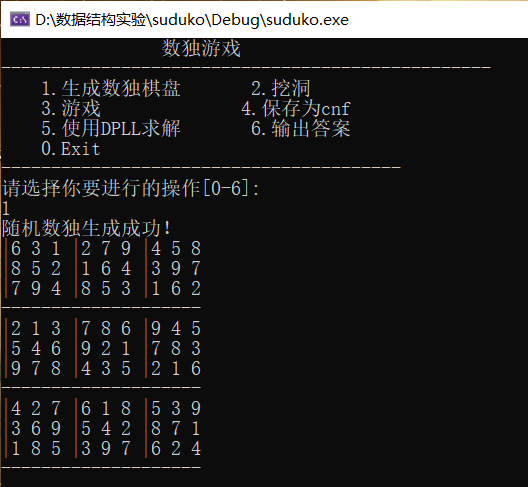


图4-18生成的随机数独

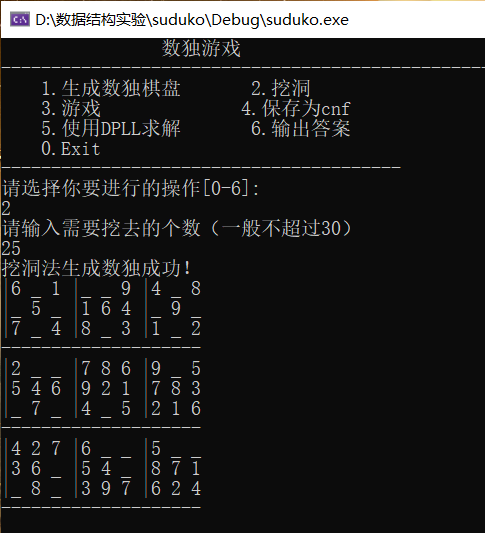


图4-19随机挖洞生成数独游戏

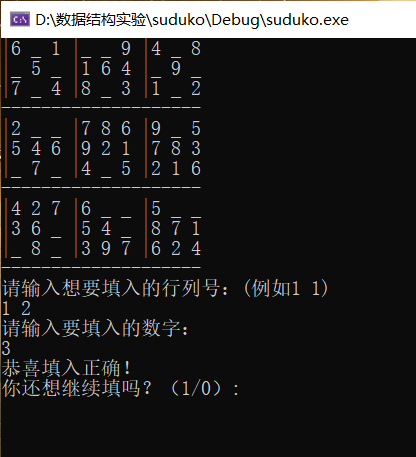


图4-20数独游戏的操作提示

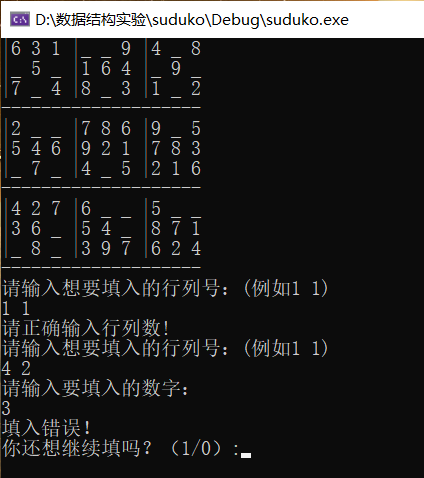


图4-21数独游戏填入错误提示

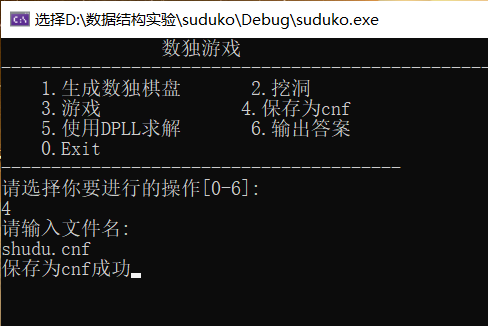
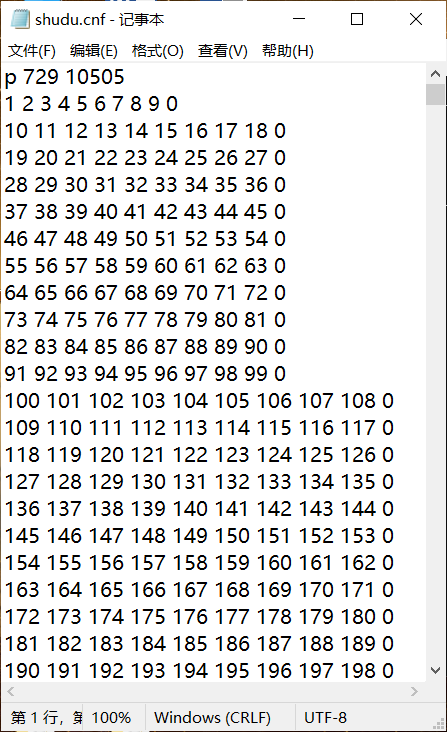


图4-22数独问题保存为CNF文件

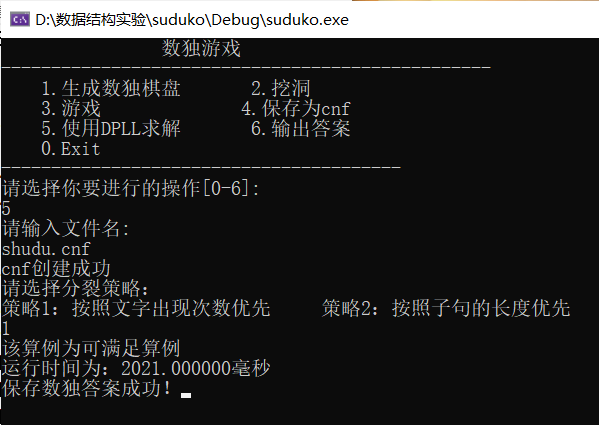
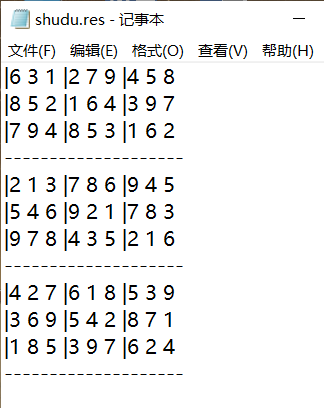


图4-23 DPLL求解CNF

# 5总结与展望

## 5.1全文总结

本次实验圆满完成了基于SAT的数独游戏求解系统的设计与实现，主要完成的工作如下：

1.查阅相关资料，了解和学习当前SAT问题的知识体系以及应用范围，掌握SAT求解器的算法内核与具体实现策略。同时，学习数独问题转换为SAT问题的一些知识体系和架构，为接下来的课程设计实验部分的开展奠定了坚实的理论基础。

2.设计并实现系统的输入输出板块的功能，使系统实现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。

3.设计并实现对读取的CNF文件进行遍历输出的功能，即遍历内部结构逐行输出与显示每个子句达到检验的目的。

4.基于DPLL算法框架，设计SAT求解器并实现SAT求解器正确求解CNF文件中提供的SAT算例。

5.对基本DPLL算法部分进行存储结构优化即建立一张用于存储的邻接表和一张用于索引的邻接表。对于变元选取策进行优化即按照变元出现次数进行选取结合单子句规则。

6.实现利用随机生成函数，创建一个初始的数独棋盘，将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，得到数独解。利用已知的数独解，通过挖洞法游戏挖出一个具有唯一解的数独棋盘，提供给用户，同时数独游戏具有简单的交互性。

## 5.2 工作展望

基于 DPLL 的SAT 求解器在求解性能上有了一定的改善，但是还有许多地方有待改进。现今的许多求解器在求解许多大规模的 SAT 实例时往往需要先对 CNF 公式进行预处理，经过预处理化简后导出一个简化的 CNF 公式，然后再用算法对简化后的公式进行求解，这样效率会更高一些，下面列出了几个将来的工作重点和研究方向：

1.为算法加入合适的预处理器，如引入高级前向推理机制，对 CNF 公式进行预处理或化简，以便能更高效地进行求解。

2.可以尝试将本文中的算法应用到更大规模的实际问题中，比如硬件验证、自动化推理等领域。

3.算法采用的数据结构和具体实现有待改进，可以引入更巧妙的数据结构来表示文字和子句信息，也可以优化代码实现以增强程序的内聚性。

4.可以在 DPLL 的求解过程中寻求更高效的启发式变量决策策略，以减少算 法的回溯次数和搜索空间。

5.可以通过对命题逻辑的可满足性问题的逻辑结构进行分析，采用一种基于 解方程组的 SAT 问题求解思路。即先将合取范式转换为等价的方程组形式，然后 再对方程组进行求解，进而得到对应 SAT 问题的解

# 6体会

本次课程设计应该说是一次较为系统的对于某一问题的求解，与之前的C语言程序的题目不同，这次实验我学到了很多东西，有很多收获，但是也能够看出来自身有很多的不足。这次的课程设计实验，对于我来说，有很大的难度，对实验内容知识体系的不了解，编码能力不强，所以刚开始看到课设设计书时，有些无从下手。之后通过查阅相关文献，认真去学习自己的知识盲点，在程序设计中，能够举一反三，专心致志，一点点去解决难题，完成课程设计任务。

在学习的过程中，认真了解SAT问题，DPLL算法，以及DPLL算法的优化策略。到之后的数独问题转化为SAT问题的研究，自己的阅读和学习能力得到了很大的提高。

接触了这个课设计题目：基于SAT的数独游戏求解。知道了SAT 问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该类问题的高效算法的意义重大。其中，SAT问题的求解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。

如下是对自己实验过程的出现的一些对应问题个人心得以及总结。

1. 在DPLL算法部分，刚开始没有理解好相关的知识点，所以在刚开始使用迭代法实现时，出现了很多的BUG。并且一开始的方法很多都是无法求解测试算例，后来在老师和同学的帮助下，自己也认真学习了DPLL算法的内容，采纳了同学的算法建议，最终成功完成。

2. 将数独游戏问题转换成CNF文件格式的时候，遇到了很大的难题，由于要将数独的编码转化，在盒约束的时候，对于其中的循环语句没有头绪，所以在完成了一个盒约束语句后，不得已将该语句复制了几份以达到完成盒约束的目的。

3. 在通过挖洞法生成唯一解数独棋盘时，我一开始并没有完成唯一解的判定和部分输出语句的准备，使得函数出现了很大的问题，好在同学和助教提醒之后才改善了程序。

# 参考文献

[1]□王静康,张凤宝,夏淑倩等.论化工本科专业国际认证与国内认证的“实质性”.高等工程教育研究,2014,5:1-4

[2]□Stone J A, Howard L P. A simple technique for observing periodic nonlinearities in Michelson interferometers. Precision Engineering,1998,22(4):220-232

[3]□朱印红,袁衍明.Dreamweaver完美网页设计——技术入门篇.(第一版).北京:中国电力出版社,2006:19～20

[4]□Lewis S L. Physics and chemistry of the solar system.北京:北京大学出版社,2014.1～2

[5]□陈剑.上博简《民之父母》“而得既塞於四海矣”句解释[EB/OL］.简帛研究网站，http://www.bamboosilk.org/Wssf/2003/chenjian03.htm．2003-01-18

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

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

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

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

[10]360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

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

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

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

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

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

[15] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

# 附录

# 优化前程序

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef struct lNode {

int l; //存储文字

int mark; //标记该文字是否被删除，初始值为1，表示没有被删除

struct lNode\* next;

} lNode, \*plNode; //存储一个文字节点

typedef struct cNode {

int l\_count; //存储一个子句的文字个数

int flag; //记录该子句因为flag而被删除，初始值为0，flag为需要删除的文字

int mark; //标记该子句是否被删除 ，初始值为1，表示没有被删除

lNode\* firstl; //指向第一个文字节点

struct cNode\* next; //指向下一条子句

} cNode, \*pcNode; //子句头节点

typedef struct cnf {

int varinum; //变元个数

int claunum; //子句个数

cNode\* firstc; //指向第一个子句节点

} cnf, \*pcnf; //一个cnf公式

typedef struct SqList {

int\* elem;

int length;

} SqList; //一个顺序表结构，存储答案

int createCNF(pcnf\* L, char filename[80]);

int showCNF(pcnf L); //打印CNF公式

int InitList(pcnf L, SqList& An);

int RemoveClause(pcnf L, int flag);

int AddClause(pcnf L, int flag);

int EmptyClause(pcnf L);

int Findl(pcnf L, SqList& An);

int DPLL(pcnf L, SqList& An, int now\_l);

int ResWrite(int res, double time, SqList& An, char filename[]);

int main() {

pcnf L;

SqList An;

clock\_t start, end; //记录开始时间与结束时间

double duration; //记录程序运行时间

char filename[200]; //文件名

int op = 1, tag; //菜单选择

int b;

double time;

double s; //记录效率的提升

while (op) {

system("cls");

printf(" 基于SAT的数独游戏求解程序的菜单\n");

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

printf(" 1.创建CNF 2.展示CNF\n");

printf(" 3.DPLL算法求解 4.保存为res\n");

printf(" 0.exit\n");

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

printf("请选择你要进行的操作[0-5]:\n");

scanf("%d", &op);

switch (op) {

case 1: //创建cnf结构体

printf("请输入文件名:\n");

scanf("%s", filename);

if (createCNF(&L, filename) == OK)

printf("cnf公式创建成功\n");

else {

printf("cnf公式创建失败\n");

getchar();

getchar();

break;

}

if (InitList(L, An) == OK)

printf("分配结果存储空间成功\n");

else {

printf("分配结果存储空间失败\n");

getchar();

getchar();

break;

}

getchar();

getchar();

break;

case 2: //显示cnf结构

showCNF(L);

getchar();

getchar();

break;

case 3: // DPLL算法

b = Findl(L, An); //先找到第一个单子句或者用于分裂策略的文字

start = clock();

if (DPLL(L, An, b) == OK) {

tag = 1;

printf("求解成功\n");

} else {

tag = 0;

printf("求解失败\n");

}

end = clock();

duration = ((double)(end - start)) / CLK\_TCK \* 1000;

printf("运行时间为：%f毫秒\n", duration);

for (int i = 1; i < An.length;

i++) //输出求解结果，如果求解不成功，就输出正数

{

if (An.elem[i] == -1)

printf("%5d", -i);

else

printf("%5d", i);

if (i % 10 == 0)

printf("\n");

}

getchar();

getchar();

break;

case 4:

if (ResWrite(tag, duration, An, filename) == OK)

printf("结果写入成功\n");

else

printf("结果写入失败\n");

getchar();

getchar();

break;

case 0:

break;

} // end of switch

} // end of while

return 0;

}

int createCNF(pcnf\* L, char filename[200]) { //创建结构体，以表的形式存储

pcNode p; //子句结点

plNode q; //文字节点

\*L = (cnf\*)malloc(sizeof(struct cnf));

p = (cNode\*)malloc(sizeof(struct cNode));

p->flag = 0; //记录子句是否因为flag而被删除

p->mark = 1; //记录子句是否被删除，1表示未被删除

(\*L)->firstc = p;

FILE\* fp;

int num = 0; //为计算子句数目和变元数目的中间变量

int k = 0; //记录每个子句文字个数

int m = 0; //记录子句的个数

int flag; //记录正负

char c; //读取文件字符使用

fp = fopen(filename, "r");

if (fp == NULL) {

printf("打开文件失败\n");

return 0;

}

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

while (!feof(fp)) {

if (c == 'c') { //把c开头的注释去掉

while (c != '\n') {

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

}

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

} else if (c == 'p') {

while (c != '\n') {

while (c < '0' || c > '9') {

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

}

while (c >= '0' && c <= '9') { //获取变元数

num = num \* 10;

num += c - '0';

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

}

(\*L)->varinum = num;

num = 0;

while (c < '0' || c > '9') {

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

}

while (c >= '0' && c <= '9') { //获取子句数

num = num \* 10;

num += c - '0';

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

}

(\*L)->claunum = num;

num = 0;

}

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

} else if ((c >= '0' && c <= '9') || (c == '-')) {

q = (lNode\*)malloc(sizeof(struct lNode));

q->mark = 1; // mark等于1表示没有被删除

p->firstl = q;

while (c != '\n') {

while (c != '0') {

while (c != ' ') {

flag = 1;

while ((c >= '0' && c <= '9') || (c == '-')) {

if (c == '-') {

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

flag = 0;

}

num = num \* 10;

num += c - '0';

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

}

if (flag == 0)

q->l = -num;

else

q->l = num;

num = 0;

k++;

}

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

if (c == '0')

q->next = NULL;

else {

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

q = q->next;

q->mark = 1;

}

}

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

}

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

p->l\_count = k;

k = 0;

m++;

if (m < (\*L)->claunum) {

p->next = (cNode\*)malloc(sizeof(struct cNode));

p = p->next;

p->flag = 0;

p->mark = 1;

} else

p->next = NULL;

} else

fscanf(fp, "%c", &c); //防止文件结尾有多个换行

}

fclose(fp);

return OK;

}

int showCNF(pcnf L) { //显示cnf结构体

if (L->claunum == 0) {

printf("cnf公式为空\n");

return OK;

}

pcNode p = L->firstc;

plNode q = p->firstl;

printf("cnf公式变元数：%d 子句数：%d\n", L->varinum, L->claunum);

int i = 1;

int j;

while (p) {

if (p->mark == 0)

p = p->next;

else {

printf("第%d句有%d个文字 ", i++, p->l\_count);

q = p->firstl;

j = 1;

while (q) {

if (q->mark == 0)

q = q->next;

else {

printf("%4d:%5d", j++, q->l);

q = q->next;

}

}

printf("\n");

p = p->next;

}

}

return OK;

}

int InitList(pcnf L, SqList& An) { //初始化结果接收顺序表

An.elem = (int\*)malloc((L->varinum + 1) \* sizeof(int)); //分配存储空间

if (!An.elem) {

return ERROR; //分配存储空间失败

};

An.length = L->varinum + 1; //初始化线性表的长度为0

for (int i = 1; i < An.length; i++) {

An.elem[i] = 0; //表示没有经过单子句简化赋值

// printf("%d ",An.elem[i]); 查看程序运行进度，没有必要打开

}

return OK;

}

int RemoveClause(pcnf L, int flag) { //一次找出一个变元，并删除

pcNode p;

plNode q;

p = L->firstc;

q = p->firstl;

while (p) {

if (p->mark == 0)

p = p->next;

else {

q = p->firstl;

while (q) {

if (q->mark == 0)

q = q->next;

else {

if (q->l == flag) { //子句中有flag，删除子句

p->mark = 0;

p->flag = flag;

L->claunum--; //子句数目减一

break;

} else if (q->l == -flag) { //子句中有-flag，删除该文字

q->mark = 0;

p->l\_count--; //文字数目减一

break;

} else

q = q->next;

}

}

p = p->next;

}

}

L->varinum--; //删除变元后，变元数目减一

return OK;

}

int AddClause(pcnf L, int flag) { //恢复前面因为flag而删除的元素，子句

pcNode p;

plNode q;

p = L->firstc;

q = p->firstl;

while (p) {

q = p->firstl;

if (p->mark == 1) { //恢复非L规则删除的文字

while (q) {

if (q->mark == 0 && q->l == -flag) {

q->mark = 1;

p->l\_count++; //子句文字数目加一

break;

} else

q = q->next;

}

} else if (p->mark == 0 && p->flag == flag) { //恢复子句

p->mark = 1;

p->flag = 0;

L->claunum++;

} else {

while (q) {

if (q->mark == 0 && q->l == -flag) {

q->mark = 1;

p->l\_count++; //子句文字数目加一

break;

} else

q = q->next;

}

}

p = p->next;

}

L->varinum++; //变元数目加一

return OK;

}

int EmptyClause(pcnf L) { //查看是否有空句

pcNode p = L->firstc;

while (p) {

if (p->l\_count == 0 && p->mark == 1)

return OK; //当文字数为0，且子句未经过删减，则是空句

p = p->next;

}

return FALSE;

}

int Findl(pcnf L, SqList& An) { //查找符合要求的变元

pcNode p;

plNode q;

int i;

int flag = 0; //用于寻找出现的最高次数和对应的变元

p = L->firstc;

while (p) { //寻找单子句

if (p->mark == 1 && p->l\_count == 1) {

q = p->firstl;

while (q) {

if (q->mark == 1) {

if (q->l > 0)

An.elem[q->l] = 1;

else

An.elem[-(q->l)] = -1;

return q->l;

} else

q = q->next;

}

} else

p = p->next;

}

//改变策略，寻找出现次数最多的变元

int\* a = (int\*)malloc((2 \* An.length - 1) \* sizeof(int));

for (i = 0; i < 2 \* An.length - 1; i++) {

a[i] = 0;

}

p = L->firstc;

while (p) { //记录各变元出现的次数

if (p->mark == 0)

p = p->next;

else {

q = p->firstl;

while (q) {

if (q->mark == 0)

q = q->next;

else {

if (q->l > 0)

a[2 \* (q->l) - 1]++;

else if (q->l < 0)

a[2 \* (-(q->l))]++;

q = q->next;

}

}

p = p->next;

}

}

for (i = 1; i < 2 \* An.length - 1; i++) { //找出出现次数的最大值

if (a[i] > flag)

flag = a[i];

}

for (i = 1; i < 2 \* An.length - 1; i++) { //找到变元

if (a[i] == flag)

break;

}

free(a);

if (i % 2) {

An.elem[(i + 1) / 2] = 1;

return ((i + 1) / 2);

} else {

An.elem[i / 2] = -1;

return (-(i / 2));

}

}

int DPLL(pcnf L, SqList& An, int now\_l) { // now\_l是当前输入用来化简的

int next\_l; //利用Findl函数生成的

if (L->claunum == 0)

return OK; // cnf公式为空，化简成功

else {

if (EmptyClause(L) == OK) { //有空子句，是不满足算例

if (AddClause(L, now\_l) != OK)

printf("恢复失败\n");

if (now\_l > 0)

An.elem[now\_l] = 0;

else

An.elem[-now\_l] = 0;

return FALSE;

} else { // cnf还未化简完，采用单子句原则+分裂策略

next\_l = Findl(L, An);

if (next\_l == 0)

return FALSE;

if (RemoveClause(L, next\_l) != OK)

printf("删除失败\n");

if (DPLL(L, An, next\_l) == OK)

return OK;

else { //化简不成功时，需要改变next\_l的真值，进行回溯

if (AddClause(L, next\_l) != OK)

printf("恢复失败\n");

if (RemoveClause(L, -next\_l) != OK)

printf("删除失败\n");

if (DPLL(L, An, -next\_l) == OK) {

if (next\_l > 0)

An.elem[next\_l] = -1;

else

An.elem[-next\_l] = 1;

return OK;

} else { //最后还未成功，将顺序表相应位置修改为未进行真值判断的状态

if (AddClause(L, -next\_l) != OK)

printf("恢复失败\n");

if (next\_l > 0)

An.elem[next\_l] = 0;

else

An.elem[-next\_l] = 0;

return FALSE;

}

}

}

}

}

int ResWrite(int res, double time, SqList& An, char filename[80]) {

int i = 0;

while (filename[i] != '\0')

i++;

filename[i - 3] = 'r'; //只改变文件的扩展名

filename[i - 2] = 'e';

filename[i - 1] = 's';

FILE\* fp;

fp = fopen(filename, "w");

if (fp == NULL) {

printf("打开文件失败\n");

return ERROR;

}

fprintf(fp, "s %d\n", res); // res是求解结果，1表示满足，0表示不满足，-1未定

fprintf(fp, "v\n");

for (i = 1; i < An.length; i++) {

if (An.elem[i] == -1)

fprintf(fp, "%5d", -i);

else

fprintf(fp, "%5d", i);

if (i % 10 == 0)

fprintf(fp, "\n");

}

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

fclose(fp);

return OK;}

# 优化后程序

优化后的程序包含main.cpp,cnfs.h,new.h,head.h,res.h,game.h

“head.h”

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <math.h>

#include <iostream>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

typedef struct lNode {

int l; //存储文字

int mark; //标记该文字是否被删除，初始值为1，表示没有被删除

struct lNode\* next;

} lNode, \*plNode; //存储一个文字节点

typedef struct cNode {

int l\_count; //存储一个子句的文字个数

int flag; //记录该子句因为flag而被删除，初始值为0，flag为需要删除的文字

int mark; //标记该子句是否被删除 ，初始值为1，表示没有被删除

lNode\* firstl; //指向第一个文字节点

struct cNode\* next; //指向下一条子句

} cNode, \*pcNode; //子句头节点

typedef struct cnf {

int varinum; //变元个数

int claunum; //子句个数

cNode\* firstc; //指向第一个子句节点

} cnf, \*pcnf; //一个cnf公式

typedef struct SqList {//用于存储答案的顺序表的

int\* elem;

int length;

} SqList;

#define SIZE 9

SqList Bn;

int choose;

int key[10000];

int value[10000];

int createCNF(pcnf\* L, char filename[]);//创建CNF文件

int showCNF(pcnf L); //打印CNF公式

int InitList(pcnf L,SqList &An);//初始化存储答案的顺序表

int RemoveClause(pcnf L,int flag);//删除含有文字的子句

int AddClause(pcnf L,int flag);//恢复含有文字的子句

int EmptyClause(pcnf L);//判断子句是否为空

int DPLL(pcnf L,SqList &An,int now\_l);//DPLL

int ResWrite(int res,double time,SqList &An,char filename[]);//将求解的结果写入到res

int option2(pcnf L);//策略选择

int reverse(pcnf L, int flag);//换用文字的负数

int option1(int \*m);//策略选择

int creatsuduko(int temp[][9]);//

void printsuduko(int arr[][9]);//打印数独

int get\_sudoku(int temp[SIZE][SIZE]);//建立数独

void cleanup(int temp[SIZE][SIZE]);//清空数独

void output\_sudoku(int temp[SIZE][SIZE]);//输出数独

int check\_sudoku(int temp[SIZE][SIZE], int i, int j, int k);//检查填入数字是否合法

int sd(int sudo[9][9]);//合法性判断

int digsudu(int shudu[9][9], int num, int beifen[9][9]);//挖洞

int check(int shudu[9][9]);//检查是否完成数独游戏

int savecnf(int shudu[9][9], char\* filename, int dignum);//数独保存为CNF

int sudures(SqList& An, char filename[],int beifen);//DPLL求解结果写入到res

“main.cpp”

#include "head.h"

#include "cnfs.h"

#include "res.h"

#include "new.h"

#include "game.h"

int main() {

pcnf L;

SqList An;

int suc = 1;

int shudu[9][9];

clock\_t start, end; //记录开始时间与结束时间

double duration; //记录程序运行时间

char filename[200]; //文件名

int op = 1, tag; //菜单选择

int b;

int num;

int dignum = 0;

double time;

double s; //记录效率的提升

while (op) {

system("cls");

printf(" 基于SAT的数独游戏求解程序的菜单\n");

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

printf(" 1.读取cnf文件 2.展示cnf文件\n");

printf(" 3.DPLL求解算例 4.保存为res\n");

printf(" 5.数独游戏 0.退出\n");

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

printf("请选择你要进行的操作[0-5]:\n");

scanf("%d", &op);

switch (op) {

case 1: //创建cnf结构体

if (Bn.elem)

{

free(Bn.elem);

Bn.elem = NULL;

free(An.elem);

An.elem = NULL;

destory(L);

}

printf("请输入文件名:\n");

scanf("%s", filename);

if (createCNF(&L, filename) == OK) {

printf("cnf创建成功\n");

InitList(L, An);

InitList(L, Bn);

Bn.length = 0;

sort(L, key, value);

}

else

printf("cnf创建失败\n");

getchar();

getchar();

break;

case 2: //显示cnf结构

showCNF(L);

getchar();

getchar();

break;

case 3: // DPLL算法

start = clock();

tauto(L);

printf("请选择分裂策略：\n");

printf("策略1：按照文字出现次数优先\t策略2：按照子句的长度优先\n");

scanf("%d", &choose);

if (DPLL(L, An) == OK) {

tag = 1;

printf("该算例为可满足算例\n");

} else {

tag = 0;

printf("该算例为不可满足算例\n");

}

end = clock();

duration = ((double)(end - start)) / CLK\_TCK \* 1000;

printf("运行时间为：%f毫秒\n", duration);

for (int i = 1; i < An.length;i++)

{

if (An.elem[i] == -1)

printf("%d ", -i);

else

printf("%d ", i);

}

getchar();

getchar();

break;

case 4:

if (ResWrite(tag, duration, An, filename) == OK)

printf("结果写入成功\n");

else

printf("结果写入失败\n");

getchar();

getchar();

break;

case 5:

int option;

option = 1;

int beifen[9][9];

suc = 1;

while (suc) {

system("cls");

printf(" \t数独游戏 \n");

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

printf(" 1.生成数独棋盘 2.挖洞\n");

printf(" 3.游戏 4.保存为cnf\n");

printf(" 5.使用DPLL求解 6.输出答案\n");

printf(" 0.Exit\n");

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

printf("请选择你要进行的操作[0-6]:\n");

scanf("%d", &suc);

switch (suc) {

case 1:

if (sd(shudu) == OK)

printf("随机数独生成成功！\n");

output\_sudoku(shudu);

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

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

beifen[i][j] = shudu[i][j];

getchar();

getchar();

break;

case 2:

printf("请输入需要挖去的个数（一般不超过30）\n");

scanf("%d", &dignum);

if (digsudu(shudu, dignum,beifen) == OK)

printf("挖洞法生成数独成功！\n");

else

printf("挖洞法生成数独失败！\n");

output\_sudoku(shudu);

getchar();

getchar();

break;

case 3:

while (option) {

system("cls");

output\_sudoku(shudu);

printf("请输入想要填入的行列号：(例如1 1)\n");

int row, col;

scanf("%d %d", &row, &col);

while(shudu[row - 1][col - 1] != 0) {

printf("请正确输入行列数!\n");

printf("请输入想要填入的行列号：(例如1 1)\n");

scanf("%d %d", &row, &col);

}

printf("请输入要填入的数字：\n");

int state;

scanf("%d", &state);

if (check\_sudoku(shudu, row-1, col-1, state) == OK){

shudu[row-1][col-1] = state;

printf("恭喜填入正确！\n");

if (check(shudu) == OK) { printf("你是个狠人！");

break;

}

}

else{

printf("填入错误！\n");

}

printf("你还想继续填吗？（1/0）:");

scanf("%d", &option);

}

getchar();

getchar();

break;

case 4:

printf("请输入文件名:\n");

scanf("%s", filename);

if (savecnf(shudu, filename,dignum) == OK)

printf("保存为cnf成功");

else

printf("保存失败");

getchar();

getchar();

break;

case 5:

printf("请输入文件名:\n");

scanf("%s", filename);

if (createCNF(&L, filename) == OK) {

printf("cnf创建成功\n");

InitList(L, An);

InitList(L, Bn);

Bn.length = 0;

sort(L, key, value);

} else

printf("cnf创建失败\n");

start = clock();

tauto(L);

printf("请选择分裂策略：\n");

printf(

"策略1：按照文字出现次数优先\t策略2：按照子句的长度优先\n");

scanf("%d", &choose);

if (DPLL(L, An) == OK) {

tag = 1;

printf("该算例为可满足算例\n");

} else {

tag = 0;

printf("该算例为不可满足算例\n");

}

end = clock();

duration = ((double)(end - start)) / CLK\_TCK \* 1000;

printf("运行时间为：%f毫秒\n", duration);

if (sudures(An, filename,beifen) == OK)

printf("保存数独答案成功！");

else {

printf("未能成功保存答案！");

}

getchar();

getchar();

break;

case 6:

output\_sudoku(beifen);

getchar();

getchar();

break;

case 0:

break;

}

}

getchar();

getchar();

break;

case 0:

break;

} // end of switch

} // end of while

return 0;

}

“cnfs.h”

int createCNF(pcnf\* L, char filename[200]) { //创建结构体，以表的形式存储

pcNode p; //子句结点

plNode q; //文字节点

\*L = (cnf\*)malloc(sizeof(struct cnf));

p = (cNode\*)malloc(sizeof(struct cNode));

p->flag = 0; //记录子句是否因为flag而被删除

p->mark = 1; //记录子句是否被删除，1表示未被删除

(\*L)->firstc = p;

(\*L)->claunum = -1;

FILE\* fp;

int num = 0; //为计算子句数目和变元数目的中间变量

int k = 0; //记录每个子句文字个数

int m = 0; //记录子句的个数

int flag; //记录正负

int lnum=0;

char c; //读取文件字符使用

fp = fopen(filename, "r");

if (fp == NULL) {

printf("打开文件失败\n");

return 0;

}

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

while (!feof(fp)) {

if (c == 'c') { //把c开头的注释去掉

while (c != '\n') {

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

}

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

} else if (c == 'p') {

while (c != '\n') {

while (c < '0' || c > '9') {

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

}

while (c >= '0' && c <= '9') { //获取变元数

num = num \* 10;

num += c - '0';

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

}

(\*L)->varinum = num;

num = 0;

while (c < '0' || c > '9') {

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

}

while (c >= '0' && c <= '9') { //获取子句数

num = num \* 10;

num += c - '0';

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

}

(\*L)->claunum = num;

lnum = 0;

}

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

} else if ((c >= '0' && c <= '9') || (c == '-')) {

q = (lNode\*)malloc(sizeof(struct lNode));

q->mark = 1; // mark等于1表示没有被删除

p->firstl = q;

while (c != '\n') {

while (c != '0') {

while (c != ' ') {

flag = 1;

while ((c >= '0' && c <= '9') || (c == '-')) {

if (c == '-') {

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

flag = 0;

}

lnum = lnum \* 10;

lnum += c - '0';

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

}

if (flag == 0)

q->l = -lnum;

else

q->l = lnum;

lnum = 0;

k++;

}

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

if (c == '0')

q->next = NULL;

else {

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

q = q->next;

q->mark = 1;

}

}

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

}

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

p->l\_count = k;

k = 0;

m++;

if (m < (\*L)->claunum) {

p->next = (cNode\*)malloc(sizeof(struct cNode));

p = p->next;

p->flag = 0;

p->mark = 1;

} else

p->next = NULL;

} else

fscanf(fp, "%c", &c); //防止文件结尾有多个换行

}

fclose(fp);

if ((\*L)->claunum!=num) {

printf("文件内容不合法！\n");

return FALSE;

}

return OK;

}

void destory(pcnf L) {

pcNode p = L->firstc;

pcNode pn;

plNode qn;

if (p == NULL)

return;

else {

while (p) {

plNode q = p->firstl;

while (q) {

qn = q->next;

free(q);

q = qn;

}

pn = p->next;

free(p);

p = pn;

}

}

L->firstc=p;

L->claunum = 0;

L->varinum=0;

return;

}

int InitList(pcnf L, SqList& An) { //初始化结果接收顺序表

An.elem = (int\*)malloc((L->varinum + 1) \* sizeof(int)); //分配存储空间

if (!An.elem) {

return ERROR; //分配存储空间失败

};

An.length = L->varinum + 1; //初始化线性表的长度为0

for (int i = 0; i < An.length; i++) {

An.elem[i] = 0; //表示没有经过单子句简化赋值

}

return OK;

}

int showCNF(pcnf L) { //显示cnf结构体

if (L->claunum == 0) {

printf("cnf公式为空\n");

return OK;

}

pcNode p = L->firstc;

plNode q = p->firstl;

printf("cnf公式变元数：%d 子句数：%d\n", L->varinum, L->claunum);

int i = 1;

while (p) {

if (p->mark == 0)

p = p->next;

else {

printf("第%d句:", i++);

q = p->firstl;

while (q) {

if (q->mark == 0)

q = q->next;

else {

printf("%5d",q->l);

q = q->next;

}

}

printf("\n");

p = p->next;

}

}

return OK;

}

void quickSort(int\* array, int\* arraynum, int low, int high) { //快排

int i, j, key1, key;

i = low;

j = high;

key1 = array[low];//表示第几位

key = arraynum[low];//权重

if (low < high) {

while (i < j) {

while (i<j&&arraynum[j] <= key)

j--;

if (i < j) {

array[i] = array[j];

arraynum[i] = arraynum[j];

}

while (i < j && arraynum[i] >= key)

i++;

if (i < j) {

array[j] = array[i];

arraynum[j] = arraynum[i];

}

}

array[i] = key1;

arraynum[i] = key;

int standard = i;

quickSort(array, arraynum, low, standard - 1);

quickSort(array, arraynum, standard + 1, high);

}

}

void sort(pcnf L, int key[3500], int value[3500]) {

memset(value, 0, sizeof(value));

for (int i = 0; i < L->varinum; i++)

key[i] = i + 1;

pcNode p = L->firstc;

int k;

while (p) {

if (p->l\_count<10)

k = 100-p->l\_count\*10;

else

k = 1;

plNode q = p->firstl;

for (int i = 1; i <= p->l\_count; i++) {

int m = q->l;

if (m > 0)

value[m - 1] += k;

else {

value[-m - 1] += k;

}

q = q->next;

}

p = p->next;

}

quickSort(key, value, 0, L->varinum - 1);

}

“new.h”

#pragma once

int RemoveClause(pcnf L, int flag) { //一次找出一个变元，并删除

pcNode p;

plNode q;

p = L->firstc;

while (p) {

if (p->mark == 0) {

p = p->next;

continue;

}

else {

q = p->firstl;

while (q) {

if (q->mark == 0) {

q = q->next;

continue;

}

else {

if (q->l == flag) { //子句中有flag，删除子句

p->mark = 0;

p->flag = flag;

L->claunum--; //子句数目减一

break;

} else if (q->l == -flag) { //子句中有-flag，删除该文字

q->mark = 0;

p->l\_count--; //文字数目减一

break;

} else

q = q->next;

}

}

p = p->next;

}

}

L->varinum--; //删除变元后，变元数目减一

return OK;

}

int AddClause(pcnf L, int flag) { //恢复前面因为flag而删除的元素，子句

pcNode p;

plNode q;

p = L->firstc;

while (p) {

q = p->firstl;

if (p->mark == 1) {

//恢复非L规则删除的文字

q = p->firstl;

while (q) {

if (q->mark == 0 && q->l == -flag) {

q->mark = 1;

p->l\_count++; //子句文字数目加一

break;

} else

q = q->next;

}

} else if (p->mark == 0 && p->flag == flag) { //恢复子句

p->mark = 1;

p->flag = 0;

L->claunum++;

} else {

while (q) {

if (q->mark == 0 && q->l == -flag) {

q->mark = 1;

p->l\_count++; //子句文字数目加一

break;

} else

q = q->next;

}

}

p = p->next;

}

L->varinum++; //变元数目加一

return OK;

}

int EmptyClause(pcnf L) { //查看是否有空句

pcNode p = L->firstc;

while (p) {

if (p->l\_count == 0 && p->mark == 1)

return OK; //当文字数为0，且子句未经过删减，则是空句

p = p->next;

}

return FALSE;

}

void tauto(pcnf L) {

pcNode p = L->firstc;

plNode q, qn;

while (p) {

q = p->firstl;

while (q) {

qn = q->next;

while (qn!=NULL) {

if (qn->l == -q->l) {

p->mark = 0;

p->flag = 0;

L->claunum--;

break;

}

qn = qn->next;

}

if (p->mark == 0)

break;

q = q->next;

}

p = p->next;

}

}

int single(pcnf L) {

pcNode p;

p = L->firstc;

while(p) {

if(p->l\_count==1&&p->mark==1 ) {

plNode q;

q = p->firstl;

while (q) {

if (q->mark == 1)

return q->l;

else {

q = q->next;

}

}

}

p = p->next;

}

return FALSE;

}

int DPLL(pcnf L, SqList& An) {

if (L->claunum == 0)

return TRUE;

if (EmptyClause(L)==OK)

return FALSE;

int tar =single(L);

int b = Bn.length;

int m;

while (tar) {

Bn.elem[Bn.length++] = tar;

RemoveClause(L, tar);

if (L->claunum == 0) {

for(int i=0;i<Bn.length;i++){

int temp = Bn.elem[i];

if (temp > 0)

An.elem[temp] = 1;

else

An.elem[-temp] = -1;

}

Bn.length = 0;

return OK;

} else if (EmptyClause(L)==OK) {

for (int i = Bn.length- 1; i >= b; i--)

AddClause(L, Bn.elem[i]);

Bn.length = b;

return FALSE;

}

tar = single(L);

}

if (choose == 1) {

tar = option1(&m);

}

else if (choose == 2) {

tar = option2(L);

}

RemoveClause(L, tar);

if (DPLL(L, An)==OK) {

if (tar > 0)

An.elem[tar] = 1;

else {

An.elem[-tar] = -1;

}

for(int i=0;i<Bn.length;i++) {

int temp = Bn.elem[i];

if (temp > 0)

An.elem[temp] = 1;

else

An.elem[-temp] = -1;

}

Bn.length = 0;

return OK;

}

tar = -tar;

reverse(L, tar);

if (DPLL(L, An)) {

if (tar > 0)

An.elem[tar] = 1;

else {

An.elem[-tar] = -1;

}

for (int i = 0; i < Bn.length; i++) {

int temp = Bn.elem[i];

if (temp > 0)

An.elem[temp] = 1;

else

An.elem[-temp] = -1;

}

Bn.length

= 0;

return OK;

}

AddClause(L, tar);

if (choose == 1)

value[m] = 1;

for (int i = Bn.length - 1; i >=b;i--)

AddClause(L, Bn.elem[i]);

Bn.length = b;

return FALSE;

}

int reverse(pcnf L, int flag) {

pcNode p = L->firstc;

while (p) {

if (p->flag == -flag) {

p->flag = 0;

p->mark = 1;

p->l\_count--;

plNode q = p->firstl;

while (q->l != -flag)

q = q->next;

q->mark = 0;

L->claunum++;

} else if (p->mark == 1) {

plNode q = p->firstl;

while (q) {

if (q->l == flag) {

p->flag = flag;

p->mark = 0;

q->mark = 1;

p->l\_count++;

L->claunum--;

break;

}

q = q->next;

}

}

p = p->next;

}

return OK;

}

int option1(int \*m) {

int i = 0;

while (1) {

if (value[i]) {

int j = 0;

for (; j < Bn.length; j++) {

if (key[i] == abs(Bn.elem[j]))

break;

}

if (j == Bn.length) {

\*m = i;

value[i] = 0;

return key[i];

}

}

i++;

}

}

int option2(pcnf L) {

int i;

pcNode p;

plNode q;

for(i=2;i;i++) {

p = L->firstc;

while (p) {

if (p->l\_count == i && p->mark == 1) {

q = p->firstl;

while (q) {

if (q->mark == 1)

return q->l;

q = q->next;

}

}

p = p->next;

}

}

}

“res.h”

#pragma once

int ResWrite(int res, double time, SqList& An, char filename[]) {

int i = 0;

while (filename[i] != '\0')

i++;

filename[i - 3] = 'r'; //只改变文件的扩展名

filename[i - 2] = 'e';

filename[i - 1] = 's';

FILE\* fp;

fp = fopen(filename, "w");

if (fp == NULL) {

printf("打开文件失败\n");

return ERROR;

}

fprintf(fp, "s %d\n", res); // res是求解结果，1表示满足，0表示不满足，-1未定

fprintf(fp, "v ");

for (i = 1; i < An.length; i++) {

if (An.elem[i] == -1)

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

else

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

}

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

fclose(fp);

return OK;

}

int sudures(SqList& An, char filename[],int beifen[9][9]) {

int i = 0;

while (filename[i] != '\0')

i++;

filename[i - 3] = 'r'; //只改变文件的扩展名

filename[i - 2] = 'e';

filename[i - 1] = 's';

FILE\* fp;

fp = fopen(filename, "w");

if (fp == NULL) {

printf("打开文件失败\n");

return ERROR;

}

for (i = 1; i < An.length; i++) {

if (An.elem[i] == 1) {

for(int row=0;row<9;row++)

for (int cal = 0; cal < 9; cal++)

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

// if ((i==row\*81+cal\*9+k)&&k==beifen[row][cal])

if ((i == row \* 81 + cal \* 9 + k))

fprintf(fp, "第%d行第%d列该填入的数为：%d \n", row + 1, cal + 1,

k);

}

}

}

fclose(fp);

return OK;

}

“game.h”

void output\_sudoku(int temp[9][9]) {

int i, j;

//数独输出到屏幕

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

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

if (j % 3 == 0)

printf("|");

if (temp[i][j]!=0)

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

else {

printf("\_ ");

}

if(j==8)

printf("\n");

}

if ((i + 1) % 3 == 0)

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

}

}

//使数独全部清零

void cleanup(int temp[SIZE][SIZE]) {

int i, j;

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

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

temp[i][j] = 0;

}

}

}

int sd(int sudo[9][9]) {

int i, n;

srand(time(NULL));

for (i = 0; i < 1; i++) {

do {

cleanup(sudo);

} while (

get\_sudoku(sudo) ==

0); //如果数独生成失败，返回结果为0，则清空sudo数组，继续再试直到成功

}

return OK;

}

int get\_sudoku(int temp[SIZE][SIZE]) {

int i, j, m, random;

random = rand() % SIZE + 1; //随机生成数独第一行第一列的数 //循环每一行

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

//循环每一列

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

m = 0;

//判断此随机数是否能够使用,如果能使用跳出循环，否则继续循环

while (check\_sudoku(temp, i, j, random) == 0) {

random = rand() % SIZE + 1; //生成一个0-SIZE的随机数

m++;

//如果同一个位置失败一百次，则重新开始

if (m > 80)

return 0;

}

temp[i][j] = random; //此数已经通过检查，放入数独

}

}

return 1;

}

//判断是否可以将第i行、第j列的数设为k

int check\_sudoku(int temp[SIZE][SIZE], int i, int j, int k) {

int m, n;

//判断行

for (n = 0; n < SIZE; n++) {

if (temp[i][n] == k)

return 0;

}

//判断列

for (m = 0; m < SIZE; m++) {

if (temp[m][j] == k)

return 0;

}

//判断所在九宫格

int t1 = (i / 3) \* 3, t2 = (j / 3) \* 3;

for (m = t1; m < t1 + 3; m++) {

for (n = t2; n < t2 + 3; n++) {

if (temp[m][n] == k)

return 0;

}

}

return 1;

}

//挖洞

int digsudu(int shudu[9][9], int num,int beifen[9][9]) {

srand(time(NULL));

int flag;

for (; num > 0; ) {

int i = rand() % 9;

int j = rand() % 9;

flag = 0;

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

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

if (k == beifen[i][j]) {

k++;

} else if (check\_sudoku(shudu, i, j, k) == OK) {

break;

flag=1;

}

}

if (flag == 0) {

shudu[i][j] = 0;

num--;

}

} else {

continue;

}

}

return OK;

}

int savecnf(int shudu[9][9],char \*filename,int dignum) {

FILE\* fp;

fp= fopen(filename, "w");

if (fp == NULL) {

printf("打开文件失败\n");

return 0;

}

int litnum=729, claunum=7533+81-dignum;

fprintf(fp, "p %d %d\n", litnum, claunum);

int i, j, k, m;

//行约束

//每行都有1-9

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

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

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

fprintf(fp, "%d ", (i-1)\*81 +(j-1)\*9 + k);//每一个空格都可以填1-9个数

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

};

} //每一行

//729

//不能同时有两个数字

for ( i = 1; i <= 9; i++) { //每一行

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

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

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

fprintf(fp, "-%d -%d ",

(i - 1) \* 81 + (j - 1) \* 9 + k,(i - 1) \* 81 + (j - 1+m) \* 9 + k); //每一个空格都可以填1-9个数

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

}

}

}

//列约束

for (k = 1; k <= 9; k++) { //每一列

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

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

fprintf(fp, "%d ",

(i-1)\*81+ (j-1)\*9 + k); //每一个空格都可以填1-9个数

fprintf(fp, "0\n");

}

}//729

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

{ //每一行

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

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

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

fprintf(

fp, "-%d -%d ", (i - 1) \*81+ (j - 1) \*9 + k,

(i - 1) \* 81 + (j - 1) \* 9 + k + m\*81);

fprintf(fp, "0\n");

}

}

}

}

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

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

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

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

for (i = 4; i <= 6; i++)

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

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

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

for (j = 4; j <= 6; j++)

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

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

for (j = 7; j <= 9; j++)

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

for (i = 4; i <= 6; i++)

for (j = 4; j <= 6; j++)

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

for (i = 4; i <= 6; i++)

for (j = 7; j <= 9; j++)

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

for (i = 7; i <= 9; i++)

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

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

for (i = 7; i <= 9; i++)

for (j = 4; j <= 6; j++)

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

for (i = 7; i <= 9; i++)

for (j = 7; j <= 9; j++)

fprintf(fp, "%d ", (i - 1) \* 81 + (j - 1) \* 9 + k);

fprintf(fp, "0\n");

}

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

{

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

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

for(int x=0;x<3;x++)

for(int y=0;y<3;y++)

for(int k=x+1;k<3;k++)

for (int l = 0; l < 3; l++)

{

if (y != l)

fprintf(

fp, "-%d -%d 0\n",

(3 \* i + x ) \* 81 + (3 \* j + y) \* 9 + z,

(3 \* i + k ) \* 81 + (3 \* j + l) \* 9 + z);

}

}

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

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

if (shudu[i][j] != 0)

fprintf(fp, "%d 0\n",i\*81+j\*9+shudu[i][j]);

}

fclose(fp);

return OK;

}

//检查是否完成数独游戏

int check(int shudu[9][9])

{

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

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

if (shudu[i][j] == 0)

return FALSE;

return OK;

}